

# Babel

Version 3.36.1844

2019/12/03

*Original author*

Johannes L. Braams

*Current maintainer*

Javier Bezos

The standard distribution of  $\text{\LaTeX}$  contains a number of document classes that are meant to be used, but also serve as examples for other users to create their own document classes. These document classes have become very popular among  $\text{\LaTeX}$  users. But it should be kept in mind that they were designed for American tastes and typography. At one time they even contained a number of hard-wired texts.

This manual describes babel, a package that makes use of the capabilities of  $\text{\TeX}$ , xetex and luatex to provide an environment in which documents can be typeset in a language other than US English, or in more than one language or script.

Current development is focused on Unicode engines (Xe $\text{\TeX}$  and Lua $\text{\TeX}$ ) and the so-called *complex scripts*. New features related to font selection, bidi writing, line breaking and so on are being added incrementally.

Babel provides support (total or partial) for about 200 languages, either as a “classical” package option or as an ini file. Furthermore, new languages can be created from scratch easily.

# Contents

<b>I</b>	<b>User guide</b>	<b>4</b>
<b>1</b>	<b>The user interface</b>	<b>4</b>
1.1	Monolingual documents . . . . .	4
1.2	Multilingual documents . . . . .	5
1.3	Modifiers . . . . .	6
1.4	xelatex and luatex . . . . .	7
1.5	Troubleshooting . . . . .	8
1.6	Plain . . . . .	8
1.7	Basic language selectors . . . . .	8
1.8	Auxiliary language selectors . . . . .	9
1.9	More on selection . . . . .	10
1.10	Shorthands . . . . .	11
1.11	Package options . . . . .	14
1.12	The base option . . . . .	16
1.13	ini files . . . . .	17
1.14	Selecting fonts . . . . .	24
1.15	Modifying a language . . . . .	26
1.16	Creating a language . . . . .	27
1.17	Digits . . . . .	29
1.18	Getting the current language name . . . . .	30
1.19	Hyphenation and line breaking . . . . .	30
1.20	Selecting scripts . . . . .	32
1.21	Selecting directions . . . . .	33
1.22	Language attributes . . . . .	37
1.23	Hooks . . . . .	37
1.24	Languages supported by babel with ldf files . . . . .	38
1.25	Unicode character properties in luatex . . . . .	39
1.26	Tweaking some features . . . . .	40
1.27	Tips, workarounds, known issues and notes . . . . .	40
1.28	Current and future work . . . . .	41
1.29	Tentative and experimental code . . . . .	42
<b>2</b>	<b>Loading languages with language.dat</b>	<b>43</b>
2.1	Format . . . . .	43
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>44</b>
3.1	Guidelines for contributed languages . . . . .	45
3.2	Basic macros . . . . .	45
3.3	Skeleton . . . . .	47
3.4	Support for active characters . . . . .	48
3.5	Support for saving macro definitions . . . . .	48
3.6	Support for extending macros . . . . .	48
3.7	Macros common to a number of languages . . . . .	49
3.8	Encoding-dependent strings . . . . .	49
<b>4</b>	<b>Changes</b>	<b>53</b>
4.1	Changes in babel version 3.9 . . . . .	53
<b>II</b>	<b>Source code</b>	<b>53</b>
<b>5</b>	<b>Identification and loading of required files</b>	<b>53</b>

<b>6</b>	<b>locale directory</b>	<b>54</b>
<b>7</b>	<b>Tools</b>	<b>54</b>
7.1	Multiple languages . . . . .	58
<b>8</b>	<b>The Package File (<math>\LaTeX</math>, babel.sty)</b>	<b>59</b>
8.1	base . . . . .	60
8.2	key=value options and other general option . . . . .	62
8.3	Conditional loading of shorthands . . . . .	63
8.4	Language options . . . . .	65
<b>9</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>67</b>
9.1	Tools . . . . .	67
9.2	Hooks . . . . .	70
9.3	Setting up language files . . . . .	72
9.4	Shorthands . . . . .	74
9.5	Language attributes . . . . .	83
9.6	Support for saving macro definitions . . . . .	86
9.7	Short tags . . . . .	86
9.8	Hyphens . . . . .	87
9.9	Multiencoding strings . . . . .	88
9.10	Macros common to a number of languages . . . . .	94
9.11	Making glyphs available . . . . .	94
9.11.1	Quotation marks . . . . .	95
9.11.2	Letters . . . . .	96
9.11.3	Shorthands for quotation marks . . . . .	97
9.11.4	Umlauts and tremas . . . . .	98
9.12	Layout . . . . .	99
9.13	Load engine specific macros . . . . .	100
9.14	Creating languages . . . . .	100
<b>10</b>	<b>The kernel of Babel (babel.def for <math>\LaTeX</math>only)</b>	<b>111</b>
10.1	The redefinition of the style commands . . . . .	111
10.2	Cross referencing macros . . . . .	111
10.3	Marks . . . . .	115
10.4	Preventing clashes with other packages . . . . .	116
10.4.1	ifthen . . . . .	116
10.4.2	varioref . . . . .	117
10.4.3	hhline . . . . .	117
10.4.4	hyperref . . . . .	117
10.4.5	fancyhdr . . . . .	118
10.5	Encoding and fonts . . . . .	118
10.6	Basic bidi support . . . . .	120
10.7	Local Language Configuration . . . . .	123
<b>11</b>	<b>Multiple languages (switch.def)</b>	<b>124</b>
11.1	Selecting the language . . . . .	125
11.2	Errors . . . . .	133
<b>12</b>	<b>Loading hyphenation patterns</b>	<b>135</b>
<b>13</b>	<b>Font handling with fontspec</b>	<b>140</b>

<b>14</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>144</b>
14.1	XeTeX . . . . .	144
14.2	Layout . . . . .	147
14.3	LuaTeX . . . . .	148
14.4	Southeast Asian scripts . . . . .	154
14.5	CJK line breaking . . . . .	158
14.6	Auto bidi with basic and basic-r . . . . .	165
<b>15</b>	<b>Data for CJK</b>	<b>176</b>
<b>16</b>	<b>The ‘nil’ language</b>	<b>176</b>
<b>17</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>177</b>
17.1	Not renaming hyphen.tex . . . . .	177
17.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	178
17.3	General tools . . . . .	178
17.4	Encoding related macros . . . . .	182
<b>18</b>	<b>Acknowledgements</b>	<b>185</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	4
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	5
You are loading directly a language style . . . . .	8
Unknown language ‘LANG’ . . . . .	8
Argument of \language@active@arg” has an extra } . . . . .	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’ . . . . .	26
Package babel Warning: The following fonts are not babel standard families . .	26

## Part I

# User guide

- This user guide focuses on  $\LaTeX$ . There are also some notes on its use with Plain  $\TeX$ .
- Changes and new features with relation to version 3.8 are highlighted with **New X.XX**. The most recent features could be still unstable. Please, report any issues you find in <https://github.com/latex3/babel/issues>, which is better than just complaining on an e-mail list or a web forum.
- If you are interested in the  $\TeX$  multilingual support, please join the kadingira list on <http://tug.org/mailman/listinfo/kadingira>. You can follow the development of babel in <https://github.com/latex3/babel> (which provides some sample files, too).
- See section 3.1 for contributing a language.
- The first sections describe the traditional way of loading a language (with `ldf` files). The alternative way based on `ini` files, which complements the previous one (it will *not* replace it), is described below.

## 1 The user interface

### 1.1 Monolingual documents

In most cases, a single language is required, and then all you need in  $\LaTeX$  is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings.

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

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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

```
!_Paragraph_ended_before_\\UTFviii@three@octets_was_complete.
```

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

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

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

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

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

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

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

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

## 1.2 Multilingual documents

In multilingual documents, just use 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  $\text{\LaTeX}$ , the preamble of the document:

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

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

You can also set the main language explicitly, but it is discouraged except if there a real reason to do so:

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

Examples of cases where `main` is useful are the following.

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

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

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

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

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

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

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

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
\usepackage[utf8]{inputenc}  
  
\usepackage[english,french]{babel}  
  
\begin{document}  
  
Plus ça change, plus c'est la même chose!  
  
\selectlanguage{english}  
  
And an English paragraph, with a short text in  
\foreignlanguage{french}{français}.  
  
\end{document}
```

### 1.3 Modifiers

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

---

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

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

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

## 1.4 xelatex and lualatex

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

The Latin script is covered by default in current  $\text{\LaTeX}$  (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** The following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and `\today` in Danish and Vietnamese. No additional packages are required.

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename{}_\alsoname{}_\today

\selectlanguage{vietnamese}

\prefacename{}_\alsoname{}_\today

\end{document}
```

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

```
\documentclass{article}

\usepackage[russian]{babel}

\babelfont{rm}{DejaVuSerif}

\begin{document}

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

\end{document}
```



## 1.5 Troubleshooting

- Loading directly sty files in 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.

**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** `{\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).

## 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\*}** `{\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}`  $\langle language \rangle$  ... `\end{hyphenrules}`

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

## 1.9 More on selection

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

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

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

**EXAMPLE** With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

## 1.10 Shorthands

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

**NOTE** Note the following:

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

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

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**     $\{\langle shorthands-list \rangle\}$   
**\shorthandoff**    $\star\{\langle shorthands-list \rangle\}$

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.

**\useshorthands**    $\star\{\langle char \rangle\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

**\aliasshorthand** `{⟨original⟩}{⟨alias⟩}`

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character `/` over `"` in typing Polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`.

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

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

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

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

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

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

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

<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.

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

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

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

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

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

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

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

## 1.11 Package options

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

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

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

<b>KeepShorthandsActive</b>	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
<b>activeacute</b>	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
<b>activegrave</b>	Same for `.
<b>shorthands=</b>	<p><math>\langle char \rangle \langle char \rangle \dots</math>   off</p> <p>The only language shorthands activated are those given, like, eg:</p> <pre style="background-color: #f0f0f0; padding: 10px;">\usepackage[esperanto,french,shorthands=:;!]{babel}</pre> <p>If ' is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \string (otherwise they will be expanded by <math>\LaTeX</math> before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.</p>
<b>safe=</b>	<p>none   ref   bib</p> <p>Some <math>\LaTeX</math> macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \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 <b>New 3.34</b>, in <math>\epsilon\TeX</math> based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).</p>
<b>math=</b>	<p>active   normal</p> <p>Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like <math>\{a'\}</math> (a closing brace after a shorthand) are not a source of trouble anymore.</p>
<b>config=</b>	<p><math>\langle file \rangle</math></p> <p>Load <math>\langle file \rangle.cfg</math> instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).</p>
<b>main=</b>	<p><math>\langle language \rangle</math></p> <p>Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.</p>
<b>headfoot=</b>	<p><math>\langle language \rangle</math></p> <p>By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.</p>



- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoiled by an unexpected .cfg file. However, if the key config is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** **New 3.9l** Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.
- silent** **New 3.9l** No warnings and no *infos* are written to the log file.<sup>9</sup>
- strings=** generic | unicode | encoded | *<label>* | *<font encoding>*  
 Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional T<sub>E</sub>X, LICR and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUppercase and the like (this feature misuses some internal L<sup>A</sup>T<sub>E</sub>X tools, so use it only as a last resort).
- hyphenmap=** off | main | select | other | other\*  
**New 3.9g** Sets the behavior of case mapping for hyphenation, provided the language defines it.<sup>10</sup> It can take the following values:  
**off** deactivates this feature and no case mapping is applied;  
**first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it's the default if a single language option has been stated;<sup>11</sup>  
**select** sets it only at \selectlanguage;  
**other** also sets it at otherlanguage;  
**other\*** also sets it at otherlanguage\* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other\* for monolingual documents.<sup>12</sup>
- bidi=** default | basic | basic-r | bidi-l | bidi-r  
**New 3.14** Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.21.
- layout=** **New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.21.

## 1.12 The base option

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

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

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

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

<sup>12</sup>Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.

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

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

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

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

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

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

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

## 1.13 ini files

An alternative approach to define a language is by means of an `ini` file. Currently `babel` provides about 200 of these files containing the basic data required for a language. Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them currently (by means of `\babelprovide`), but a higher interface, based on package options, is under development (in other words, `\babelprovide` is mainly intended for auxiliary tasks).

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

```
\documentclass{book}

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

\babelfont{rm}{DejaVuSans}

\begin{document}

\tableofcontents

\chapter{სამზარეულოდასუფრისტრადიციები}

ქართულიტრადიციულისამზარეულოერთ-ერთიუმდიდრესიამთვედმსოფლიოში.
```

```
\end{document}
```

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

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

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

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

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

Other Indic scripts are still under development in luatex. On the other hand, xetex is better.

**Southeast scripts** Thai works in both luatex and xetex, but line breaking differs (rules can be modified in luatex; they are hardcoded in xetex). Lao seems to work, too, but there are no patterns for the latter in luatex. Some quick patterns could help, with something similar to:

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{1໐1໙1໒1໑1໓1໒1໑}໒%Random
```

Khemer clusters are rendered wrongly.

**East Asia scripts** Settings for either Simplified or Traditional should work out of the box. `luatex` does basic line breaking, but currently `xetex` does not (you may load `zhspacing`). Although for a few words and short texts the `ini` files should be fine, CJK texts are best set with a dedicated framework (CJK, `luatexja`, `kotex`, `CtEx`, etc.). This is what the class `ltjbook` does with `luatex`, which can be used in conjunction with the `ldf` for `japanese`, because the following piece of code loads `luatexja`:

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

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

af	Afrikaans <sup>ul</sup>	asa	Asu
agq	Aghem	ast	Asturian <sup>ul</sup>
ak	Akan	az-Cyrl	Azerbaijani
am	Amharic <sup>ul</sup>	az-Latn	Azerbaijani
ar	Arabic <sup>ul</sup>	az	Azerbaijani <sup>ul</sup>
ar-DZ	Arabic <sup>ul</sup>	bas	Basaa
ar-MA	Arabic <sup>ul</sup>	be	Belarusian <sup>ul</sup>
ar-SY	Arabic <sup>ul</sup>	bem	Bemba
as	Assamese	bez	Bena

bg	Bulgarian <sup>ul</sup>	gl	Galician <sup>ul</sup>
bm	Bambara	gsw	Swiss German
bn	Bangla <sup>ul</sup>	gu	Gujarati
bo	Tibetan <sup>u</sup>	guz	Gusii
brx	Bodo	gv	Manx
bs-Cyrl	Bosnian	ha-GH	Hausa
bs-Latn	Bosnian <sup>ul</sup>	ha-NE	Hausa <sup>l</sup>
bs	Bosnian <sup>ul</sup>	ha	Hausa
ca	Catalan <sup>ul</sup>	haw	Hawaiian
ce	Chechen	he	Hebrew <sup>ul</sup>
cgg	Chiga	hi	Hindi <sup>u</sup>
chr	Cherokee	hr	Croatian <sup>ul</sup>
ckb	Central Kurdish	hsb	Upper Sorbian <sup>ul</sup>
cs	Czech <sup>ul</sup>	hu	Hungarian <sup>ul</sup>
cy	Welsh <sup>ul</sup>	hy	Armenian
da	Danish <sup>ul</sup>	ia	Interlingua <sup>ul</sup>
dav	Taita	id	Indonesian <sup>ul</sup>
de-AT	German <sup>ul</sup>	ig	Igbo
de-CH	German <sup>ul</sup>	ii	Sichuan Yi
de	German <sup>ul</sup>	is	Icelandic <sup>ul</sup>
dje	Zarma	it	Italian <sup>ul</sup>
dsb	Lower Sorbian <sup>ul</sup>	ja	Japanese
dua	Duala	jgo	Ngomba
dyo	Jola-Fonyi	jmc	Machame
dz	Dzongkha	ka	Georgian <sup>ul</sup>
ebu	Embu	kab	Kabyle
ee	Ewe	kam	Kamba
el	Greek <sup>ul</sup>	kde	Makonde
en-AU	English <sup>ul</sup>	kea	Kabuverdianu
en-CA	English <sup>ul</sup>	khq	Koyra Chiini
en-GB	English <sup>ul</sup>	ki	Kikuyu
en-NZ	English <sup>ul</sup>	kk	Kazakh
en-US	English <sup>ul</sup>	kkj	Kako
en	English <sup>ul</sup>	kl	Kalaallisut
eo	Esperanto <sup>ul</sup>	kln	Kalenjin
es-MX	Spanish <sup>ul</sup>	km	Khmer
es	Spanish <sup>ul</sup>	kn	Kannada <sup>ul</sup>
et	Estonian <sup>ul</sup>	ko	Korean
eu	Basque <sup>ul</sup>	kok	Konkani
ewo	Ewondo	ks	Kashmiri
fa	Persian <sup>ul</sup>	ksb	Shambala
ff	Fulah	ksf	Bafia
fi	Finnish <sup>ul</sup>	ksh	Colognian
fil	Filipino	kw	Cornish
fo	Faroese	ky	Kyrgyz
fr	French <sup>ul</sup>	lag	Langi
fr-BE	French <sup>ul</sup>	lb	Luxembourgish
fr-CA	French <sup>ul</sup>	lg	Ganda
fr-CH	French <sup>ul</sup>	lkt	Lakota
fr-LU	French <sup>ul</sup>	ln	Lingala
fur	Friulian <sup>ul</sup>	lo	Lao <sup>ul</sup>
fy	Western Frisian	lrc	Northern Luri
ga	Irish <sup>ul</sup>	lt	Lithuanian <sup>ul</sup>
gd	Scottish Gaelic <sup>ul</sup>	lu	Luba-Katanga

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

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

---

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

---

aghem	brazilian
akan	breton
albanian	british
american	bulgarian
amharic	burmese
arabic	canadian
arabic-algeria	cantonese
arabic-DZ	catalan
arabic-morocco	centralatlastamazight
arabic-MA	centralkurdish
arabic-syria	chechen
arabic-SY	cherokee
armenian	chiga
assamese	chinese-hans-hk
asturian	chinese-hans-mo
asu	chinese-hans-sg
australian	chinese-hans
austrian	chinese-hant-hk
azerbaijani-cyrillic	chinese-hant-mo
azerbaijani-cyrl	chinese-hant
azerbaijani-latin	chinese-simplified-hongkongsarchina
azerbaijani-latn	chinese-simplified-macausarchina
azerbaijani	chinese-simplified-singapore
bafia	chinese-simplified
bambara	chinese-traditional-hongkongsarchina
basaa	chinese-traditional-macausarchina
basque	chinese-traditional
belarusian	chinese
bemba	cognian
bena	cornish
bengali	croatian
bodo	czech
bosnian-cyrillic	danish
bosnian-cyrl	duala
bosnian-latin	dutch
bosnian-latn	dzongkha
bosnian	embu

english-au  
english-australia  
english-ca  
english-canada  
english-gb  
english-newzealand  
english-nz  
english-unitedkingdom  
english-unitedstates  
english-us  
english  
esperanto  
estonian  
ewe  
ewondo  
faroese  
filipino  
finnish  
french-be  
french-belgium  
french-ca  
french-canada  
french-ch  
french-lu  
french-luxembourg  
french-switzerland  
french  
friulian  
fulah  
galician  
ganda  
georgian  
german-at  
german-austria  
german-ch  
german-switzerland  
german  
greek  
gujarati  
gusii  
hausa-gh  
hausa-ghana  
hausa-ne  
hausa-niger  
hausa  
hawaiian  
hebrew  
hindi  
hungarian  
icelandic  
igbo  
inarisami  
indonesian  
interlingua

irish  
italian  
japanese  
jolafonyi  
kabuverdianu  
kabyle  
kako  
kalaallisut  
kalenjin  
kamba  
kannada  
kashmiri  
kazakh  
khmer  
kikuyu  
kinyarwanda  
konkani  
korean  
koyraborosenni  
koyrachiini  
kwasio  
kyrgyz  
lakota  
langi  
lao  
latvian  
lingala  
lithuanian  
lowersorbian  
lsorbian  
lubakatanga  
luo  
luxembourgish  
luyia  
macedonian  
machame  
makhuwameetto  
makonde  
malagasy  
malay-bn  
malay-brunei  
malay-sg  
malay-singapore  
malay  
malayalam  
maltese  
manx  
marathi  
masai  
mazanderani  
meru  
meta  
mexican  
mongolian

morisyen  
mundang  
nama  
nepali  
newzealand  
ngiemboon  
ngomba  
norsk  
northernluri  
northernsami  
northndebele  
norwegianbokmal  
norwegiannynorsk  
nswissgerman  
nuer  
nyankole  
nynorsk  
occitan  
oriya  
oromo  
ossetic  
pashto  
persian  
piedmontese  
polish  
portuguese-br  
portuguese-brazil  
portuguese-portugal  
portuguese-pt  
portuguese  
punjabi-arab  
punjabi-arabic  
punjabi-gurmukhi  
punjabi-guru  
punjabi  
quechua  
romanian  
romansh  
rombo  
rundi  
russian  
rwa  
sakha  
samburu  
samin  
sango  
sangu  
sanskrit-beng  
sanskrit-bengali  
sanskrit-deva  
sanskrit-devanagari  
sanskrit-gujarati  
sanskrit-gujr  
sanskrit-kannada

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



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

## 1.14 Selecting fonts

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

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

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

If no language is given, then it is considered the default font for the family, activated when a language is selected. On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, *\*devanagari*).

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

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

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska␣\foreignlanguage{hebrew}{עִבְרִית}␣svenska.

\end{document}
```

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

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

```
\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:

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

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

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

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

This makes sure the OpenType script for Devanagari is `deva` and not `dev2` (luatex does not detect automatically the correct script<sup>14</sup>). 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 (nor `Language`). In fact, it is even discouraged.

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

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

**WARNING** Do not use `\setxxxxfont` and `\babelfont` at the same time. `\babelfont` follows the standard  $\TeX$  conventions to set the basic families – define `\xxdefault`, and activate it with `\xxfamily`. On the other hand, `\setxxxxfont` in `fontspec` takes a different approach, because `\xxfamily` is redefined with the family name hardcoded (so that `\xxdefault` becomes no-op). Of course, both methods are incompatible, and if you use `\setxxxxfont`, font switching with `\babelfont` just does *not* work (nor the standard `\xxdefault`, for that matter). As of [New 3.34](#) there is an attempt to make them compatible, but the language system will not be set by babel and should be set with `fontspec` if necessary.

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

**TROUBLESHOOTING** *Package fontspec Warning: 'Language 'LANG' not available for font 'FONT' with script 'SCRIPT' 'Default' language used instead'. This is **not** and error.* This warning is shown by fontspec, not by babel. It could be irrelevant for English, but not for many other languages, including Urdu and Turkish. This is a useful and harmless warning, and if everything is fine with your document the best thing you can do is just to ignore it altogether.

**TROUBLESHOOTING** *Package babel Warning: The following fonts are not babel standard families. This is **not** and error.* The main purpose of `\babel font` 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, `\babel font{rm}{FreeSerif}` defines 4 fonts (with their variants, of course), which are switched with the language by babel. It's just a tool to make things easier and transparent to the user.

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

babel assumes that if you are using `\babel font` 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.

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 `\babel font` at all. But you must be aware that this may lead to some problems. And this is the very reason of the warning.

## 1.15 Modifying a language

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

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

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

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

- 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 `\babel provide` and its key import, is:

```
\renewcommand\spanishchaptername{Foo}
```

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

```
\addto\extrasrussian{\mymacro}
```

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

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

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

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

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

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

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

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

## 1.16 Creating a language

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

`\babelprovide` [ $\langle options \rangle$ ] { $\langle language-name \rangle$ }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Loads only the strings. For example:

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

**hyphenrules=** *<language-list>*

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

```
\babelprovide[hyphenrules=chavacano_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 TeX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with `\babelpatterns`, as for example:

```
\babelprovide[hyphenrules=+]{neo}
\babelpatterns[neo]{a1_e1_i1_o1_u1}
```

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

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

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

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

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

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

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

**mapfont=** direction

Assigns the font for the writing direction of this language (only with bidi=basic).<sup>15</sup> 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.<sup>16</sup> So, there should be at most 3 directives of this kind.

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

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

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

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

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

## 1.17 Digits

**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

<sup>15</sup>There will be another value, language, not yet implemented.

<sup>16</sup>In future releases a new value (script) will be added.

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

**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).

## 1.18 Getting the current language name

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

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

`\iflanguage` `{⟨language⟩}{⟨true⟩}{⟨false⟩}`

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

**WARNING** The advice about `\language` also applies here – use `iflang` instead of `\iflanguage` if possible.

## 1.19 Hyphenation and line breaking

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

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

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

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

There are also some differences with  $\TeX$ : (1) the character used is that set for the current font, while in  $\TeX$  it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in  $\TeX$ , but it can be changed to another value by redefining `\babelexhyphen`; (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 `\babelexpatterns` (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>17</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*).

## 1.20 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.<sup>18</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>19</sup>

**\ensureascii** { *<text>* }

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

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

<sup>17</sup>With *luatex* exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and babel only provides the most basic tools.

<sup>18</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>19</sup>But still defined for backwards compatibility.

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

## 1.21 Selecting directions

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

**WARNING** The current code for **text** in luatex should be considered essentially stable, but, of course, it is not bug-free and there could be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including the picture environment (with pict2e) and pfg/tikz. Also, indexes and the like are under study, as well as math (there is progress in the latter, too, but for example cases may fail).

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

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.

**New 3.32** There is some experimental support for harftex. Since it is based on luatex, the option basic mostly works. You may need to deactivate the rtlm or the rtla font features (besides loading harfload before babel and activating mode=harf; there is a sample in the GitHub repository).

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

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

```
\documentclass{article}

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

```

\babelprovide[import,main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

وقد عرف شبه جزيرة العرب طيلة العصر الهليني (الغريقي) بـ
Aravia أو Arabia (بالغريقية  $\alpha\rho\alpha\beta$ )، استخدم الرومان ثلاث
بدايات 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[mapfont=direction]{arabic}

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

\begin{document}

Most Arabic speakers consider the two varieties to be two registers
of one language, although the two registers can be referred to in
Arabic as فصحى العصر  $\textit{fuṣḥā} - 'aṣr$  (MSA) and
فصحى التراث  $\textit{fuṣḥā} - t-turāth$  (CA).

\end{document}

```

In this example, and thanks to `mapfont=direction`, 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{<subsection>.<section>}`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks  $>9$  with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it could depend on the counter format.

With counters, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while `1.2` in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is `c2.c1`. Of course, you may always adjust the order by changing the language, if necessary.<sup>20</sup>

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

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

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

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

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

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

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

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

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

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

---

<sup>20</sup>Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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

**\babelsublr**  $\{\langle lr\text{-}text\rangle\}$

Digits in pdf<sub>tex</sub> must be marked up explicitly (unlike luat<sub>ex</sub> with `bidi=basic` or `bidi=basic-r` and, usually, xet<sub>ex</sub>). This command is provided to set  $\{\langle lr\text{-}text\rangle\}$  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**  $\{\langle section\text{-}name\rangle\}$

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

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

**New 3.17** Something like:

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

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

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

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

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

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

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

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

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

## 1.22 Language attributes

`\languageattribute`

This is a user-level command, to be used in the preamble of a document (after `\usepackage[...]{babel}`), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language. Very often, using a *modifier* in a package option is better. Several language definition files use their own methods to set options. For example, french uses `\frenchsetup`, magyar (1.5) uses `\magyarOptions`; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, `\ProsodicMarksOn` in latin).

## 1.23 Hooks

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

`\AddBabelHook`

```
[<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  $\text{\TeX}$  parameters (`#1`, `#2`, `#3`), with the meaning given:

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

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

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

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

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

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

**write** This event comes just after the switching commands are written to the aux file.  
**beforeextras** Just before executing `\extras<language>`. This event and the next one should not contain language-dependent code (for that, add it to `\extras<language>`).  
**afterextras** Just after executing `\extras<language>`. For example, the following deactivates shorthands in all languages:

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

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

```
\AddBabelHook{myhook}{stringprocess}{%  

\protected@edef\BabelString{\BabelString}}
```

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

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

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

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

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

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

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

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

## 1.24 Languages supported by babel with ldf files

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

**Afrikaans** afrikaans  
**Azerbaijani** azerbaijani  
**Basque** basque  
**Breton** breton  
**Bulgarian** bulgarian  
**Catalan** catalan  
**Croatian** croatian  
**Czech** czech  
**Danish** danish  
**Dutch** dutch

**English** english, USenglish, american, UKenglish, british, canadian, australian, newzealand  
**Esperanto** esperanto  
**Estonian** estonian  
**Finnish** finnish  
**French** french, francais, canadien, acadian  
**Galician** galician  
**German** austrian, german, germanb, ngerman, naustrian  
**Greek** greek, polutonikogreek  
**Hebrew** hebrew  
**Icelandic** icelandic  
**Indonesian** bahasa, indonesian, indon, bahasai  
**Interlingua** interlingua  
**Irish Gaelic** irish  
**Italian** italian  
**Latin** latin  
**Lower Sorbian** lowersorbian  
**Malay** bahasam, malay, melayu  
**North Sami** samin  
**Norwegian** norsk, nynorsk  
**Polish** polish  
**Portuguese** portuges, portuguese, brazilian, brazil  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** uppersorbian  
**Welsh** welsh

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

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

```

\documentclass{article}
\usepackage[hindi]{babel}
\begin{document}
{\dn_devaanaa.m_priya.h}
\end{document}

```

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

## 1.25 Unicode character properties in luatex

**New 3.32** Part of the babel job is to apply Unicode rules to some script-specific features based on some properties. Currently, they are 3, namely, direction (ie, bidi class), mirroring



glyphs, and line breaking for CJK scripts. These properties are stored in lua tables, which you can modify with the following macro (for example, to set them for glyphs in the PUA).

**`\babelcharproperty`** `{⟨char-code⟩}[⟨to-char-code⟩]{⟨property⟩}{⟨value⟩}`

**New 3.32** Here, `{⟨char-code⟩}` 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. 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
```

This command is allowed only in vertical mode (the preamble or between paragraphs).

## 1.26 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  $\text{\LaTeX}$ ), 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  $\text{\LaTeX}$  you may need `bidi.mirroring=off`. Use with care, because these options do not deactivate other related options (like paragraph direction with `bidi.text`).

## 1.27 Tips, workarounds, known issues and notes

- If you use the document class *book* and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`),  $\text{\LaTeX}$  will keep complaining about an undefined label. To prevent such problems, you could revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the safe option to `none` or `bib`.
- Both  $\text{\LaTeX}$  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 ( $\text{\LaTeX}$ ); (2) make it unactive (your settings); (3) make babel shorthands active (babel); (4) reload `hline` (babel, now with the correct catcodes for `|` and `:`).

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because T<sub>E</sub>X only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>21</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 T<sub>E</sub>X, not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the .aux file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make T<sub>E</sub>X 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.

## 1.28 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.<sup>22</sup> But that is the easy part, because they don't require modifying the L<sup>A</sup>T<sub>E</sub>X internals.

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

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

<sup>21</sup>This explains why L<sup>A</sup>T<sub>E</sub>X assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingsphcodes` is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.

<sup>22</sup>See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to T<sub>E</sub>X because their aim is just to display information and not fine typesetting.

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

## 1.29 Tentative and experimental code

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

### Modifying, and adding, values of ini files

**New 3.37** There is a way to modify the values of ini files when they get loaded with \babelprovide. To set, say, digits.native in the numbers section, use something like numbers..digits.native=abcdefghijklj (note the double dot between the section and the key name). New keys may be added, too.

### Non-standard hyphenation

**New 3.37** With luatex it is now possible to define non-standard hyphenation rules, like f-f → ff-f. No rules are currently provided by default, but they can be defined as shown in the following example:

```
\babelposthyphenation{ngerman}{([fmtrp])_|_|_{}
{
  _|_{}no_|_{}{1},_|_{}pre_|_{}{1}{1}-,
  _|_{}remove,
  _|_{}{}
}
```

See the babel wiki for a description and some examples:

<https://github.com/latex3/babel/wiki>

### Old stuff

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

- \babelFSstore{<babel-language>} sets the current three basic families (rm, sf, tt) as the default for the language given.
- \babelFSdefault{<babel-language>}{<fontspec-features>} patches \fontspec so that the given features are always passed as the optional argument or added to it (not an ideal solution).

So, for example:

```
\setmainfont[Language=Turkish]{Minion_|_Pro}
\babelFSstore{turkish}
\setmainfont{Minion_|_Pro}
\babelFSfeatures{turkish}{Language=Turkish}
```

## 2 Loading languages with `language.dat`

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

**New 3.9q** With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).<sup>23</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>24</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>25</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_what_files_with_patterns_to_load.
english_english.hyphenations
=british

dutch_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>26</sup> For example:

```
german:T1_hyphenT1.ger
german_hyphen.ger
```

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

A typical error when using babel is the following:

```
No_hyphenation_patterns_were_preloaded_for
the_language_`<lang>'_into_the_format.
Please,_configure_your_TeX_system_to_add_them_and
```

<sup>23</sup>This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

<sup>24</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>25</sup>This is because different operating systems sometimes use very different file-naming conventions.

<sup>26</sup>This is not a new feature, but in former versions it didn't work correctly.

```
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  $\text{\LaTeX}$  (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.

- Avoid adding things to `\noextras⟨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>27</sup>
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by `babel` and it can lead to incompatibilities.

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

### 3.1 Guidelines for contributed languages

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

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

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

### 3.2 Basic macros

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

- `\addlanguage` The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. For older versions of `plain.tex` and `lplain.tex` a substitute definition is used. Here “language” is used in the TeX sense of set of hyphenation patterns.
- `\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the `babel` system is to define

<sup>27</sup>But not removed, for backward compatibility.

`\<lang>hyphenmins`

this language as a ‘dialect’ of the language for which the patterns were loaded as `\language0`. Here “language” is used in the  $\TeX$  sense of set of hyphenation patterns. The macro `\<lang>hyphenmins` is used to store the values of the `\lefthyphenmin` and `\righthyphenmin`. Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:

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

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

`\providehyphenmins`

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

`\captions<lang>`

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

`\date<lang>`

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

`\extras<lang>`

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

`\noextras<lang>`

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

`\bbl@declare@ttribute`

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

`\main@language`

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

`\ProvidesLanguage`

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

`\LdfInit`

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

`\ldf@quit`

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

`\ldf@finish`

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

`\loadlocalcfg`

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

`\substitutefontfamily`

(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct  $\LaTeX$  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>}
\begin{document}
\ldefinit{<language>}{captions<language>}

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

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

\bbl@declareattribute{<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>}
%\MoreStrings

\StartBabelCommands*{<language>}{date}
\SetString\monthinamerica{<name_of_first_month>}
%\MoreStrings

\StartBabelCommands*{<dialect>}{captions}
\SetString\chaptername{<chapter_name>}
%\MoreStrings

\StartBabelCommands*{<dialect>}{date}
\SetString\monthinamerica{<name_of_first_month>}
%\MoreStrings

\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}%
\Delaypackage
```



```

\savebox{\myeye}{\eye}}%AAAAAAAAAnd_direct_usage
\newsavebox{\myeye}
\newcommand\myanchor{\anchor}%AAAABut_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` The command `\bbl@activate` is used to change the way an active character expands.

`\bbl@deactivate` `\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` 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.” [2, p. 380]  
`\bbl@remove@special` 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>28</sup>.

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

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

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

### 3.6 Support for extending macros

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

<sup>28</sup>This mechanism was introduced by Bernd Raichle.

### 3.7 Macros common to a number of languages

<code>\bbl@allowhyphens</code>	In several languages compound words are used. This means that when $\TeX$ has to hyphenate such a compound word, it only does so at the ‘-’ that is used in such words. To allow hyphenation in the rest of such a compound word, the macro <code>\bbl@allowhyphens</code> can be used.
<code>\allowhyphens</code>	Same as <code>\bbl@allowhyphens</code> , but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with <code>\accent</code> in OT1. Note the previous command ( <code>\bbl@allowhyphens</code> ) has different applications (hyphens and discretionary) than this one (composite chars). Note also prior to version 3.7, <code>\allowhyphens</code> had the behavior of <code>\bbl@allowhyphens</code> .
<code>\set@low@box</code>	For some languages, quotes need to be lowered to the baseline. For this purpose the macro <code>\set@low@box</code> is available. It takes one argument and puts that argument in an <code>\hbox</code> , at the baseline. The result is available in <code>\box0</code> for further processing.
<code>\save@sf@q</code>	Sometimes it is necessary to preserve the <code>\spacefactor</code> . For this purpose the macro <code>\save@sf@q</code> is available. It takes one argument, saves the current <code>\spacefactor</code> , executes the argument, and restores the <code>\spacefactor</code> .
<code>\bbl@frenchspacing</code> <code>\bbl@nonfrenchspacing</code>	The commands <code>\bbl@frenchspacing</code> and <code>\bbl@nonfrenchspacing</code> 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`  $\{ \langle \textit{language-list} \rangle \} \{ \langle \textit{category} \rangle \} [ \langle \textit{selector} \rangle ]$

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

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

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

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

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

The `<category>` is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.<sup>29</sup> It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
  \SetString{monthiiname}{Februar}
  \SetString{monthiiname}{M\"a}rz}
  \SetString{monthivname}{April}
  \SetString{monthvname}{Mai}
  \SetString{monthviname}{Juni}
  \SetString{monthviiname}{Juli}
  \SetString{monthviiiname}{August}
  \SetString{monthixname}{September}
  \SetString{monthxname}{Oktober}
  \SetString{monthxiiname}{November}
  \SetString{monthxiiiname}{Dezenber}
```

<sup>29</sup>In future releases further categories may be added.

```

\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  $\langle date \rangle \langle language \rangle$  exists).

**\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>30</sup>

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

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

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

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

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

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

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

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

```

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

```

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would typically be things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A  $\langle map-list \rangle$  is a series of macros using the internal format of `@uclclist` (eg, `\bb\BB\cc\CC`). The mandatory

<sup>30</sup>This replaces in 3.9g a short-lived `\UseStrings` which has been removed because it did not work.

arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in  $\text{\TeX}$ , we could set for Turkish:

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

\StartBabelCommands{turkish}{}[unicode,fontenc=TU_EU1_EU2,charset=utf8]
\SetCase
\lccode\lccode`i=\I\relax
\lccode\lccode`1=\I\relax
\lccode\lccode`İ=\i\relax
\lccode\lccode`I=\i\relax

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

## `\SetHyphenMap` `{(to-lower-macros)}`

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

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

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

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

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

## 4 Changes

### 4.1 Changes in babel version 3.9

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

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

## Part II

## Source code

babel is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use babel only as documented (except, of course, if you want to explore and test them – you can post suggestions about multilingual issues to [kadingira@tug.org](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 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. By default it also loads `switch.def`.

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

## 6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

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

**version** of the ini file

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

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

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

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

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

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

## 7 Tools

```
1 <<version=3.36.1844>>
2 <<date=2019/12/03>>
```

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

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

```
3 <<(*Basic macros)>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
```

```

7 \bbl@ifunset{\bbl@stripslash#1}%
8 {\def#1{#2}}%
9 {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
14 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
15 \def\bbl@loop#1#2#3,{%
16 \ifx\@nnil#3\relax\else
17 \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
18 \fi}
19 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}

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

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

\bbl@afterelse \bbl@afterfi Because the code that is used in the handling of active characters may need to look ahead,
we take extra care to ‘throw’ it over the \else and \fi parts of an \if-statement31. These
macros will break if another \if... \fi statement appears in one of the arguments and it
is not enclosed in braces.

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

\bbl@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.

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

34 \def\bbl@tempa#1{%
35 \long\def\bbl@trim##1##2{%
36 \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
37 \def\bbl@trim@c{%
38 \ifx\bbl@trim@a\@sptoken
39 \expandafter\bbl@trim@b
40 \else

```

<sup>31</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.



```

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

```

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

```

**\bb1@ifblank** A tool from url, by Donald Arseneau, which tests if a string is empty or space.

```

67 \def\bb1@ifblank#1{%
68 \bb1@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
69 \long\def\bb1@ifblank@i#1#2\@nil#3#4#5\@nil{#4}

```

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

```

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

```

81 \def\bb1@vforeach#1#2{%
82 \def\bb1@forcmd##1{#2}%
83 \bb1@fornext#1,\@nil,}
84 \def\bb1@fornext#1,{%
85 \ifx\@nil#1\relax\else

```

```

86 \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
87 \expandafter\bbl@fornext
88 \fi}
89 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

\bbl@replace

```

90 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
91 \toks@{}}%
92 \def\bbl@replace@aux##1#2##2#2{%
93 \ifx\bbl@nil##2%
94 \toks@\expandafter{\the\toks@##1}%
95 \else
96 \toks@\expandafter{\the\toks@##1#3}%
97 \bbl@afterfi
98 \bbl@replace@aux##2#2%
99 \fi}%
100 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
101 \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 retokenized). It may change! (or even merged with \bbl@replace; I'm not sure checking the replacement is really necessary or just paranoia).

```

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

```

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

```

127 \def\bbl@ifsamestring#1#2{%
128 \begingroup
129 \protected@edef\bbl@tempb{#1}%

```

```

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

```

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

```

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

```

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

```

157 <<*Load patterns in luatex>> ≡
158 \ifx\directlua\@undefined\else
159 \ifx\bbl@luapatterns\@undefined
160 \input luababel.def
161 \fi
162 \fi
163 <</Load patterns in luatex>>

```

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

```

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

```

## 7.1 Multiple languages

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

```

169 <<*Define core switching macros>> ≡
170 \ifx\language\@undefined
171 \csname newcount\endcsname\language

```

```

172 \fi
173 <</Define core switching macros>>

```

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

`\addlanguage` To add languages to T<sub>E</sub>X's memory plain T<sub>E</sub>X version 3.0 supplies `\newlanguage`, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`. For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefore `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain T<sub>E</sub>X version 3.0.

For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`. Plain T<sub>E</sub>X version 3.0 uses `\count 19` for this purpose.

```

174 <<*Define core switching macros>> ≡
175 \ifx\newlanguage\@undefined
176   \csname newcount\endcsname\last@language
177   \def\addlanguage#1{%
178     \global\advance\last@language\@ne
179     \ifnum\last@language<\@ccclvi
180       \else
181         \errmessage{No room for a new \string\language!}%
182       \fi
183     \global\chardef#1\last@language
184     \wlog{\string#1 = \string\language\the\last@language}}
185 \else
186   \countdef\last@language=19
187   \def\addlanguage{\alloc@9\language\chardef\@ccclvi}
188 \fi
189 <</Define core switching macros>>

```

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

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

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

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

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

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

## 8.1 base

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

```
190 (*package)
191 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
192 \ProvidesPackage{babel}[\langle\date\rangle\langle\version\rangle The Babel package]
193 \@ifpackagewith{babel}{debug}
194   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
195   \let\bbl@debug\@firstofone}
196   {\providecommand\bbl@trace[1]{}%
197   \let\bbl@debug\@gobble}
198 \ifx\bbl@switchflag\undefined % Prevent double input
199   \let\bbl@switchflag\relax
200   \input switch.def\relax
201 \fi
202 \langle\Load patterns in luatex\rangle
203 \langle\Basic macros\rangle
204 \def\AfterBabelLanguage#1{%
205   \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

```

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

```
206 \ifx\bbl@languages\undefined\else
207   \begingroup
208     \catcode\^^I=12
209     \@ifpackagewith{babel}{showlanguages}{%
210       \begingroup
211         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
212         \wlog{<*languages>}%
213         \bbl@languages
214         \wlog{</languages>}%
215       \endgroup}{%
216     \endgroup
217     \def\bbl@elt#1#2#3#4{%
218       \ifnum#2=\z@
219         \gdef\bbl@nulllanguage{#1}%
220         \def\bbl@elt##1##2##3##4{}%
221       \fi}%
222     \bbl@languages
223 \fi
224 \ifodd\bbl@engine
225   % Harftex is evolving, so the callback is not hardcoded, just in case
226   \def\bbl@harfprefline{Harf pre_linebreak_filter callback}%
227   \def\bbl@activate@preotf{%
228     \let\bbl@activate@preotf\relax % only once
229     \directlua{
230       Babel = Babel or {}
231       %
232       function Babel.pre_otfload_v(head)
233         if Babel.numbers and Babel.digits_mapped then
234           head = Babel.numbers(head)
235         end
236         if Babel.bidi_enabled then
237           head = Babel.bidi(head, false, dir)
238         end
239         return head
240       end

```

```

241 %
242 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
243   if Babel.numbers and Babel.digits_mapped then
244     head = Babel.numbers(head)
245   end
246   if Babel.bidi_enabled then
247     head = Babel.bidi(head, false, dir)
248   end
249   return head
250 end
251 %
252 luatexbase.add_to_callback('pre_linebreak_filter',
253   Babel.pre_otfload_v,
254   'Babel.pre_otfload_v',
255   luatexbase.priority_in_callback('pre_linebreak_filter',
256     '\bbl@harfpreamble')
257   or luatexbase.priority_in_callback('pre_linebreak_filter',
258     'luaotfload.node_processor')
259   or nil)
260 %
261 luatexbase.add_to_callback('hpack_filter',
262   Babel.pre_otfload_h,
263   'Babel.pre_otfload_h',
264   luatexbase.priority_in_callback('hpack_filter',
265     '\bbl@harfpreamble')
266   or luatexbase.priority_in_callback('hpack_filter',
267     'luaotfload.node_processor')
268   or nil)
269 }%
270 \@ifpackageloaded{harfload}%
271   {\directlua{ Babel.mirroring_enabled = false }}%
272   {}%
273 \let\bbl@tempa\relax
274 \@ifpackagewith{babel}{bidi=basic}%
275   {\def\bbl@tempa{basic}}%
276   {\@ifpackagewith{babel}{bidi=basic-r}%
277     {\def\bbl@tempa{basic-r}}%
278     {}%
279   \ifx\bbl@tempa\relax\else
280     \let\bbl@beforeforeign\leavevmode
281     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
282     \RequirePackage{luatexbase}%
283     \directlua{
284       require('babel-data-bidi.lua')
285       require('babel-bidi-\bbl@tempa.lua')
286     }
287     \bbl@activate@preotf
288   \fi
289 \fi

```

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

```

290 \bbl@trace{Defining option 'base'}
291 \@ifpackagewith{babel}{base}{%
292   \ifx\directlua\undefined
293     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
294   \else
295     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
296   \fi

```

```

297 \DeclareOption{base}{}%
298 \DeclareOption{showlanguages}{}%
299 \ProcessOptions
300 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
301 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
302 \global\let@ifl@ter@@\ifl@ter
303 \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter@ifl@ter@@}%
304 \endinput}{}%

```

## 8.2 key=value options and other general option

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

```

305 \bbl@trace{key=value and another general options}
306 \bbl@csarg\let\tempa\expandafter\csname opt@babel.sty\endcsname
307 \def\bbl@tempb#1.#2{%
308   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
309 \def\bbl@tempd#1.#2\@nnil{%
310   \ifx\@empty#2%
311     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
312   \else
313     \in@{=}{#1}\ifin@
314     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
315   \else
316     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
317     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
318   \fi
319 \fi}
320 \let\bbl@tempc\@empty
321 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
322 \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

```

The next option tells babel to leave shorthand characters active at the end of processing the package. This is *not* the default as it can cause problems with other packages, but for those who want to use the shorthand characters in the preamble of their documents this can help.

```

323 \DeclareOption{KeepShorthandsActive}{}
324 \DeclareOption{activeacute}{}
325 \DeclareOption{activegrave}{}
326 \DeclareOption{debug}{}
327 \DeclareOption{noconfigs}{}
328 \DeclareOption{showlanguages}{}
329 \DeclareOption{silent}{}
330 \DeclareOption{mono}{}
331 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
332 % Don't use. Experimental:
333 \newif\ifbbl@single
334 \DeclareOption{selectors=off}{\bbl@singletrue}
335 <<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.

```

336 \let\bbl@opt@shorthands\@nnil
337 \let\bbl@opt@config\@nnil
338 \let\bbl@opt@main\@nnil
339 \let\bbl@opt@headfoot\@nnil
340 \let\bbl@opt@layout\@nnil

```

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

```

341 \def\bbl@tempa#1=#2\bbl@tempa{%
342   \bbl@csarg\ifx{opt@#1}\@nnil
343     \bbl@csarg\edef{opt@#1}{#2}%
344   \else
345     \bbl@error{%
346       Bad option `#1=#2'. Either you have misspelled the\\%
347       key or there is a previous setting of `#1'}{%
348       Valid keys are `shorthands', `config', `strings', `main',\\%
349       `headfoot', `safe', `math', among others.}
350   \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.

```

351 \let\bbl@language@opts\@empty
352 \DeclareOption*{%
353   \bbl@xin@{\string=}{\CurrentOption}%
354   \ifin@
355     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
356   \else
357     \bbl@add@list\bbl@language@opts{\CurrentOption}%
358   \fi}

```

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

```

359 \ProcessOptions*

```

### 8.3 Conditional loading of shorthands

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

```

360 \bbl@trace{Conditional loading of shorthands}
361 \def\bbl@sh@string#1{%
362   \ifx#1\@empty\else
363     \ifx#1t\string~%
364     \else\ifx#1c\string,%
365     \else\string#1%
366   \fi\fi
367   \expandafter\bbl@sh@string
368 \fi}
369 \ifx\bbl@opt@shorthands\@nnil
370   \def\bbl@ifshorthand#1#2#3{#2}%
371 \else\ifx\bbl@opt@shorthands\@empty
372   \def\bbl@ifshorthand#1#2#3{#3}%
373 \else

```

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



```

374 \def\bbl@ifshorthand#1{%
375   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
376   \ifin@
377     \expandafter\@firstoftwo
378   \else
379     \expandafter\@secondoftwo
380   \fi}

```

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

```

381 \edef\bbl@opt@shorthands{%
382   \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.

```

383 \bbl@ifshorthand{'}%
384   {\PassOptionsToPackage{activeacute}{babel}}{}
385 \bbl@ifshorthand{`}%
386   {\PassOptionsToPackage{activegrave}{babel}}{}
387 \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.

```

388 \ifx\bbl@opt@headfoot\@nnil\else
389   \g@addto@macro\@resetactivechars{%
390     \set@typeset@protect
391     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
392     \let\protect\noexpand}
393 \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.

```

394 \ifx\bbl@opt@safe\@undefined
395   \def\bbl@opt@safe{BR}
396 \fi
397 \ifx\bbl@opt@main\@nnil\else
398   \edef\bbl@language@opts{%
399     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
400     \bbl@opt@main}
401 \fi

```

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

```

402 \bbl@trace{Defining IfBabelLayout}
403 \ifx\bbl@opt@layout\@nnil
404   \newcommand\IfBabelLayout[3]{#3}%
405 \else
406   \newcommand\IfBabelLayout[1]{%
407     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
408     \ifin@
409       \expandafter\@firstoftwo
410     \else
411       \expandafter\@secondoftwo
412     \fi}
413 \fi

```

## 8.4 Language options

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

```
414 \bbl@trace{Language options}
415 \let\bbl@afterlang\relax
416 \let\BabelModifiers\relax
417 \let\bbl@loaded@empty
418 \def\bbl@load@language#1{%
419   \InputIfFileExists{#1.ldf}%
420   {\edef\bbl@loaded{\CurrentOption
421     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
422     \expandafter\let\expandafter\bbl@afterlang
423       \csname\CurrentOption.ldf-h@@k\endcsname
424     \expandafter\let\expandafter\BabelModifiers
425       \csname bbl@mod@\CurrentOption\endcsname}%
426   {\bbl@error{%
427     Unknown option '\CurrentOption'. Either you misspelled it\\%
428     or the language definition file \CurrentOption.ldf was not found}{%
429     Valid options are: shorthands=, KeepShorthandsActive,\\%
430     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
431     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

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

```
432 \def\bbl@try@load@lang#1#2#3{%
433   \IfFileExists{\CurrentOption.ldf}%
434   {\bbl@load@language{\CurrentOption}}%
435   {#1\bbl@load@language{#2}#3}}
436 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
437 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
438 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
439 \DeclareOption{hebrew}{%
440   \input{rlbabel.def}%
441   \bbl@load@language{hebrew}}
442 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
443 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
444 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
445 \DeclareOption{polutonikogreek}{%
446   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
447 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
448 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
449 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
450 \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.

```
451 \ifx\bbl@opt@config\@nnil
452   \@ifpackagewith{babel}{noconfigs}{}%
453   {\InputIfFileExists{bblopts.cfg}%
454     {\typeout{*****^J%
455       * Local config file bblopts.cfg used^^J%
456       *}}%
457     {}}%
```

```

458 \else
459   \InputIfFileExists{\bbl@opt@config.cfg}%
460   {\typeout{*****^J%
461             * Local config file \bbl@opt@config.cfg used^^J%
462             *}}%
463   {\bbl@error{%
464     Local config file '\bbl@opt@config.cfg' not found}{%
465     Perhaps you misspelled it.}}%
466 \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.

```

467 \bbl@for\bbl@tempa\bbl@language@opts{%
468   \bbl@ifunset{ds@\bbl@tempa}%
469   {\edef\bbl@tempb{%
470     \noexpand\DeclareOption
471     {\bbl@tempa}%
472     {\noexpand\bbl@load@language{\bbl@tempa}}}%
473   \bbl@tempb}%
474   \@empty}

```

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

```

475 \bbl@foreach\@classoptionslist{%
476   \bbl@ifunset{ds@#1}%
477   {\IfFileExists{#1.ldf}%
478     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
479     {}}%
480   {}}

```

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

```

481 \ifx\bbl@opt@main\@nnil\else
482   \expandafter
483   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
484   \DeclareOption{\bbl@opt@main}{}
485 \fi

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (except, of course, global options, which  $\LaTeX$  processes before):

```

486 \def\AfterBabelLanguage#1{%
487   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
488 \DeclareOption*{}
489 \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`.

```

490 \ifx\bbl@opt@main\@nnil
491   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
492   \let\bbl@tempc\@empty

```

```

493 \bbl@for\bbl@tempb\bbl@tempa{%
494   \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
495   \ifin@edef\bbl@tempc{\bbl@tempb}\fi}
496 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
497 \expandafter\bbl@tempa\bbl@loaded,\@nnil
498 \ifx\bbl@tempb\bbl@tempc\else
499   \bbl@warning{%
500     Last declared language option is '\bbl@tempc',\%
501     but the last processed one was '\bbl@tempb'.\%
502     The main language cannot be set as both a global\%
503     and a package option. Use 'main=\bbl@tempc' as\%
504     option. Reported}%
505   \fi
506 \else
507   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
508   \ExecuteOptions{\bbl@opt@main}
509   \DeclareOption*{}
510   \ProcessOptions*
511 \fi
512 \def\AfterBabelLanguage{%
513   \bbl@error
514   {Too late for \string\AfterBabelLanguage}%
515   {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.

```

516 \ifx\bbl@main@language\undefined
517   \bbl@info{%
518     You haven't specified a language. I'll use 'nil'\%
519     as the main language. Reported}
520   \bbl@load@language{nil}
521 \fi
522 \</package>
523 \<core>

```

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

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `babel.def` contains most of the code, while `switch.def` defines the language-switching commands; both can be read at run time. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns (by default, it also inputs `switch.def`, for “historical reasons”, but it is not necessary). When `babel.def` is loaded it checks if the current version of `switch.def` is in the format; if not, it is loaded. A further file, `babel.sty`, contains  $\LaTeX$ -specific stuff. 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.

### 9.1 Tools

```

524 \ifx\ldf@quit\undefined

```

```

525 \else
526   \expandafter\endinput
527 \fi
528 <<Make sure ProvidesFile is defined>>
529 \ProvidesFile{babel.def}[\<<date>>] \<<version>> Babel common definitions]
530 <<Load macros for plain if not LaTeX>>

```

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

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

```

531 \ifx\bbl@ifshorthand\@undefined
532   \let\bbl@opt@shorthands\@nnil
533   \def\bbl@ifshorthand#1#2#3{#2}%
534   \let\bbl@language@opts\@empty
535   \ifx\babeloptionstrings\@undefined
536     \let\bbl@opt@strings\@nnil
537   \else
538     \let\bbl@opt@strings\babeloptionstrings
539   \fi
540   \def\BabelStringsDefault{generic}
541   \def\bbl@tempa{normal}
542   \ifx\babeloptionmath\bbl@tempa
543     \def\bbl@mathnormal{\noexpand\textormath}
544   \fi
545   \def\AfterBabelLanguage#1#2{}
546   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
547   \let\bbl@afterlang\relax
548   \def\bbl@opt@safe{BR}
549   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
550   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
551   \expandafter\newif\csname ifbbl@single\endcsname
552 \fi

```

And continue.

```

553 \ifx\bbl@switchflag\@undefined % Prevent double input
554   \let\bbl@switchflag\relax
555   \input switch.def\relax
556 \fi
557 \bbl@trace{Compatibility with language.def}
558 \ifx\bbl@languages\@undefined
559   \ifx\directlua\@undefined
560     \openin1 = language.def
561     \ifeof1
562       \closein1
563       \message{I couldn't find the file language.def}
564     \else
565       \closein1
566       \begingroup
567         \def\addlanguage#1#2#3#4#5{%
568           \expandafter\ifx\csname lang@#1\endcsname\relax\else
569             \global\expandafter\let\csname l@#1\endcsname\expandafter\endcsname
570             \csname lang@#1\endcsname
571           \fi}%
572         \def\uselanguage#1{}%
573         \input language.def
574       \endgroup

```

```

575 \fi
576 \fi
577 \chardef\l@english\z@
578 \fi
579 <<Load patterns in luatex>>
580 <<Basic macros>>

```

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

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

```

581 \def\addto#1#2{%
582   \ifx#1\@undefined
583     \def#1{#2}%
584   \else
585     \ifx#1\relax
586       \def#1{#2}%
587     \else
588       {\toks@\expandafter{#1#2}%
589        \xdef#1{\the\toks@}}%
590   \fi
591 \fi}

```

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

```

592 \def\bbl@withactive#1#2{%
593   \begingroup
594     \lccode`~=#2\relax
595     \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 L<sup>A</sup>T<sub>E</sub>X macros completely in case their definitions change (they have changed in the past).

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

```

596 \def\bbl@redefine#1{%
597   \edef\bbl@tempa{\bbl@stripslash#1}%
598   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
599   \expandafter\def\csname\bbl@tempa\endcsname}

```

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

```

600 \@onlypreamble\bbl@redefine

```

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

```

601 \def\bbl@redefine@long#1{%
602   \edef\bbl@tempa{\bbl@stripslash#1}%
603   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
604   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
605 \@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`.

```

606 \def\bbl@redefineroobust#1{%
607   \edef\bbl@tempa{\bbl@stripslash#1}%
608   \bbl@ifunset{\bbl@tempa\space}%
609   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
610     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
611   {\bbl@exp{\let\<org@\bbl@tempa\>\<\bbl@tempa\space>}}}%
612   \@namedef{\bbl@tempa\space}}

```

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

```

613 \@onlypreamble\bbl@redefineroobust

```

## 9.2 Hooks

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

```

614 \bbl@trace{Hooks}
615 \newcommand\AddBabelHook[3][{}{%
616   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}}%
617   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
618   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
619   \bbl@ifunset{bbl@ev@#2@#3@#1}%
620   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
621   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
622   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
623 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
624 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
625 \def\bbl@usehooks#1#2{%
626   \def\bbl@elt##1{%
627     \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1@#2}}%
628     \@nameuse{bbl@ev@#1@}%
629     \ifx\language\@undefined\else % Test required for Plain (?)
630       \def\bbl@elt##1{%
631         \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1@\language}#2}}%
632         \@nameuse{bbl@ev@#1@\language}%
633       \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).

```

634 \def\bbl@evargs{,% <- don't delete this comma
635   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
636   addialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
637   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
638   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
639   beforestart=0}

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This

part is somewhat involved because we have to make sure things are expanded the correct number of times.

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

```

640 \bbl@trace{Defining babelensure}
641 \newcommand\babelensure[2][{}]{% TODO - revise test files
642   \AddBabelHook{babel-ensure}{afterextras}{%
643     \ifcase\bbl@select@type
644       \@nameuse{\bbl@e@\language}{}%
645     \fi}%
646   \begingroup
647     \let\bbl@ens@include\@empty
648     \let\bbl@ens@exclude\@empty
649     \def\bbl@ens@fontenc{\relax}%
650     \def\bbl@tempb##1{%
651       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
652     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
653     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
654     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
655     \def\bbl@tempc{\bbl@ensure}%
656     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
657       \expandafter{\bbl@ens@include}}%
658     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
659       \expandafter{\bbl@ens@exclude}}%
660     \toks@\expandafter{\bbl@tempc}%
661     \bbl@exp{%
662       \endgroup
663       \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
664     \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
665       \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
666         \ifx##1\@undefined % 3.32 - Don't assume the macros exists
667           \edef##1{\noexpand\bbl@nocaption
668             {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
669         \fi
670         \ifx##1\@empty\else
671           \in@{##1}{#2}%
672         \ifin@ \else
673           \bbl@ifunset{\bbl@ensure@\language}%
674           {\bbl@exp{%
675             \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
676               \\\foreignlanguage{\language}%
677               {\ifx\relax#3\else
678                 \\\fontencoding{#3}\selectfont
679                 \fi
680               #####1}}}%
681           {}%
682           \toks@\expandafter{##1}%
683           \edef##1{%
684             \bbl@csarg\noexpand{ensure@\language}%
685             {\the\toks@}}%
686         \fi
687         \expandafter\bbl@tempb
688       \fi}%
689     \expandafter\bbl@tempb\bbl@captionslist\today\@empty

```



```

690 \def\bbl@tempa##1{% elt for include list
691   \ifx##1\@empty\else
692     \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
693     \ifin\else
694       \bbl@tempb##1\@empty
695       \fi
696     \expandafter\bbl@tempa
697   \fi}%
698 \bbl@tempa#1\@empty}
699 \def\bbl@captionslist{%
700 \prefacename\refname\abstractname\bibname\chaptername\appendixname
701 \contentsname\listfigurename\listtablename\indexname\figurename
702 \tablename\partname\enclname\ccname\headtoname\pagename\seename
703 \alsoname\proofname\glossaryname}

```

### 9.3 Setting up language files

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

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

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

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

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

Finally we check `\originalTeX`.

```

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

```

```

723 \ldf@quit{#1}%
724 \fi
725 \else
726 \expandafter\ifx\csname#2\endcsname\relax\else
727 \ldf@quit{#1}%
728 \fi
729 \fi
730 \bbl@ldfinit}

```

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

```

731 \def\ldf@quit#1{%
732 \expandafter\main@language\expandafter{#1}%
733 \catcode\@=\atcatcode \let\atcatcode\relax
734 \catcode\==\eqcatcode \let\eqcatcode\relax
735 \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.

```

736 \def\bbl@afterldf#1{%
737 \bbl@afterlang
738 \let\bbl@afterlang\relax
739 \let\BabelModifiers\relax
740 \let\bbl@screset\relax}%
741 \def\ldf@finish#1{%
742 \loadlocalcfg{#1}%
743 \bbl@afterldf{#1}%
744 \expandafter\main@language\expandafter{#1}%
745 \catcode\@=\atcatcode \let\atcatcode\relax
746 \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  $\LaTeX$ .

```

747 \@onlypreamble\LdfInit
748 \@onlypreamble\ldf@quit
749 \@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.

```

750 \def\main@language#1{%
751 \def\bbl@main@language{#1}%
752 \let\language\bbl@main@language
753 \bbl@id@assign
754 \bbl@patterns{\language}}

```

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

```

755 \def\bbl@beforestart{%
756 \bbl@usehooks{beforestart}{}}%
757 \global\let\bbl@beforestart\relax}
758 \AtBeginDocument{%
759 \@nameuse{bbl@beforestart}%
760 \if@filesw

```

```

761 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
762 \fi
763 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
764 \ifbbl@single % must go after the line above
765 \renewcommand\selectlanguage[1]{}%
766 \renewcommand\foreignlanguage[2]{#2}%
767 \global\let\babel@aux\@gobbletwo % Also as flag
768 \fi
769 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

770 \def\select@language@x#1{%
771 \ifcase\bbl@select@type
772 \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
773 \else
774 \select@language{#1}%
775 \fi}

```

## 9.4 Shorthands

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

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

```

776 \bbl@trace{Shorhands}
777 \def\bbl@add@special#1{% 1:a macro like "\", \?, etc.
778 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
779 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
780 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
781 \begingroup
782 \catcode`#1\active
783 \nfss@catcodes
784 \ifnum\catcode`#1=\active
785 \endgroup
786 \bbl@add\nfss@catcodes{\@makeother#1}%
787 \else
788 \endgroup
789 \fi
790 \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.

```

791 \def\bbl@remove@special#1{%
792 \begingroup
793 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
794 \else\noexpand##1\noexpand##2\fi}%
795 \def\do{\x\do}%
796 \def\@makeother{\x\@makeother}%
797 \edef\x{\endgroup
798 \def\noexpand\dospecials{\dospecials}%
799 \expandafter\ifx\csname @sanitize\endcsname\relax\else
800 \def\noexpand\@sanitize{\@sanitize}%
801 \fi}%
802 \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).

```
803 \def\bbl@active@def#1#2#3#4{%
804   \@namedef{#3#1}{%
805     \expandafter\ifx\csname#2@sh@#1@endcsname\relax
806       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
807     \else
808       \bbl@afterfi\csname#2@sh@#1@endcsname
809     \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.

```
810   \long\@namedef{#3@arg#1}##1{%
811     \expandafter\ifx\csname#2@sh@#1\string##1@endcsname\relax
812       \bbl@afterelse\csname#4#1@endcsname##1%
813     \else
814       \bbl@afterfi\csname#2@sh@#1\string##1@endcsname
815     \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.

```
816 \def\initiate@active@char#1{%
817   \bbl@ifunset{active@char\string#1}%
818   {\bbl@withactive
819     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
820   {}}

```

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`).

```
821 \def\@initiate@active@char#1#2#3{%
822   \bbl@csarg\edef\oricat@#2{\catcode`#2=\the\catcode`#2\relax}%
823   \ifx#1\@undefined
824     \bbl@csarg\edef\oridef@#2{\let\noexpand#1\noexpand\@undefined}%
825   \else
826     \bbl@csarg\let\oridef@#2#1%
827     \bbl@csarg\edef\oridef@#2{%

```

```

828 \let\noexpand#1%
829 \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
830 \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`).

```

831 \ifx#1#3\relax
832 \expandafter\let\csname normal@char#2\endcsname#3%
833 \else
834 \bbl@info{Making #2 an active character}%
835 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
836 \@namedef{normal@char#2}{%
837 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
838 \else
839 \@namedef{normal@char#2}{#3}%
840 \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).

```

841 \bbl@restoreactive{#2}%
842 \AtBeginDocument{%
843 \catcode`#2\active
844 \if@filesw
845 \immediate\write\@mainaux{\catcode`\string#2\active}%
846 \fi}%
847 \expandafter\bbl@add@special\csname#2\endcsname
848 \catcode`#2\active
849 \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⟩`).

```

850 \let\bbl@tempa\@firstoftwo
851 \if\string^#2%
852 \def\bbl@tempa{\noexpand\textormath}%
853 \else
854 \ifx\bbl@mathnormal\@undefined\else
855 \let\bbl@tempa\bbl@mathnormal
856 \fi
857 \fi
858 \expandafter\edef\csname active@char#2\endcsname{%
859 \bbl@tempa
860 {\noexpand\if@safe@actives
861 \noexpand\expandafter
862 \expandafter\noexpand\csname normal@char#2\endcsname
863 \noexpand\else
864 \noexpand\expandafter

```

```

865 \expandafter\noexpand\csname bbl@doactive#2\endcsname
866 \noexpand\fi}%
867 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
868 \bbl@csarg\edef{doactive#2}{%
869 \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!).

```

870 \bbl@csarg\edef{active@#2}{%
871 \noexpand\active@prefix\noexpand#1%
872 \expandafter\noexpand\csname active@char#2\endcsname}%
873 \bbl@csarg\edef{normal@#2}{%
874 \noexpand\active@prefix\noexpand#1%
875 \expandafter\noexpand\csname normal@char#2\endcsname}%
876 \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.

```

877 \bbl@active@def#2\user@group{user@active}{language@active}%
878 \bbl@active@def#2\language@group{language@active}{system@active}%
879 \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.

```

880 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
881 {\expandafter\noexpand\csname normal@char#2\endcsname}%
882 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
883 {\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.

```

884 \if\string'#2%
885 \let\prim@s\bbl@prim@s
886 \let\active@math@prime#1%
887 \fi
888 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

889 <<{*More package options}>> ≡
890 \DeclareOption{math=active}{}
891 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
892 <</More package options>>

```

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package *and* and the end of the *ldf*.

```

893 \ifpackagewith{babel}{KeepShorthandsActive}%

```

```

894 {\let\bbl@restoreactive\@gobble}%
895 {\def\bbl@restoreactive#1{%
896   \bbl@exp{%
897     \\\AfterBabelLanguage\\CurrentOption
898     {\catcode`#1=\the\catcode`#1\relax}%
899     \\\AtEndOfPackage
900     {\catcode`#1=\the\catcode`#1\relax}}}%
901   \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.

```

902 \def\bbl@sh@select#1#2{%
903   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
904     \bbl@afterelse\bbl@scndcs
905   \else
906     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
907   \fi}

```

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

```

908 \begingroup
909 \bbl@ifunset{ifincsname}%
910 {\gdef\active@prefix#1{%
911   \ifx\protect\@typeset@protect
912   \else
913     \ifx\protect\@unexpandable@protect
914       \noexpand#1%
915     \else
916       \protect#1%
917     \fi
918     \expandafter\@gobble
919   \fi}}
920 {\gdef\active@prefix#1{%
921   \ifincsname
922     \string#1%
923     \expandafter\@gobble
924   \else
925     \ifx\protect\@typeset@protect
926     \else
927       \ifx\protect\@unexpandable@protect
928         \noexpand#1%
929       \else
930         \protect#1%
931       \fi
932       \expandafter\expandafter\expandafter\@gobble
933     \fi
934   \fi}}
935 \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⟩`.

```

936 \newif\if@safe@actives
937 \@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.

```

938 \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`.

```

939 \def\bbl@activate#1{%
940   \bbl@withactive{\expandafter\let\expandafter}#1%
941   \csname bbl@active@\string#1\endcsname}
942 \def\bbl@deactivate#1{%
943   \bbl@withactive{\expandafter\let\expandafter}#1%
944   \csname bbl@normal@\string#1\endcsname}

```

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

```

945 \def\bbl@firstcs#1#2{\csname#1\endcsname}
946 \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.

```

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

```



```

970      \fi}%
971      \@namedef{#1@sh@string#2@string#3@}{#4}%
972      \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.

973 \def\textormath{%
974   \ifmmode
975     \expandafter\@secondoftwo
976   \else
977     \expandafter\@firstoftwo
978   \fi}

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

979 \def\user@group{user}
980 \def\language@group{english}
981 \def\system@group{system}

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

982 \def\useshorthands{%
983   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
984   \def\bbl@usesh@s#1{%
985     \bbl@usesh@x
986     {\AddBabelHook{babel-sh-}\string#1}{afterextras}{\bbl@activate{#1}}}%
987     {#1}}
988 \def\bbl@usesh@x#1#2{%
989   \bbl@ifshorthand{#2}%
990     {\def\user@group{user}%
991       \initiate@active@char{#2}%
992       #1%
993       \bbl@activate{#2}}%
994     {\bbl@error
995       {Cannot declare a shorthand turned off (\string#2)}
996       {Sorry, but you cannot use shorthands which have been\\%
997         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.

998 \def\user@language@group{user@\language@group}
999 \def\bbl@set@user@generic#1#2{%
1000   \bbl@ifunset{user@generic@active#1}%
1001     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1002       \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1003       \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1004         \expandafter\noexpand\csname normal@char#1\endcsname}%
1005       \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1006         \expandafter\noexpand\csname user@active#1\endcsname}}%

```

```

1007 \@empty}
1008 \newcommand\defineshorthand[3][user]{%
1009 \edef\bbl@tempa{\zap@space#1 \@empty}%
1010 \bbl@for\bbl@tempb\bbl@tempa{%
1011 \if*\expandafter\@car\bbl@tempb\@nil
1012 \edef\bbl@tempb{user\expandafter\@gobble\bbl@tempb}%
1013 \@expandtwoargs
1014 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1015 \fi
1016 \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.

```

1017 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

1018 \def\aliasshorthand#1#2{%
1019 \bbl@ifshorthand{#2}%
1020 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1021 \ifx\document\@notprerr
1022 \@notshorthand{#2}%
1023 \else
1024 \initiate@active@char{#2}%

```

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

```

1025 \expandafter\let\csname active@char\string#2\expandafter\endcsname
1026 \csname active@char\string#1\endcsname
1027 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1028 \csname normal@char\string#1\endcsname
1029 \bbl@activate{#2}%
1030 \fi
1031 \fi}%
1032 {\bbl@error
1033 {Cannot declare a shorthand turned off (\string#2)}
1034 {Sorry, but you cannot use shorthands which have been\\%
1035 turned off in the package options}}}

```

`\@notshorthand`

```

1036 \def\@notshorthand#1{%
1037 \bbl@error{%
1038 The character '\string #1' should be made a shorthand character;\\%
1039 add the command \string\usesshorthands\string{#1\string} to
1040 the preamble.\\%
1041 I will ignore your instruction}%
1042 {You may proceed, but expect unexpected results}}

```

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

```

1043 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1044 \DeclareRobustCommand*\shorthandoff{%
1045 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1046 \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.

```

1047 \def\bbl@switch@sh#1#2{%
1048   \ifx#2\@nnil\else
1049     \bbl@ifunset{\bbl@active@\string#2}%
1050     {\bbl@error
1051       {I cannot switch '\string#2' on or off--not a shorthand}%
1052       {This character is not a shorthand. Maybe you made\\%
1053         a typing mistake? I will ignore your instruction}}}%
1054     {\ifcase#1%
1055       \catcode'#212\relax
1056       \or
1057       \catcode'#2\active
1058       \or
1059       \csname bbl@oricat@\string#2\endcsname
1060       \csname bbl@oridef@\string#2\endcsname
1061       \fi}%
1062     \bbl@afterfi\bbl@switch@sh#1%
1063   \fi}

```

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

```

1064 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1065 \def\bbl@putsh#1{%
1066   \bbl@ifunset{\bbl@active@\string#1}%
1067   {\bbl@putsh@i#1\@empty\@nnil}%
1068   {\csname bbl@active@\string#1\endcsname}}
1069 \def\bbl@putsh@i#1#2\@nnil{%
1070   \csname\language @sh@\string#1@%
1071     \ifx\@empty#2\else\string#2@\fi\endcsname}
1072 \ifx\bbl@opt@shorthands\@nnil\else
1073   \let\bbl@s@initiate@active@char\initiate@active@char
1074   \def\initiate@active@char#1{%
1075     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1076   \let\bbl@s@switch@sh\bbl@switch@sh
1077   \def\bbl@switch@sh#1#2{%
1078     \ifx#2\@nnil\else
1079       \bbl@afterfi
1080       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1081       \fi}
1082   \let\bbl@s@activate\bbl@activate
1083   \def\bbl@activate#1{%
1084     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1085   \let\bbl@s@deactivate\bbl@deactivate
1086   \def\bbl@deactivate#1{%
1087     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1088   \fi

```

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

```

1089 \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  
`\bbl@pr@m@s` mathmode is `\prim@s`. This checks if the next character is a right quote. When the right  
quote is active, the definition of this macro needs to be adapted to look also for an active  
right quote; the hat could be active, too.

```

1090 \def\bbl@prim@s{%
1091   \prime\futurelet\@let@token\bbl@pr@m@s}
1092 \def\bbl@if@primes#1#2{%
1093   \ifx#1\@let@token
1094     \expandafter\@firstoftwo
1095   \else\ifx#2\@let@token
1096     \bbl@afterelse\expandafter\@firstoftwo
1097   \else
1098     \bbl@afterfi\expandafter\@secondoftwo
1099   \fi\fi}
1100 \begingroup
1101   \catcode`\^=7 \catcode`\*=\active \lccode`\*='^
1102   \catcode`\'=12 \catcode`\"=\active \lccode`\"='\'
1103   \lowercase{%
1104     \gdef\bbl@pr@m@s{%
1105       \bbl@if@primes""%
1106       \pr@@@s
1107       {\bbl@if@primes*^ \pr@@@t\egroup}}
1108 \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).

```

1109 \initiate@active@char{~}
1110 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1111 \bbl@activate{~}

```

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

```

1112 \expandafter\def\csname OT1dqpos\endcsname{127}
1113 \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

```

1114 \ifx\f@encoding\@undefined
1115   \def\f@encoding{OT1}
1116 \fi

```

## 9.5 Language attributes

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

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

```

1117 \bbl@trace{Language attributes}
1118 \newcommand\languageattribute[2]{%

```

```

1119 \def\bbl@tempc{#1}%
1120 \bbl@fixname\bbl@tempc
1121 \bbl@iflanguage\bbl@tempc{%
1122   \bbl@vforeach{#2}{%

```

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

```

1123   \ifx\bbl@known@attribs\@undefined
1124     \in@false
1125   \else

```

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

```

1126     \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1127   \fi

```

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

```

1128   \ifin@
1129     \bbl@warning{%
1130       You have more than once selected the attribute '##1'\%
1131       for language #1. Reported}%
1132   \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.

```

1133     \bbl@exp{%
1134       \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1135     \edef\bbl@tempa{\bbl@tempc-##1}%
1136     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1137     {\csname\bbl@tempc @attr@##1\endcsname}%
1138     {\@attrerrr{\bbl@tempc}{##1}}%
1139   \fi}}

```

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

```

1140 \@onlypreamble\languageattribute

```

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

```

1141 \newcommand*{\@attrerrr}[2]{%
1142   \bbl@error
1143   {The attribute #2 is unknown for language #1.}%
1144   {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}`.

```

1145 \def\bbl@declare@ttribute#1#2#3{%
1146   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1147   \ifin@
1148     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1149   \fi
1150   \bbl@add@list\bbl@attributes{#1-#2}%
1151   \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret T<sub>E</sub>X 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.

```
1152 \def\bbl@ifattributeset#1#2#3#4{%
```

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

```
1153 \ifx\bbl@known@attribs\@undefined
```

```
1154 \in@false
```

```
1155 \else
```

The we need to check the list of known attributes.

```
1156 \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
```

```
1157 \fi
```

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

```
1158 \ifin@
```

```
1159 \bbl@afterelse#3%
```

```
1160 \else
```

```
1161 \bbl@afterfi#4%
```

```
1162 \fi
```

```
1163 }
```

`\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 T<sub>E</sub>X-code to be executed when the attribute is known and the T<sub>E</sub>X-code to be executed otherwise.

```
1164 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1165 \let\bbl@tempa\@secondoftwo
```

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

```
1166 \bbl@loopx\bbl@tempb{#2}{%
```

```
1167 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
```

```
1168 \ifin@
```

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

```
1169 \let\bbl@tempa\@firstoftwo
```

```
1170 \else
```

```
1171 \fi}%
```

Finally we execute `\bbl@tempa`.

```
1172 \bbl@tempa
```

```
1173 }
```

`\bbl@clear@ttribs` This macro removes all the attribute code from L<sup>A</sup>T<sub>E</sub>X's memory at `\begin{document}` time (if any is present).

```
1174 \def\bbl@clear@ttribs{%
```

```
1175 \ifx\bbl@attributes\@undefined\else
```

```
1176 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
```

```
1177 \expandafter\bbl@clear@ttrib\bbl@tempa.
```

```
1178 }%
```

```
1179 \let\bbl@attributes\@undefined
```

```
1180 \fi}
```

```
1181 \def\bbl@clear@ttrib#1-#2.{%
```

```
1182 \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
```

```
1183 \AtBeginDocument{\bbl@clear@ttribs}
```

## 9.6 Support for saving macro definitions

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

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

```
\babel@beginsave 1184 \bbl@trace{Macros for saving definitions}
                  1185 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1186 \newcount\babel@savecnt
1187 \babel@beginsave
```

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX`<sup>32</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.

```
1188 \def\babel@save#1{%
1189   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1190   \toks@\expandafter{\originalTeX\let#1=}%
1191   \bbl@exp{%
1192     \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1193   \advance\babel@savecnt\@ne}
```

`\babel@savevariable` The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive.

```
1194 \def\babel@savevariable#1{%
1195   \toks@\expandafter{\originalTeX #1=}%
1196   \bbl@exp{\def\originalTeX{\the\toks@the#1\relax}}}
```

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

```
1197 \def\bbl@frenchspacing{%
1198   \ifnum\the\sfcode`\.=\@m
1199     \let\bbl@nonfrenchspacing\relax
1200   \else
1201     \frenchspacing
1202     \let\bbl@nonfrenchspacing\nonfrenchspacing
1203   \fi}
1204 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 9.7 Short tags

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

```
1205 \bbl@trace{Short tags}
1206 \def\babeltags#1{%
```

---

<sup>32</sup>`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

```

1207 \edef\bbl@tempa{\zap@space#1 \@empty}%
1208 \def\bbl@tempb##1=##2\@@{%
1209   \edef\bbl@tempc{%
1210     \noexpand\newcommand
1211     \expandafter\noexpand\csname ##1\endcsname{%
1212       \noexpand\protect
1213       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1214     \noexpand\newcommand
1215     \expandafter\noexpand\csname text##1\endcsname{%
1216       \noexpand\foreignlanguage{##2}}
1217   \bbl@tempc}%
1218 \bbl@for\bbl@tempa\bbl@tempa{%
1219   \expandafter\bbl@tempb\bbl@tempa\@@}}

```

## 9.8 Hyphens

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

```

1220 \bbl@trace{Hyphens}
1221 \@onlypreamble\babelhyphenation
1222 \AtEndOfPackage{%
1223   \newcommand\babelhyphenation[2][\@empty]{%
1224     \ifx\bbl@hyphenation@relax
1225       \let\bbl@hyphenation@\@empty
1226     \fi
1227     \ifx\bbl@hyphlist\@empty\else
1228       \bbl@warning{%
1229         You must not intermingle \string\selectlanguage\space and\\%
1230         \string\babelhyphenation\space or some exceptions will not\\%
1231         be taken into account. Reported}%
1232     \fi
1233     \ifx\@empty#1%
1234       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1235     \else
1236       \bbl@vforeach{#1}{%
1237         \def\bbl@tempa{##1}%
1238         \bbl@fixname\bbl@tempa
1239         \bbl@iflanguage\bbl@tempa{%
1240           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1241             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1242             \@empty
1243             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1244             #2}}}%
1245     \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip Opt plus Opt`<sup>33</sup>.

```

1246 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
1247 \def\bbl@t@one{T1}
1248 \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.

<sup>33</sup> $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.



```

1249 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1250 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
1251 \def\bb1@hyphen{%
1252   \@ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i \@empty}}
1253 \def\bb1@hyphen@i#1#2{%
1254   \bb1@ifunset{\bb1@hy@#1#2\@empty}%
1255   {\csname bb1@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1256   {\csname bb1@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.

```

1257 \def\bb1@usehyphen#1{%
1258   \leavevmode
1259   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1260   \nobreak\hskip\z@skip}
1261 \def\bb1@@usehyphen#1{%
1262   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1263 \def\bb1@hyphenchar{%
1264   \ifnum\hyphenchar\font=\m@ne
1265     \babelnullhyphen
1266   \else
1267     \char\hyphenchar\font
1268   \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 \bb1@hy@nobreak is redundant.

```

1269 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1270 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1271 \def\bb1@hy@hard{\bb1@usehyphen\bb1@hyphenchar}
1272 \def\bb1@hy@@hard{\bb1@usehyphen\bb1@hyphenchar}
1273 \def\bb1@hy@nobreak{\bb1@usehyphen{\mbox{\bb1@hyphenchar}}}
1274 \def\bb1@hy@nobreak{\mbox{\bb1@hyphenchar}}
1275 \def\bb1@hy@repeat{%
1276   \bb1@usehyphen{%
1277     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
1278 \def\bb1@hy@@repeat{%
1279   \bb1@usehyphen{%
1280     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
1281 \def\bb1@hy@empty{\hskip\z@skip}
1282 \def\bb1@hy@@empty{\discretionary{}{}{}}

```

\bb1@disc For some languages the macro \bb1@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.

```

1283 \def\bb1@disc#1#2{\nobreak\discretionary{#2-}{#1}\bb1@allowhyphens}

```

## 9.9 Multiencoding strings

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

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

```

1284 \bbl@trace{Multiencoding strings}
1285 \def\bbl@tglobal#1{\global\let#1#1}
1286 \def\bbl@recatcode#1{%
1287   \@tempcnta="7F
1288   \def\bbl@tempa{%
1289     \ifnum\@tempcnta>"FF\else
1290       \catcode\@tempcnta=#1\relax
1291       \advance\@tempcnta\@ne
1292       \expandafter\bbl@tempa
1293     \fi}%
1294   \bbl@tempa}

```

The second one. We need to patch `\@uclclist`, but it is done once and only if `\SetCase` is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact `\@uclclist` is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually `\reserved@a`), we pass it as argument to `\bbl@uclc`. The parser is restarted inside `\<lang>\bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).

```

1295 \@ifpackagewith{babel}{nocase}%
1296   {\let\bbl@patchuclc\relax}%
1297   {\def\bbl@patchuclc{%
1298     \global\let\bbl@patchuclc\relax
1299     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1300     \gdef\bbl@uclc##1{%
1301       \let\bbl@encoded\bbl@encoded@uclc
1302       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1303       {##1}%
1304       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1305        \csname\language @bbl@uclc\endcsname}%
1306       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1307     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1308     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1309 <<(*More package options)>> ≡
1310 \DeclareOption{nocase}{}
1311 <</More package options>>

```

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

```

1312 <<(*More package options)>> ≡
1313 \let\bbl@opt@strings\@nnil % accept strings=value
1314 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1315 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1316 \def\BabelStringsDefault{generic}
1317 <</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.

```
1318 \@onlypreamble\StartBabelCommands
```

```

1319 \def\StartBabelCommands{%
1320   \begingroup
1321   \bbl@recatcode{11}%
1322   <<Macros local to BabelCommands>>
1323   \def\bbl@provstring##1##2{%
1324     \providecommand##1{##2}%
1325     \bbl@toglobal##1}%
1326   \global\let\bbl@scafter\@empty
1327   \let\StartBabelCommands\bbl@startcmds
1328   \ifx\BabelLanguages\relax
1329     \let\BabelLanguages\CurrentOption
1330   \fi
1331   \begingroup
1332   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1333   \StartBabelCommands}
1334 \def\bbl@startcmds{%
1335   \ifx\bbl@screset\@nnil\else
1336     \bbl@usehooks{stopcommands}{}%
1337   \fi
1338   \endgroup
1339   \begingroup
1340   \@ifstar
1341     {\ifx\bbl@opt@strings\@nnil
1342       \let\bbl@opt@strings\BabelStringsDefault
1343     \fi
1344     \bbl@startcmds@i}%
1345   \bbl@startcmds@i}
1346 \def\bbl@startcmds@i#1#2{%
1347   \edef\bbl@L{\zap@space#1 \@empty}%
1348   \edef\bbl@G{\zap@space#2 \@empty}%
1349   \bbl@startcmds@ii}

```

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

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

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

```

1350 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1351   \let\SetString\@gobbletwo
1352   \let\bbl@stringdef\@gobbletwo
1353   \let\AfterBabelCommands\@gobble
1354   \ifx\@empty#1%
1355     \def\bbl@sc@label{generic}%
1356     \def\bbl@encstring##1##2{%
1357       \ProvideTextCommandDefault##1{##2}%
1358       \bbl@toglobal##1%
1359       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1360     \let\bbl@sctest\in@true
1361   \else
1362     \let\bbl@sc@charset\space % <- zapped below
1363     \let\bbl@sc@fontenc\space % <- " "
1364     \def\bbl@tempa##1=##2\@nil{%
1365       \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%

```

```

1366 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1367 \def\bbl@tempa##1 ##2{% space -> comma
1368   ##1%
1369   \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1370 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1371 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1372 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1373 \def\bbl@encstring##1##2{%
1374   \bbl@foreach\bbl@sc@fontenc{%
1375     \bbl@ifunset{T####1}%
1376     {}%
1377     {\ProvideTextCommand##1{####1}{##2}%
1378     \bbl@tglobal##1%
1379     \expandafter
1380     \bbl@tglobal\csname####1\string##1\endcsname}}}%
1381 \def\bbl@sctest{%
1382   \bbl@xin@{\,\bbl@opt@strings,}{,\,\bbl@sc@label,\bbl@sc@fontenc,}}%
1383 \fi
1384 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1385 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1386   \let\AfterBabelCommands\bbl@aftercmds
1387   \let\SetString\bbl@setstring
1388   \let\bbl@stringdef\bbl@encstring
1389 \else % ie, strings=value
1390 \bbl@sctest
1391 \ifin@
1392   \let\AfterBabelCommands\bbl@aftercmds
1393   \let\SetString\bbl@setstring
1394   \let\bbl@stringdef\bbl@provstring
1395 \fi\fi\fi
1396 \bbl@scswitch
1397 \ifx\bbl@G\@empty
1398   \def\SetString##1##2{%
1399     \bbl@error{Missing group for string \string##1}%
1400     {You must assign strings to some category, typically\\%
1401     captions or extras, but you set none}}%
1402 \fi
1403 \ifx\@empty#1%
1404   \bbl@usehooks{defaultcommands}{}%
1405 \else
1406   \@expandtwoargs
1407   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1408 \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).

```

1409 \def\bbl@forlang#1##2{%
1410   \bbl@for#1\bbl@L{%
1411     \bbl@xin@{, #1,}{,\,\BabelLanguages,}%
1412     \ifin@#2\relax\fi}}
1413 \def\bbl@scswitch{%
1414   \bbl@forlang\bbl@tempa{%

```

```

1415 \ifx\bb1@G\@empty\else
1416 \ifx\SetString\@gobbletwo\else
1417 \edef\bb1@GL{\bb1@G\bb1@tempa}%
1418 \bb1@xin@{\bb1@GL,}{,\bb1@screset,}%
1419 \ifin@else
1420 \global\expandafter\let\csname\bb1@GL\endcsname\@undefined
1421 \xdef\bb1@screset{\bb1@screset,\bb1@GL}%
1422 \fi
1423 \fi
1424 \fi}}
1425 \AtEndOfPackage{%
1426 \def\bb1@forlang#1#2{\bb1@for#1\bb1@L{\bb1@ifunset{date#1}{\#2}}}%
1427 \let\bb1@scswitch\relax}
1428 \@onlypreamble\EndBabelCommands
1429 \def\EndBabelCommands{%
1430 \bb1@usehooks{stopcommands}{}%
1431 \endgroup
1432 \endgroup
1433 \bb1@scafter}

```

Now we define commands to be used inside \StartBabelCommands.

**Strings** The following macro is the actual definition of \SetString when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like \providescommand). With the event stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

1434 \def\bb1@setstring#1#2{%
1435 \bb1@forlang\bb1@tempa{%
1436 \edef\bb1@LC{\bb1@tempa\bb1@stripslash#1}%
1437 \bb1@ifunset{\bb1@LC}% eg, \germanchaptername
1438 {\global\expandafter % TODO - con \bb1@exp ?
1439 \bb1@add\csname\bb1@G\bb1@tempa\expandafter\endcsname\expandafter
1440 {\expandafter\bb1@scset\expandafter#1\csname\bb1@LC\endcsname}}}%
1441 }%
1442 \def\BabelString{\#2}%
1443 \bb1@usehooks{stringprocess}{}%
1444 \expandafter\bb1@stringdef
1445 \csname\bb1@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include \bb1@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.

```

1446 \ifx\bb1@opt@strings\relax
1447 \def\bb1@scset#1#2{\def#1{\bb1@encoded#2}}
1448 \bb1@patchuclc
1449 \let\bb1@encoded\relax
1450 \def\bb1@encoded@uclc#1{%
1451 \@inmathwarn#1%
1452 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1453 \expandafter\ifx\csname ?\string#1\endcsname\relax
1454 \TextSymbolUnavailable#1%
1455 \else
1456 \csname ?\string#1\endcsname
1457 \fi
1458 \else
1459 \csname\cf@encoding\string#1\endcsname

```

```

1460 \fi}
1461 \else
1462 \def\bbl@scset#1#2{\def#1{#2}}
1463 \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.

```

1464 <<(*Macros local to BabelCommands)>> ≡
1465 \def\SetStringLoop##1##2{%
1466   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1467   \count@\z@
1468   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1469     \advance\count@\@ne
1470     \toks@\expandafter{\bbl@tempa}%
1471     \bbl@exp{%
1472       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1473       \count@=\the\count@\relax}}}%
1474 <</Macros local to BabelCommands>>

```

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

```

1475 \def\bbl@aftercmds#1{%
1476   \toks@\expandafter{\bbl@scafter#1}%
1477   \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.

```

1478 <<(*Macros local to BabelCommands)>> ≡
1479 \newcommand\SetCase[3][]{%
1480   \bbl@patchuclc
1481   \bbl@forlang\bbl@tempa{%
1482     \expandafter\bbl@encstring
1483     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1484     \expandafter\bbl@encstring
1485     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1486     \expandafter\bbl@encstring
1487     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1488 <</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.

```

1489 <<(*Macros local to BabelCommands)>> ≡
1490 \newcommand\SetHyphenMap[1]{%
1491   \bbl@forlang\bbl@tempa{%
1492     \expandafter\bbl@stringdef
1493     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1494 <</Macros local to BabelCommands>>

```

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

```

1495 \newcommand\BabelLower[2]{% one to one.
1496   \ifnum\lccode#1=#2\else
1497     \babel@savevariable{\lccode#1}%
1498     \lccode#1=#2\relax
1499   \fi}

```

```

1500 \newcommand\BabelLowerMM[4]{% many-to-many
1501   \@tempcnta=#1\relax
1502   \@tempcntb=#4\relax
1503   \def\bbl@tempa{%
1504     \ifnum\@tempcnta>#2\else
1505       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1506       \advance\@tempcnta#3\relax
1507       \advance\@tempcntb#3\relax
1508       \expandafter\bbl@tempa
1509     \fi}%
1510   \bbl@tempa}
1511 \newcommand\BabelLowerMO[4]{% many-to-one
1512   \@tempcnta=#1\relax
1513   \def\bbl@tempa{%
1514     \ifnum\@tempcnta>#2\else
1515       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1516       \advance\@tempcnta#3
1517       \expandafter\bbl@tempa
1518     \fi}%
1519   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1520 <<{*More package options}> ≡
1521 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1522 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1523 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1524 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1525 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1526 <</More package options>>

```

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

```

1527 \AtEndOfPackage{%
1528   \ifx\bbl@opt@hyphenmap\undefined
1529     \bbl@xin@{,}{\bbl@language@opts}%
1530     \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
1531   \fi}

```

## 9.10 Macros common to a number of languages

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

```

1532 \bbl@trace{Macros related to glyphs}
1533 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1534   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1535   \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.

```

1536 \def\save@sf@q#1{\leavevmode
1537   \begingroup
1538     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1539   \endgroup}

```

## 9.11 Making glyphs available

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

### 9.11.1 Quotation marks

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

```
1540 \ProvideTextCommand{\quotedblbase}{OT1}{%
1541   \save@sf@q{\set@low@box{\textquotedblright\}%
1542     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
1543 \ProvideTextCommandDefault{\quotedblbase}{%
1544   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1545 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1546   \save@sf@q{\set@low@box{\textquoteright\}%
1547     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
1548 \ProvideTextCommandDefault{\quotesinglbase}{%
1549   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
\guillemotright 1550 \ProvideTextCommand{\guillemotleft}{OT1}{%
1551   \ifmmode
1552     \ll
1553   \else
1554     \save@sf@q{\nobreak
1555       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
1556   \fi}
1557 \ProvideTextCommand{\guillemotright}{OT1}{%
1558   \ifmmode
1559     \gg
1560   \else
1561     \save@sf@q{\nobreak
1562       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
1563   \fi}
```

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

```
1564 \ProvideTextCommandDefault{\guillemotleft}{%
1565   \UseTextSymbol{OT1}{\guillemotleft}}
1566 \ProvideTextCommandDefault{\guillemotright}{%
1567   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1568 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1569   \ifmmode
1570     <%
1571   \else
1572     \save@sf@q{\nobreak
1573       \raise.2ex\hbox{$\scriptscriptstyle<$}\bb1@allowhyphens}%
1574   \fi}
1575 \ProvideTextCommand{\guilsinglright}{OT1}{%
1576   \ifmmode
```



```

1577 >%
1578 \else
1579 \save@sf@q{\nobreak
1580 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1581 \fi}

```

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

```

1582 \ProvideTextCommandDefault{\guilsinglleft}{%
1583 \UseTextSymbol{OT1}{\guilsinglleft}}
1584 \ProvideTextCommandDefault{\guilsinglright}{%
1585 \UseTextSymbol{OT1}{\guilsinglright}}

```

### 9.11.2 Letters

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

```

1586 \DeclareTextCommand{\ij}{OT1}{%
1587 i\kern-0.02em\bbl@allowhyphens j}
1588 \DeclareTextCommand{\IJ}{OT1}{%
1589 I\kern-0.02em\bbl@allowhyphens J}
1590 \DeclareTextCommand{\ij}{T1}{\char188}
1591 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

1592 \ProvideTextCommandDefault{\ij}{%
1593 \UseTextSymbol{OT1}{\ij}}
1594 \ProvideTextCommandDefault{\IJ}{%
1595 \UseTextSymbol{OT1}{\IJ}}

```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, `\DJ` but not in the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```

1596 \def\crrtic@{\hrule height0.1ex width0.3em}
1597 \def\crrtic@{\hrule height0.1ex width0.33em}
1598 \def\ddj@{%
1599 \setbox0\hbox{d}\dimen@=\ht0
1600 \advance\dimen@1ex
1601 \dimen@.45\dimen@
1602 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1603 \advance\dimen@ii.5ex
1604 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1605 \def\DDJ@{%
1606 \setbox0\hbox{D}\dimen@=.55\ht0
1607 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1608 \advance\dimen@ii.15ex % correction for the dash position
1609 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1610 \dimen\thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1611 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1612 %
1613 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1614 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

1615 \ProvideTextCommandDefault{\dj}{%
1616   \UseTextSymbol{OT1}{\dj}}
1617 \ProvideTextCommandDefault{\DJ}{%
1618   \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.

```

1619 \DeclareTextCommand{\SS}{OT1}{SS}
1620 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

### 9.11.3 Shorthands for quotation marks

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

\glq The ‘german’ single quotes.

```

\grq 1621 \ProvideTextCommandDefault{\glq}{%
1622   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

1623 \ProvideTextCommand{\grq}{T1}{%
1624   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
1625 \ProvideTextCommand{\grq}{TU}{%
1626   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1627 \ProvideTextCommand{\grq}{OT1}{%
1628   \save@sf@q{\kern-.0125em
1629     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1630     \kern.07em\relax}}
1631 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

\glqq The ‘german’ double quotes.

```

\grqq 1632 \ProvideTextCommandDefault{\glqq}{%
1633   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1634 \ProvideTextCommand{\grqq}{T1}{%
1635   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1636 \ProvideTextCommand{\grqq}{TU}{%
1637   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1638 \ProvideTextCommand{\grqq}{OT1}{%
1639   \save@sf@q{\kern-.07em
1640     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1641     \kern.07em\relax}}
1642 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}

```

\flq The ‘french’ single guillemets.

```

\frq 1643 \ProvideTextCommandDefault{\flq}{%
1644   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1645 \ProvideTextCommandDefault{\frq}{%
1646   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```
\frqq 1647 \ProvideTextCommandDefault{\flqq}{%
1648   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1649 \ProvideTextCommandDefault{\frqq}{%
1650   \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

#### 9.11.4 Umlauts and tremas

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

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

```
1651 \def\umlauthigh{%
1652   \def\bbl@umlauta##1{\leavevmode\bgroup%
1653     \expandafter\accent\csname\fontencoding dqpos\endcsname
1654     ##1\bbl@allowhyphens\egroup}%
1655   \let\bbl@umlaute\bbl@umlauta}
1656 \def\umlautlow{%
1657   \def\bbl@umlauta{\protect\lower@umlaut}}
1658 \def\umlautelower{%
1659   \def\bbl@umlaute{\protect\lower@umlaut}}
1660 \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.

```
1661 \expandafter\ifx\csname U@D\endcsname\relax
1662   \csname newdimen\endcsname\U@D
1663 \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.

```
1664 \def\lower@umlaut#1{%
1665   \leavevmode\bgroup
1666   \U@D 1ex%
1667   {\setbox\z@\hbox{%
1668     \expandafter\char\csname\fontencoding dqpos\endcsname}%
1669     \dimen@ -.45ex\advance\dimen@\ht\z@
1670     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1671   \expandafter\accent\csname\fontencoding dqpos\endcsname
1672   \fontdimen5\font\U@D #1%
1673   \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).

```

1674 \AtBeginDocument{%
1675   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
1676   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
1677   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
1678   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
1679   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
1680   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
1681   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
1682   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
1683   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
1684   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
1685   \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}%
1686 }

```

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

```

1687 \ifx\l@english\@undefined
1688   \chardef\l@english\z@
1689 \fi
1690 \main@language{english}

```

## 9.12 Layout

### Work in progress.

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

```

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

```

```

1719 \BabelPatchSection{subsubsection}%
1720 \BabelPatchSection{paragraph}%
1721 \BabelPatchSection{subparagraph}%
1722 \def\babel@toc#1{%
1723   \select@language@x{\bbl@main@language}}{}
1724 \IfBabelLayout{captions}%
1725   {\BabelPatchSection{caption}}{}

```

### 9.13 Load engine specific macros

```

1726 \bbl@trace{Input engine specific macros}
1727 \ifcase\bbl@engine
1728   \input txtbabel.def
1729 \or
1730   \input luababel.def
1731 \or
1732   \input xebabel.def
1733 \fi

```

### 9.14 Creating languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```

1734 \bbl@trace{Creating languages and reading ini files}
1735 \newcommand\babelprovide[2][{}]{%
1736   \let\bbl@savelangname\language@name
1737   \edef\bbl@savelocaleid{\the\localeid}%
1738   % Set name and locale id
1739   \def\language@name{#2}%
1740   \bbl@id@assign
1741   \let\bbl@KVP@captions\@nil
1742   \let\bbl@KVP@import\@nil
1743   \let\bbl@KVP@main\@nil
1744   \let\bbl@KVP@script\@nil
1745   \let\bbl@KVP@language\@nil
1746   \let\bbl@KVP@hyphenrules\@nil % only for provide@new
1747   \let\bbl@KVP@mapfont\@nil
1748   \let\bbl@KVP@maparabic\@nil
1749   \let\bbl@KVP@mapdigits\@nil
1750   \let\bbl@KVP@intraspace\@nil
1751   \let\bbl@KVP@intrapenalty\@nil
1752   \bbl@forkv{#1}{% TODO - error handling
1753     \in@{.}{##1}%
1754     \ifin@
1755       \bbl@renewinikey##1\@{##2}%
1756     \else
1757       \bbl@csarg\def{KVP@##1}{##2}%
1758     \fi}%
1759   % == import, captions ==
1760   \ifx\bbl@KVP@import\@nil\else
1761     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1762     {\begingroup
1763       \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1764       \InputIfFileExists{babel-#2.tex}{}{}%
1765     \endgroup}%
1766   }%
1767 \fi

```

```

1768 \ifx\bb1@KVP@captions\@nil
1769 \let\bb1@KVP@captions\bb1@KVP@import
1770 \fi
1771 % Load ini
1772 \bb1@ifunset{date#2}%
1773 {\bb1@provide@new{#2}}%
1774 {\bb1@ifblank{#1}%
1775 {\bb1@error
1776 {If you want to modify `#2' you must tell how in\\
1777 the optional argument. See the manual for the\\
1778 available options.}%
1779 {Use this macro as documented}}%
1780 {\bb1@provide@renew{#2}}}%
1781 % Post tasks
1782 \bb1@exp{\\babelensure[exclude=\\today]{#2}}%
1783 \bb1@ifunset{bb1@ensure@\\language}%
1784 {\bb1@exp{
1785 \\DeclareRobustCommand\<bb1@ensure@\\language>[1]{%
1786 \\foreignlanguage{\\language}%
1787 {###1}}}%
1788 }%
1789 % At this point all parameters are defined if 'import'. Now we
1790 % execute some code depending on them. But what about if nothing was
1791 % imported? We just load the very basic parameters: ids and a few
1792 % more.
1793 \bb1@ifunset{bb1@lname#2}%
1794 {\def\BabelBeforeIni##1##2{%
1795 \begingroup
1796 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
1797 \let\bb1@ini@captions@aux\@gobbletwo
1798 \def\bb1@inidate ###1.###2.###3.###4\relax ###5###6{%
1799 \bb1@read@ini{##1}{basic data}%
1800 \bb1@exportkey{chrng}{characters.ranges}{}%
1801 \bb1@exportkey{dgnat}{numbers.digits.native}{}%
1802 \bb1@exportkey{hyphr}{typography.hyphenrules}{}%
1803 \bb1@exportkey{intsp}{typography.intraspaces}{}%
1804 \endgroup}% boxed, to avoid extra spaces:
1805 {\setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
1806 }%
1807 % -
1808 % == script, language ==
1809 % Override the values from ini or defines them
1810 \ifx\bb1@KVP@script\@nil\else
1811 \bb1@csarg\edef{sname#2}{\bb1@KVP@script}%
1812 \fi
1813 \ifx\bb1@KVP@language\@nil\else
1814 \bb1@csarg\edef{lname#2}{\bb1@KVP@language}%
1815 \fi
1816 % == mapfont ==
1817 % For bidi texts, to switch the font based on direction
1818 \ifx\bb1@KVP@mapfont\@nil\else
1819 \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}{}%
1820 {\bb1@error{Option ``\bb1@KVP@mapfont' unknown for\\
1821 mapfont. Use `direction'.%
1822 {See the manual for details.}}}%
1823 \bb1@ifunset{bb1@lsys@\\language}{\bb1@provide@lsys{\\language}}}%
1824 \bb1@ifunset{bb1@wdir@\\language}{\bb1@provide@dirs{\\language}}}%
1825 \ifx\bb1@mapselect\@undefined
1826 \AtBeginDocument{%

```

```

1827     \expandafter\bbbl@add\csname selectfont \endcsname{\bbbl@mapselect}}%
1828     {\selectfont}}%
1829     \def\bbbl@mapselect{%
1830         \let\bbbl@mapselect\relax
1831         \edef\bbbl@prefontid{\fontid\font}}%
1832     \def\bbbl@mapdir##1{%
1833         {\def\language\language{##1}%
1834         \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
1835         \bbbl@switchfont
1836         \directlua{Babel.fontmap
1837             [\the\csname bbl@wdir@##1\endcsname]%
1838             [\bbbl@prefontid]=\fontid\font}}}%
1839     \fi
1840     \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
1841     \fi
1842     % == intraspace, intrapenalty ==
1843     % For CJK, East Asian, Southeast Asian, if interspace in ini
1844     \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
1845         \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
1846     \fi
1847     \bbbl@provide@intraspace
1848     % == maparabic ==
1849     % Native digits, if provided in ini (TeX level, xe and lua)
1850     \ifcase\bbbl@engine\else
1851         \bbbl@ifunset{\bbbl@dgnat@\language}\{%
1852             {\expandafter\ifx\csname bbl@dgnat@\language\endcsname\@empty\else
1853                 \expandafter\expandafter\expandafter
1854                 \bbbl@setdigits\csname bbl@dgnat@\language\endcsname
1855                 \ifx\bbbl@KVP@maparabic\@nil\else
1856                     \ifx\bbbl@latinarabic\@undefined
1857                         \expandafter\let\expandafter\@arabic
1858                         \csname bbl@counter@\language\endcsname
1859                     \else % ie, if layout=counters, which redefines \@arabic
1860                         \expandafter\let\expandafter\bbbl@latinarabic
1861                         \csname bbl@counter@\language\endcsname
1862                     \fi
1863                 \fi
1864             \fi}%
1865     \fi
1866     % == mapdigits ==
1867     % Native digits (lua level).
1868     \ifodd\bbbl@engine
1869         \ifx\bbbl@KVP@mapdigits\@nil\else
1870             \bbbl@ifunset{\bbbl@dgnat@\language}\{%
1871                 {\RequirePackage{luatexbase}%
1872                 \bbbl@activate@preotf
1873                 \directlua{
1874                     Babel = Babel or {} %% -> presets in luababel
1875                     Babel.digits_mapped = true
1876                     Babel.digits = Babel.digits or {}
1877                     Babel.digits[\the\localeid] =
1878                         table.pack(string.utfvalue('\bbbl@cs{dgnat@\language}'))
1879                     if not Babel.numbers then
1880                         function Babel.numbers(head)
1881                             local LOCALE = luatexbase.registernumber'bbl@attr@locale'
1882                             local GLYPH = node.id'glyph'
1883                             local inmath = false
1884                             for item in node.traverse(head) do
1885                                 if not inmath and item.id == GLYPH then

```

```

1886         local temp = node.get_attribute(item, LOCALE)
1887         if Babel.digits[temp] then
1888             local chr = item.char
1889             if chr > 47 and chr < 58 then
1890                 item.char = Babel.digits[temp][chr-47]
1891             end
1892         end
1893         elseif item.id == node.id'math' then
1894             inmath = (item.subtype == 0)
1895         end
1896     end
1897     return head
1898 end
1899 end
1900 }}
1901 \fi
1902 \fi
1903 % == require.babel in ini ==
1904 % To load or reload the babel-*.tex, if require.babel in ini
1905 \bbl@ifunset{bbl@rqtex@languagename}{}%
1906 {\expandafter\ifx\csname bbl@rqtex@languagename\endcsname\@empty\else
1907     \let\BabelBeforeIni\@gobbletwo
1908     \chardef\atcatcode=\catcode`\@
1909     \catcode`\@=11\relax
1910     \InputIfFileExists{babel-bbl@cs{rqtex@languagename}.tex}{}%
1911     \catcode`\@=\atcatcode
1912     \let\atcatcode\relax
1913 \fi}%
1914 % == main ==
1915 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
1916     \let\languagename\bbl@savelangname
1917     \chardef\localeid\bbl@savelocaleid\relax
1918 \fi}

```

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

```

1919 \def\bbl@setdigits#1#2#3#4#5{%
1920     \bbl@exp{%
1921         \def<\languagename digits>####1{% ie, \langdigits
1922             \<bbl@digits@languagename>####1\\\@nil}%
1923         \def<\languagename counter>####1{% ie, \langcounter
1924             \expandafter\<bbl@counter@languagename>%
1925             \\\csname c@####1\endcsname}%
1926         \def<bbl@counter@languagename>####1{% ie, \bbl@counter@lang
1927             \expandafter\<bbl@digits@languagename>%
1928             \\\number####1\\\@nil}}%
1929 \def\bbl@tempa##1##2##3##4##5{%
1930     \bbl@exp{% Wow, quite a lot of hashes! :- (
1931         \def<bbl@digits@languagename>#####1{%
1932             \\\ifx#####1\\\@nil % ie, \bbl@digits@lang
1933             \\\else
1934                 \\\ifx0#####1#1%
1935                 \\\else\\\ifx1#####1#2%
1936                 \\\else\\\ifx2#####1#3%
1937                 \\\else\\\ifx3#####1#4%
1938                 \\\else\\\ifx4#####1#5%
1939                 \\\else\\\ifx5#####1##1%
1940                 \\\else\\\ifx6#####1##2%
1941                 \\\else\\\ifx7#####1##3%

```





```

1998 \fi
1999 % == hyphenrules ==
2000 \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2001 \def\bbl@provide@hyphens#1{%
2002   \let\bbl@tempa\relax
2003   \ifx\bbl@KVP@hyphenrules\@nil\else
2004     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2005     \bbl@foreach\bbl@KVP@hyphenrules{%
2006       \ifx\bbl@tempa\relax % if not yet found
2007         \bbl@ifsamestring{##1}{+}%
2008         {\bbl@exp{\addlanguage\<1@##1>}}}%
2009         {}%
2010         \bbl@ifunset{1@##1}%
2011         {}%
2012         {\bbl@exp{\let\bbl@tempa\<1@##1>}}}%
2013     \fi}%
2014 \fi
2015 \ifx\bbl@tempa\relax % if no opt or no language in opt found
2016   \ifx\bbl@KVP@import\@nil\else % if importing
2017     \bbl@exp{%
2018       \bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
2019       {}%
2020       {\let\bbl@tempa\<1@\@nameuse{bbl@hyphr@language}>}}%
2021   \fi
2022 \fi
2023 \bbl@ifunset{bbl@tempa}% ie, relax or undefined
2024   {\bbl@ifunset{1@#1}% no hyphenrules found - fallback
2025     {\bbl@exp{\adddialect\<1@#1>\language}}%
2026     {}% so, 1@<lang> is ok - nothing to do
2027     {\bbl@exp{\adddialect\<1@#1>\bbl@tempa}}% found in opt list or ini
2028   \bbl@ifunset{bbl@prehc\language}%
2029   {}% TODO - XeTeX, based on \babelfont and HyphenChar?
2030   {\ifodd\bbl@engine\bbl@exp{%
2031     \bbl@ifblank{\@nameuse{bbl@prehc@#1}}%
2032     {}%
2033     {\AddBabelHook[\language]{babel-prehc-\language}{patterns}%
2034       {\prehyphenchar=\@nameuse{bbl@prehc\language}\relax}}}%
2035   \fi}}

```

The reader of ini files. There are 3 possible cases: a section name (in the form [ . . . ]), a comment (starting with ;) and a key/value pair. *TODO - Work in progress.*

```

2036 \def\bbl@read@ini#1#2{%
2037   \openin1=babel-#1.ini % FIXME - number must not be hardcoded
2038   \ifeof1
2039     \bbl@error
2040     {There is no ini file for the requested language\%
2041       (#1). Perhaps you misspelled it or your installation\%
2042       is not complete.}%
2043     {Fix the name or reinstall babel.}%
2044   \else
2045     \let\bbl@section\@empty
2046     \let\bbl@savestrings\@empty
2047     \let\bbl@savetoday\@empty
2048     \let\bbl@savestate\@empty
2049     \def\bbl@inipreread##1=##2\@{#%
2050       \bbl@trim\def\bbl@tempa{##1}% Redundant below !!
2051       % Move trims here ??

```

```

2052 \bbl@ifunset{\bbl@KVP@\bbl@section..\bbl@tempa}%
2053 {\expandafter\bbl@inireader\bbl@tempa=##2\@@}%
2054 {}}%
2055 \let\bbl@inireader\bbl@iniskip
2056 \bbl@info{Importing #2 for \language\name}%
2057 from babel-#1.ini. Reported}%
2058 \loop
2059 \if T\ifeof1F\fi T\relax % Trick, because inside \loop
2060 \endlinechar\m@ne
2061 \read1 to \bbl@line
2062 \endlinechar\^^M
2063 \ifx\bbl@line\empty\else
2064 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2065 \fi
2066 \repeat
2067 \bbl@foreach\bbl@renewlist{%
2068 \bbl@ifunset{\bbl@renew@##1}{\bbl@inisec[##1]\@@}%
2069 \global\let\bbl@renewlist\empty
2070 % Ends last section. See \bbl@inisec
2071 \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
2072 \@nameuse{\bbl@renew@\bbl@section}%
2073 \global\bbl@csarg\let{\renew@\bbl@section}\relax
2074 \@nameuse{\bbl@secpost@\bbl@section}%
2075 \fi}
2076 \def\bbl@iniline#1\bbl@iniline{%
2077 \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@@}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

2078 \def\bbl@iniskip#1\@@{% if starts with ;
2079 \def\bbl@inisec[#1]##2\@@{% if starts with opening bracket
2080 \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
2081 \@nameuse{\bbl@renew@\bbl@section}%
2082 \global\bbl@csarg\let{\renew@\bbl@section}\relax
2083 \@nameuse{\bbl@secpost@\bbl@section}% ends previous section
2084 \def\bbl@section{#1}% starts current section
2085 \def\bbl@elt##1##2{%
2086 \@namedef{\bbl@KVP@#1..##1}{}}%
2087 \@nameuse{\bbl@renew@#1}%
2088 \@nameuse{\bbl@secpre@#1}% pre-section `hook'
2089 \bbl@ifunset{\bbl@inikv@#1}%
2090 {\let\bbl@inireader\bbl@iniskip}%
2091 {\bbl@exp{\let\\bbl@inireader\<\bbl@inikv@#1>}}
2092 \let\bbl@renewlist\empty
2093 \def\bbl@renewinikey#1..##2\@@#3{%
2094 \bbl@ifunset{\bbl@renew@#1}%
2095 {\bbl@add@list\bbl@renewlist{#1}}%
2096 {}}%
2097 \bbl@csarg\bbl@add{\renew@#1}{\bbl@elt{#2}{#3}}

```

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

```

2098 \def\bbl@inikv#1=##2\@@{% key=value
2099 \bbl@trim@def\bbl@tempa{#1}%
2100 \bbl@trim\toks@{#2}%
2101 \bbl@csarg\edef{\bbl@kv@\bbl@section..\bbl@tempa}{\the\toks@}}

```

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

```

2102 \def\bbl@exportkey#1#2#3{%
2103   \bbl@ifunset{bbl@kv@#2}%
2104     {\bbl@csarg\gdef{#1@\language}\{#3}}%
2105     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2106       \bbl@csarg\gdef{#1@\language}\{#3}}%
2107     \else
2108       \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
2109     \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@secpost@identification` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

```

2110 \let\bbl@inikv@identification\bbl@inikv
2111 \def\bbl@secpost@identification{%
2112   \bbl@ifunset{bbl@kv@identification.name.opentype}%
2113     {\bbl@exportkey{lname}{identification.name.english}{}}%
2114     {\bbl@exportkey{lname}{identification.name.opentype}{}}%
2115   \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
2116   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2117   \bbl@ifunset{bbl@kv@identification.script.name.opentype}%
2118     {\bbl@exportkey{sname}{identification.script.name}{}}%
2119     {\bbl@exportkey{sname}{identification.script.name.opentype}{}}%
2120   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2121   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
2122 \let\bbl@inikv@typography\bbl@inikv
2123 \let\bbl@inikv@characters\bbl@inikv
2124 \let\bbl@inikv@numbers\bbl@inikv
2125 \def\bbl@after@ini{%
2126   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2127   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2128   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2129   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2130   \bbl@exportkey{intsp}{typography.intraspace}{}%
2131   \bbl@exportkey{jstfy}{typography.justify}{w}%
2132   \bbl@exportkey{chrng}{characters.ranges}{}%
2133   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2134   \bbl@exportkey{rqtex}{identification.require.babel}{}%
2135   \bbl@xin@{0.5}{\@nameuse{bbl@kv@identification.version}}%
2136   \ifin@
2137     \bbl@warning{%
2138       There are neither captions nor date in '\language'.\%
2139       It may not be suitable for proper typesetting, and it\%
2140       could change. Reported}%
2141   \fi
2142   \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
2143   \ifin@
2144     \bbl@warning{%
2145       The '\language' date format may not be suitable\%
2146       for proper typesetting, and therefore it very likely will\%
2147       change in a future release. Reported}%
2148   \fi
2149   \bbl@toglobal\bbl@savetoday
2150   \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.

```
2151 \ifcase\bbl@engine
2152   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
2153     \bbl@ini@captions@aux{#1}{#2}}
2154 \else
2155   \def\bbl@inikv@captions#1=#2\@@{%
2156     \bbl@ini@captions@aux{#1}{#2}}
2157 \fi
```

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

```
2158 \def\bbl@ini@captions@aux#1#2{%
2159   \bbl@trim@def\bbl@tempa{#1}%
2160   \bbl@ifblank{#2}%
2161   {\bbl@exp{%
2162     \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2163   {\bbl@trim\toks@{#2}}}%
2164   \bbl@exp{%
2165     \bbl@add\bbl@savestrings{%
2166       \SetString\<\bbl@tempa name>{\the\toks@}}}
```

But dates are more complex. The full date format is stores in date.gregorian, so we must read it in non-Unicode engines, too (saved months are just discarded when the LICR section is reached).

TODO. Remove copypaste pattern.

```
2167 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%           for defaults
2168   \bbl@inidate#1...\relax{#2}{}}
2169 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
2170   \bbl@inidate#1...\relax{#2}{islamic}}
2171 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
2172   \bbl@inidate#1...\relax{#2}{hebrew}}
2173 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
2174   \bbl@inidate#1...\relax{#2}{persian}}
2175 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
2176   \bbl@inidate#1...\relax{#2}{indian}}
2177 \ifcase\bbl@engine
2178   \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%      override
2179     \bbl@inidate#1...\relax{#2}{}}
2180   \bbl@csarg\def{secpre@date.gregorian.licr}{%             discard uni
2181     \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
2182 \fi
2183 % eg: 1=months, 2=wide, 3=1, 4=dummy
2184 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2185   \bbl@trim@def\bbl@tempa{#1.#2}%
2186   \bbl@ifsamestring{\bbl@tempa}{months.wide}%              to savestate
2187   {\bbl@trim@def\bbl@tempa{#3}%
2188     \bbl@trim\toks@{#5}%
2189     \bbl@exp{%
2190       \bbl@add\bbl@savestate{%
2191         \SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}%
2192   {\bbl@ifsamestring{\bbl@tempa}{date.long}%                 defined now
2193     {\bbl@trim@def\bbl@toreplace{#5}%
2194       \bbl@TG@date
2195       \global\bbl@csarg\let{date@\language}\bbl@toreplace
2196       \bbl@exp{%
2197         \gdef\<\language date>{\protect\<\language date >}}%
2198         \gdef\<\language date >###1###2###3{%
2199           \bbl@usedategrouptue
2200           \<\bbl@ensure@\language>{%
```

```

2201 \<bbl@date@language>{####1}{####2}{####3}}}%
2202 \\bbl@add\\bbl@savetoday{%
2203 \\SetString\\today{%
2204 \<language date>{\\the\year}{\\the\month}{\\the\day}}}%
2205 {}

```

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.

```

2206 \let\bbl@calendar\@empty
2207 \newcommand\BabelDateSpace{\nobreakspace}
2208 \newcommand\BabelDateDot{.\@}
2209 \newcommand\BabelDated[1]{\number#1}
2210 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
2211 \newcommand\BabelDateM[1]{\number#1}
2212 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2213 \newcommand\BabelDateMMMM[1]{%
2214 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
2215 \newcommand\BabelDatey[1]{\number#1}%
2216 \newcommand\BabelDateyy[1]{%
2217 \ifnum#1<10 0\number#1 %
2218 \else\ifnum#1<100 \number#1 %
2219 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2220 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2221 \else
2222 \bbl@error
2223 {Currently two-digit years are restricted to the\
2224 range 0-9999.}%
2225 {There is little you can do. Sorry.}%
2226 \fi\fi\fi}}
2227 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2228 \def\bbl@replace@finish@iii#1{%
2229 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
2230 \def\bbl@TG@date{%
2231 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
2232 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot}}%
2233 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
2234 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
2235 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
2236 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
2237 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
2238 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
2239 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
2240 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
2241 % Note after \bbl@replace \toks@ contains the resulting string.
2242 % TODO - Using this implicit behavior doesn't seem a good idea.
2243 \bbl@replace@finish@iii\bbl@toreplace}

```

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

```

2244 \def\bbl@provide@lsys#1{%
2245 \bbl@ifunset{bbl@lname@#1}%
2246 {\bbl@ini@basic{#1}}%
2247 {}%
2248 \bbl@csarg\let{lsys@#1}\@empty
2249 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
2250 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
2251 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2252 \bbl@ifunset{bbl@lname@#1}}%

```

```

2253     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2254     \bbl@csarg\bbl@tglobal{lsys@#1}}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too.

```

2255 \def\bbl@ini@basic#1{%
2256   \def\BabelBeforeIni##1##2{%
2257     \begingroup
2258       \bbl@add\bbl@secpost@identification{\closein1 }%
2259       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
2260       \bbl@read@ini{##1}{font and identification data}%
2261       \endinput           % babel- .tex may contain onlypreamble's
2262       \endgroup}%         boxed, to avoid extra spaces:
2263     {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}{}}
2264 % \section{Adjusting the Babel bahavior}
2265 %
2266 % \changes{babel~3.36}{2019/10/30}{New macro \cs{babeladjust}}
2267 %
2268 % A generic high level inteface is provided to adjust some global
2269 % and general settings.
2270 %
2271 %   \begin{macrocode}
2272 \newcommand\babeladjust[1]{% TODO. Error handling.
2273   \bbl@forkv{#1}{\@nameuse{\bbl@ADJ@##1@##2}}
2274 %
2275 \def\bbl@adjust@lua#1#2{%
2276   \ifvmode
2277     \ifnum\currentgrouplevel=\z@
2278       \directlua{ Babel.#2 }%
2279       \expandafter\expandafter\expandafter\@gobble
2280     \fi
2281     \fi
2282     {\bbl@error   % The error is gobbled if everything went ok.
2283       {Currently, #1 related features can be adjusted only\%
2284         in the main vertical list.}%
2285       {Maybe things change in the future, but this is what it is.}}}
2286 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
2287   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
2288 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
2289   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
2290 \@namedef{\bbl@ADJ@bidi.text@on}{%
2291   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
2292 \@namedef{\bbl@ADJ@bidi.text@off}{%
2293   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
2294 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
2295   \bbl@adjust@lua{bidi}{digits_mapped=true}}
2296 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
2297   \bbl@adjust@lua{bidi}{digits_mapped=false}}
2298 %
2299 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
2300   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
2301 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
2302   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
2303 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
2304   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
2305 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%

```

```

2306 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
2307 %
2308 \def\bbl@adjust@layout#1{%
2309   \ifvmode
2310     #1%
2311     \expandafter\@gobble
2312   \fi
2313   {\bbl@error % The error is gobbled if everything went ok.
2314     {Currently, layout related features can be adjusted only\\%
2315       in vertical mode.}%
2316     {Maybe things change in the future, but this is what it is.}}}
2317 \@namedef{bbl@ADJ@layout.tabular@on}{%
2318   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
2319 \@namedef{bbl@ADJ@layout.tabular@off}{%
2320   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
2321 \@namedef{bbl@ADJ@layout.lists@on}{%
2322   \bbl@adjust@layout{\let\list\bbl@NL@list}}
2323 \@namedef{bbl@ADJ@layout.lists@on}{%
2324   \bbl@adjust@layout{\let\list\bbl@OL@list}}
2325 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
2326   \directlua{
2327     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
2328   }}

```

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

### 10.1 The redefinition of the style commands

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

```

2329 {\def\format{plain}
2330 \ifx\fmtname\format
2331 \else
2332   \def\format{LaTeX2e}
2333   \ifx\fmtname\format
2334   \else
2335     \aftergroup\endinput
2336   \fi
2337 \fi}

```

### 10.2 Cross referencing macros

The  $\text{\LaTeX}$  book states:

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

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

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



The only way to accomplish this in most cases is to use the trick described in the *T<sub>E</sub>Xbook* [2] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, `'\meaning\A'` with `\A` defined as `'\def\A#1{\B}'` expands to the characters `'macro:#1->\B'` with all category codes set to `'other'` or `'space'`.

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

```
2338 %\bbl@redefine\newlabel#1#2{%
2339 % \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activfalse}
```

`\@newl@bel` We need to change the definition of the *T<sub>E</sub>X*-internal macro `\@newl@bel`. This is needed because we need to make sure that shorthand characters expand to their non-active version.

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

```
2340 <<(*More package options)>> ≡
2341 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2342 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2343 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2344 <</More package options>>
```

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

```
2345 \bbl@trace{Cross referencing macros}
2346 \ifx\bbl@opt@safe\@empty\else
2347 \def\@newl@bel#1#2#3{%
2348   {\@safe@activestrue
2349     \bbl@ifunset{#1@#2}%
2350     \relax
2351     {\gdef\@multiplelabels{%
2352       \@latex@warning@no@line{There were multiply-defined labels}}}%
2353     \@latex@warning@no@line{Label `#2' multiply defined}}%
2354   \global\@namedef{#1@#2}{#3}}}
```

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

```
2355 \CheckCommand*\@testdef[3]{%
2356   \def\reserved@a{#3}%
2357   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2358   \else
2359     \@tempwattrue
2360   \fi}
```

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

```
2361 \def\@testdef#1#2#3{%
2362   \@safe@activestrue
```

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

```
2363   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```
2364 \def\bbl@tempb{#3}%
2365 \@safe@activesfalse
```

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

```
2366 \ifx\bbl@tempa\relax
2367 \else
2368 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2369 \fi
```

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

```
2370 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```
2371 \ifx\bbl@tempa\bbl@tempb
2372 \else
2373 \@tempswatrue
2374 \fi}
2375 \fi
```

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

```
2376 \bbl@xin@{R}\bbl@opt@safe
2377 \ifin@
2378 \bbl@redefineroobust\ref{#1}{%
2379 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
2380 \bbl@redefineroobust\pageref{#1}{%
2381 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
2382 \else
2383 \let\org@ref\ref
2384 \let\org@pageref\pageref
2385 \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.

```
2386 \bbl@xin@{B}\bbl@opt@safe
2387 \ifin@
2388 \bbl@redefine\@citex[#1]#2{%
2389 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
2390 \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.

```
2391 \AtBeginDocument{%
2392 \@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.)

```

2393 \def\@citex[#1][#2]#3{%
2394   \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
2395   \org@citex[#1][#2]{\@tempa}}%
2396 }{}}

```

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

```

2397 \AtBeginDocument{%
2398   \@ifpackageloaded{cite}{%
2399     \def\@citex[#1]#2{%
2400       \@safe@activetrue\org@citex[#1][#2]\@safe@activesfalse}%
2401     }{}}

```

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

```

2402 \bbl@redefine\nocite#1{%
2403   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

2404 \bbl@redefine\bibcite{%
2405   \bbl@cite@choice
2406   \bibcite}

```

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

```

2407 \def\bbl@bibcite#1#2{%
2408   \org@bibcite{#1}{\@safe@activesfalse#2}}

```

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

```

2409 \def\bbl@cite@choice{%
2410   \global\let\bibcite\bbl@bibcite

```

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

```

2411 \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
2412 \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%

```

Make sure this only happens once.

```

2413 \global\let\bbl@cite@choice\relax}

```

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

```

2414 \AtBeginDocument{\bbl@cite@choice}

```

\@bibitem One of the two internal L<sup>A</sup>T<sub>E</sub>X macros called by \bibitem that write the citation label on the .aux file.

```

2415 \bbl@redefine\@bibitem#1{%
2416   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}

```

```

2417 \else
2418   \let\org@nocite\nocite
2419   \let\org@@citex\citex
2420   \let\org@bibcite\bibcite
2421   \let\org@bibitem\bibitem
2422 \fi

```

### 10.3 Marks

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

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

```

2423 \bbl@trace{Marks}
2424 \IfBabelLayout{sectioning}
2425   {\ifx\bbl@opt@headfoot\@nnil
2426     \g@addto@macro\@resetactivechars{%
2427       \set@typeset@protect
2428       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2429       \let\protect\noexpand
2430       \edef\thepage{%
2431         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2432     \fi}
2433   {\ifbbl@single\else
2434     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
2435     \markright#1{%
2436       \bbl@ifblank{#1}%
2437       {\org@markright{}}}%
2438     {\toks@{#1}%
2439     \bbl@exp{%
2440       \org@markright{\protect\foreignlanguage{\language}%
2441         {\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,  $\LaTeX$  stores the definition in an intermediate macros, so it's not necessary anymore, but it's preserved for older versions.)

```

2442   \ifx\@mkboth\markboth
2443     \def\bbl@tempc{\let\@mkboth\markboth}
2444   \else
2445     \def\bbl@tempc{}
2446   \fi
2447   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
2448   \markboth#1#2{%
2449     \protected@edef\bbl@tempb##1{%
2450       \protect\foreignlanguage
2451       {\language}{\protect\bbl@restore@actives##1}}%
2452   \bbl@ifblank{#1}%
2453   {\toks@{}}%
2454   {\toks@\expandafter{\bbl@tempb{#1}}}%

```

```

2455      \bbl@ifblank{#2}%
2456      {\@temptokena{}}%
2457      {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2458      \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}
2459      \bbl@tempc
2460      \fi} % end ifbbl@single, end \IfBabelLayout

```

## 10.4 Preventing clashes with other packages

### 10.4.1 ifthen

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

```

\ifthenelse{\isodd{\pageref{some:label}}}{
\code_for_odd_pages}
\code_for_even_pages}

```

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

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

```

2461 \bbl@trace{Preventing clashes with other packages}
2462 \bbl@xin@{R}\bbl@opt@safe
2463 \ifin@
2464   \AtBeginDocument{%
2465     \@ifpackageloaded{ifthen}{%

```

Then we can redefine `\ifthenelse`:

```

2466       \bbl@redefine@long\ifthenelse#1#2#3{%

```

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

```

2467       \let\bbl@temp@pref\pageref
2468       \let\pageref\org@pageref
2469       \let\bbl@temp@ref\ref
2470       \let\ref\org@ref

```

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. When the package wasn't loaded we do nothing.

```

2471       \@safe@activestrue
2472       \org@ifthenelse{#1}%
2473       {\let\pageref\bbl@temp@pref
2474        \let\ref\bbl@temp@ref
2475        \@safe@activesfalse
2476        #2}%
2477       {\let\pageref\bbl@temp@pref
2478        \let\ref\bbl@temp@ref
2479        \@safe@activesfalse
2480        #3}%
2481     }%
2482   }{}%
2483 }

```

### 10.4.2 varioref

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

```
2484 \AtBeginDocument{%
2485   \ifpackageloaded{varioref}{%
2486     \bbl@redefine\@@vpageref#1[#2]#3{%
2487       \@safe@activetrue
2488       \org@@vpageref{#1}[#2]#3}%
2489     \@safe@activesfalse}%
2490   \bbl@redefine\vrefpagemum#1#2{%
2491     \@safe@activetrue
2492     \org\vrefpagemum{#1}#2}%
2493   \@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.

```
2494   \expandafter\def\csname Ref \endcsname#1{%
2495     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2496   }{}%
2497 }
2498 \fi
```

### 10.4.3 hhline

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

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

```
2499 \AtEndOfPackage{%
2500   \AtBeginDocument{%
2501     \ifpackageloaded{hhline}%
```

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

```
2502     {\expandafter\ifx\csname normal@char:string\endcsname\relax
2503       \else
```

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

```
2504         \makeatletter
2505         \def\@currname{hhline}\input{hhline.sty}\makeatother
2506       \fi}%
2507     {}}}
```

### 10.4.4 hyperref

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

```
2508 \AtBeginDocument{%
```

```

2509 \ifx\pdfstringdefDisableCommands\@undefined\else
2510 \pdfstringdefDisableCommands{\languageshorthands{system}}%
2511 \fi}

```

#### 10.4.5 fancyhdr

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

```

2512 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2513 \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.

```

2514 \def\substitutefontfamily#1#2#3{%
2515 \lowercase{\immediate\openout15=#1#2.fd\relax}%
2516 \immediate\write15{%
2517 \string\ProvidesFile{#1#2.fd}%
2518 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2519 \space generated font description file]^^J
2520 \string\DeclareFontFamily{#1}{#2}{^^J
2521 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
2522 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
2523 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
2524 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
2525 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
2526 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
2527 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
2528 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
2529 }%
2530 \closeout15
2531 }

```

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

```

2532 \@onlypreamble\substitutefontfamily

```

### 10.5 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  and  $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$  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`

```

2533 \bbl@trace{Encoding and fonts}
2534 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
2535 \newcommand\BabelNonText{TS1,T3,TS3}
2536 \let\org@TeX\TeX
2537 \let\org@LaTeX\LaTeX
2538 \let\ensureascii@firstofone
2539 \AtBeginDocument{%
2540 \in@false

```

```

2541 \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2542   \ifin@ \else
2543     \lowercase{\bbl@xin@{, #1 enc. def, }{, \@filelist, } }%
2544   \fi}%
2545 \ifin@ % if a text non-ascii has been loaded
2546   \def\ensureasci#1{{\fontencoding{OT1}\selectfont#1}}%
2547   \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2548   \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2549   \def\bbl@tempb#1\@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
2550   \def\bbl@tempc#1ENC.DEF#2\@{\%
2551     \ifx\@empty#2\else
2552       \bbl@ifunset{T#1}%
2553       {}%
2554       {\bbl@xin@{, #1, }{, \BabelNonASCII, \BabelNonText, } }%
2555     \ifin@
2556       \DeclareTextCommand{\TeX}{#1}{\ensureasci{\org@TeX}}%
2557       \DeclareTextCommand{\LaTeX}{#1}{\ensureasci{\org@LaTeX}}%
2558     \else
2559       \def\ensureasci##1{{\fontencoding{#1}\selectfont##1}}%
2560     \fi}%
2561   \fi}%
2562 \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@ de mas??
2563 \bbl@xin@{, \cf@encoding, }{, \BabelNonASCII, \BabelNonText, }%
2564 \ifin@ \else
2565   \edef\ensureasci#1{%
2566     \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}%
2567   \fi
2568 \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.

```

2569 \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.

```

2570 \AtBeginDocument{%
2571   \@ifpackageloaded{fontspec}%
2572   {\xdef\latinencoding{%
2573     \ifx\UTFencname\@undefined
2574       EU\ifcase\bbl@engine\or2\or1\fi
2575     \else
2576       \UTFencname
2577     \fi}}%
2578   {\gdef\latinencoding{OT1}%
2579     \ifx\cf@encoding\bbl@t@one
2580       \xdef\latinencoding{\bbl@t@one}%
2581     \else
2582       \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
2583     \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.



```

2584 \DeclareRobustCommand{\latintext}{%
2585   \fontencoding{\latinencoding}\selectfont
2586   \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.

```

2587 \ifx\@undefined\DeclareTextFontCommand
2588   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2589 \else
2590   \DeclareTextFontCommand{\textlatin}{\latintext}
2591 \fi

```

## 10.6 Basic bidi support

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

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

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

- `pdfTeX` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour  $\TeX$  grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\TeX$ -ja` shows, vertical typesetting is possible, too. Its main drawback is font handling is often considered to be less mature than `xetex`, mainly in Indic scripts (but there are steps to make `HarfBuzz`, the `xetex` font engine, available in `luatex`; see <<https://github.com/tatzetwerk/luatex-harfbuzz>>).

```

2592 \bbl@trace{Basic (internal) bidi support}
2593 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2594 \def\bbl@rscripts{%
2595   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2596   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
2597   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
2598   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2599   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2600   Old South Arabian,%
2601 \def\bbl@provide@dirs#1{%
2602   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2603   \ifin@
2604     \global\bbl@csarg\chardef{wdir@#1}\@ne
2605     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2606     \ifin@
2607       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2608   \fi

```

```

2609 \else
2610   \global\bbbl@csarg\chardef{wdir@#1}\z@
2611 \fi
2612 \ifodd\bbbl@engine
2613   \bbbl@csarg\ifcase{wdir@#1}%
2614     \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
2615   \or
2616     \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
2617   \or
2618     \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
2619   \fi
2620 \fi}
2621 \def\bbbl@switchdir{%
2622   \bbbl@ifunset{bbbl@sys@\languagename}{\bbbl@provide@sys{\languagename}}{}%
2623   \bbbl@ifunset{bbbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}{}%
2624   \bbbl@exp{\bbbl@setdirs\bbbl@cs{wdir@\languagename}}%
2625 \def\bbbl@setdirs#1{% TODO - math
2626   \ifcase\bbbl@select@type % TODO - strictly, not the right test
2627     \bbbl@bodydir{#1}%
2628     \bbbl@pardir{#1}%
2629   \fi
2630   \bbbl@texmdir{#1}}
2631 \ifodd\bbbl@engine % luatex=1
2632   \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
2633   \DisableBabelHook{babel-bidi}
2634   \chardef\bbbl@thetexmdir\z@
2635   \chardef\bbbl@thepardir\z@
2636   \def\bbbl@getluadir#1{%
2637     \directlua{
2638       if tex.#1dir == 'TLT' then
2639         tex.sprint('0')
2640       elseif tex.#1dir == 'TRT' then
2641         tex.sprint('1')
2642       end}}
2643 \def\bbbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
2644   \ifcase#3\relax
2645     \ifcase\bbbl@getluadir{#1}\relax\else
2646       #2 TLT\relax
2647     \fi
2648   \else
2649     \ifcase\bbbl@getluadir{#1}\relax
2650       #2 TRT\relax
2651     \fi
2652   \fi}
2653 \def\bbbl@texmdir#1{%
2654   \bbbl@setluadir{text}\texmdir{#1}%
2655   \chardef\bbbl@thetexmdir#1\relax
2656   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*3+#1}}
2657 \def\bbbl@pardir#1{%
2658   \bbbl@setluadir{par}\pardir{#1}%
2659   \chardef\bbbl@thepardir#1\relax}
2660 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}
2661 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}
2662 \def\bbbl@dirparastext{\pardir\the\texmdir\relax}% %%%
2663 % Sadly, we have to deal with boxes in math with basic.
2664 % Activated every math with the package option bidi=:
2665 \def\bbbl@mathboxdir{%
2666   \ifcase\bbbl@thetexmdir\relax
2667     \everyhbox{\texmdir TLT\relax}%

```

```

2668 \else
2669 \everyhbox{\textdir TRT\relax}%
2670 \fi}
2671 \else % pdftex=0, xetex=2
2672 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2673 \DisableBabelHook{babel-bidi}
2674 \newcount\bbl@dirlevel
2675 \chardef\bbl@thetextdir\z@
2676 \chardef\bbl@thepardir\z@
2677 \def\bbl@textdir#1{%
2678 \ifcase#1\relax
2679 \chardef\bbl@thetextdir\z@
2680 \bbl@textdir@i\beginL\endL
2681 \else
2682 \chardef\bbl@thetextdir\@ne
2683 \bbl@textdir@i\beginR\endR
2684 \fi}
2685 \def\bbl@textdir@i#1#2{%
2686 \ifhmode
2687 \ifnum\currentgrouplevel>\z@
2688 \ifnum\currentgrouplevel=\bbl@dirlevel
2689 \bbl@error{Multiple bidi settings inside a group}%
2690 {I'll insert a new group, but expect wrong results.}%
2691 \bgroup\aftergroup#2\aftergroup\egroup
2692 \else
2693 \ifcase\currentgrouptype\or % 0 bottom
2694 \aftergroup#2% 1 simple {}
2695 \or
2696 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2697 \or
2698 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2699 \or\or\or % vbox vtop align
2700 \or
2701 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2702 \or\or\or\or\or\or % output math disc insert vcent mathchoice
2703 \or
2704 \aftergroup#2% 14 \begingroup
2705 \else
2706 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2707 \fi
2708 \fi
2709 \bbl@dirlevel\currentgrouplevel
2710 \fi
2711 #1%
2712 \fi}
2713 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2714 \let\bbl@bodydir\@gobble
2715 \let\bbl@pagedir\@gobble
2716 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

2717 \def\bbl@xebidipar{%
2718 \let\bbl@xebidipar\relax
2719 \TeXeTstate\@ne
2720 \def\bbl@xeverypar{%
2721 \ifcase\bbl@thepardir
2722 \ifcase\bbl@thetextdir\else\beginR\fi

```

```

2723 \else
2724   {\setbox\z@\lastbox\beginR\box\z@}%
2725   \fi}%
2726 \let\bbl@severypar\everypar
2727 \newtoks\everypar
2728 \everypar=\bbl@severypar
2729 \bbl@severypar{\bbl@xeverypar\the\everypar}}
2730 \@ifpackagewith{babel}{bidi=bidi}%
2731 {\let\bbl@textdir\i@gobbletwo
2732  \let\bbl@xebidipar\@empty
2733  \AddBabelHook{bidi}{foreign}{%
2734   \def\bbl@tempa{\def\BabelText####1}%
2735   \ifcase\bbl@thetextdir
2736     \expandafter\bbl@tempa\expandafter{\BabelText\LR{##1}}}%
2737   \else
2738     \expandafter\bbl@tempa\expandafter{\BabelText\RL{##1}}}%
2739   \fi}
2740 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}}
2741 {}%
2742 \fi

```

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

```

2743 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
2744 \AtBeginDocument{%
2745   \ifx\pdfstringdefDisableCommands\@undefined\else
2746     \ifx\pdfstringdefDisableCommands\relax\else
2747       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2748     \fi
2749   \fi}

```

## 10.7 Local Language Configuration

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

```

2750 \bbl@trace{Local Language Configuration}
2751 \ifx\loadlocalcfg\@undefined
2752   \@ifpackagewith{babel}{noconfigs}%
2753   {\let\loadlocalcfg\@gobble}%
2754   {\def\loadlocalcfg#1{%
2755     \InputIfFileExists{#1.cfg}%
2756     {\typeout{*****^J%
2757              * Local config file #1.cfg used^^J%
2758              *}}}%
2759     \@empty}}
2760 \fi

```

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

```

2761 \ifx\@unexpandable@protect\@undefined
2762   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2763   \long\def\protected@write#1#2#3{%
2764     \begingroup
2765       \let\thepage\relax
2766       #2%
2767       \let\protect\@unexpandable@protect

```

```

2768 \edef\reserved@a{\write#1{#3}}%
2769 \reserved@a
2770 \endgroup
2771 \if@nobreak\ifvmode\nobreak\fi\fi}
2772 \fi
2773 \</core>
2774 \<*kernel>

```

## 11 Multiple languages (switch.def)

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

```

2775 <<Make sure ProvidesFile is defined>>
2776 \ProvidesFile{switch.def}[\<date>] [\<version>] Babel switching mechanism]
2777 <<Load macros for plain if not LaTeX>>
2778 <<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.

```

2779 \def\bbl@version{\<version>}
2780 \def\bbl@date{\<date>}
2781 \def\adddialect#1#2{%
2782   \global\chardef#1#2\relax
2783   \bbl@usehooks{adddialect}{\#1}{\#2}}%
2784 \begingroup
2785   \count@#1\relax
2786   \def\bbl@elt##1##2##3##4{%
2787     \ifnum\count@=##2\relax
2788       \bbl@info{\string#1 = using hyphenrules for ##1\\%
2789         (\string\language\the\count@)}%
2790       \def\bbl@elt####1####2####3####4{%
2791         \fi}%
2792       \bbl@languages
2793     \endgroup}

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```

2794 \def\bbl@fixname#1{%
2795   \begingroup
2796   \def\bbl@tempe{l@}%
2797   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2798   \bbl@tempd
2799   {\lowercase\expandafter{\bbl@tempd}%
2800    {\uppercase\expandafter{\bbl@tempd}%
2801     \@empty
2802     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2803      {\uppercase\expandafter{\bbl@tempd}}}%
2804     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2805      {\lowercase\expandafter{\bbl@tempd}}}%
2806     \@empty

```

```

2807 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2808 \bbl@tempd}
2809 \def\bbl@iflanguage#1{%
2810 \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

```

2811 \def\iflanguage#1{%
2812 \bbl@iflanguage{#1}{%
2813 \ifnum\csname l@#1\endcsname=\language
2814 \expandafter\@firstoftwo
2815 \else
2816 \expandafter\@secondoftwo
2817 \fi}}

```

## 11.1 Selecting the language

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

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

```

2818 \let\bbl@select@type\z@
2819 \edef\selectlanguage{%
2820 \noexpand\protect
2821 \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`.

```

2822 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

2823 \ifx\documentclass\@undefined
2824 \def\xstring{\string\string\string}
2825 \else
2826 \let\xstring\string
2827 \fi

```

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

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

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

```
2828 \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`

```
2829 \def\bbl@push@language{%
2830   \xdef\bbl@language@stack{\language+\bbl@language@stack}}
```

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

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

```
2831 \def\bbl@pop@lang#1+#2-#3{%
2832   \edef\language{#1}\xdef#3{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before `\bbl@pop@lang` is executed T<sub>E</sub>X first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something has been pushed on the stack) followed by the '-'-sign and finally the reference to the stack.

```
2833 \let\bbl@ifrestoring\@secondoftwo
2834 \def\bbl@pop@language{%
2835   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2836   \let\bbl@ifrestoring\@firstoftwo
2837   \expandafter\bbl@set@language\expandafter{\language}%
2838   \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.

```
2839 \chardef\localeid\z@
2840 \def\bbl@id@last{0} % No real need for a new counter
2841 \def\bbl@id@assign{%
2842   \bbl@ifunset{bbl@id@@\language}%
```

```

2843 {\count@bbl@id@last\relax
2844 \advance\count@one
2845 \bbl@csarg\chardef{id@\language}\count@
2846 \edef\bbl@id@last{\the\count@}%
2847 \ifcase\bbl@engine\or
2848 \directlua{
2849     Babel = Babel or {}
2850     Babel.locale_props = Babel.locale_props or {}
2851     Babel.locale_props[\bbl@id@last] = {}
2852     Babel.locale_ids = Babel.locale_ids or {}
2853     Babel.locale_ids['\language'] = \bbl@id@last
2854 }%
2855 \fi}%
2856 {}%
2857 \chardef\localeid\nameuse{bbl@id@\language}}

```

The unprotected part of \selectlanguage.

```

2858 \expandafter\def\csname selectlanguage \endcsname#1{%
2859 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
2860 \bbl@push@language
2861 \aftergroup\bbl@pop@language
2862 \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.

```

2863 \def\BabelContentsFiles{toc,lof,lot}
2864 \def\bbl@set@language#1{% from selectlanguage, pop@
2865 \edef\language{%
2866 \ifnum\escapechar=\expandafter`\string#1\@empty
2867 \else\string#1\@empty\fi}%
2868 \select@language{\language}%
2869 % write to auxs
2870 \expandafter\ifx\csname date\language\endcsname\relax\else
2871 \if@filesw
2872 \ifx\babel@aux\@gobbles\else % Set if single in the first, redundant
2873 \protected@write\auxout{\string\babel@aux{\language}}}%
2874 \fi
2875 \bbl@usehooks{write}}}%
2876 \fi
2877 \fi}
2878 \def\select@language#1{% from set@, babel@aux
2879 % set hmap
2880 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2881 % set name
2882 \edef\language{#1}%
2883 \bbl@fixname\language
2884 \bbl@iflanguage\language{%
2885 \expandafter\ifx\csname date\language\endcsname\relax
2886 \bbl@error
2887 {Unknown language `#1'. Either you have\\%
2888 misspelled its name, it has not been installed,\\%
2889 or you requested it in a previous run. Fix its name,\\%

```



```

2890         install it or just rerun the file, respectively. In\\%
2891         some cases, you may need to remove the aux file}%
2892     {You may proceed, but expect wrong results}%
2893 \else
2894     % set type
2895     \let\bbl@select@type\z@
2896     \expandafter\bbl@switch\expandafter{\language}%
2897 \fi}}
2898 \def\babel@aux#1#2{%
2899 \expandafter\ifx\csname date#1\endcsname\relax
2900 \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
2901 \namedef{bbl@auxwarn@#1}{}%
2902 \bbl@warning
2903 {Unknown language `#1'. Very likely you\\%
2904 requested it in a previous run. Expect some\\%
2905 wrong results in this run, which should vanish\\%
2906 in the next one. Reported}%
2907 \fi
2908 \else
2909 \select@language{#1}%
2910 \bbl@foreach\BabelContentsFiles{%
2911 \writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
2912 \fi}
2913 \def\babel@toc#1#2{%
2914 \select@language{#1}}

```

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

```

2915 \let\select@language@x\select@language

```

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

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

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

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

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

```

2916 \newif\ifbbl@usedatagroup
2917 \def\bbl@switch#1{% from select@, foreign@
2918 % restore
2919 \originalTeX
2920 \expandafter\def\expandafter\originalTeX\expandafter{%
2921 \csname noextras#1\endcsname
2922 \let\originalTeX\@empty
2923 \babel@beginsave}%
2924 \bbl@usehooks{afterreset}{}%
2925 \languageshorthands{none}%
2926 % set the locale id
2927 \bbl@id@assign
2928 % switch captions, date

```

```

2929 \ifcase\bb1@select@type
2930   \ifhmode
2931     \hskip\z@skip % trick to ignore spaces
2932     \csname captions#1\endcsname\relax
2933     \csname date#1\endcsname\relax
2934     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2935   \else
2936     \csname captions#1\endcsname\relax
2937     \csname date#1\endcsname\relax
2938   \fi
2939 \else
2940   \ifbb1@usedategroup % if \foreign... within \<lang>date
2941     \bb1@usedategroupfalse
2942     \ifhmode
2943       \hskip\z@skip % trick to ignore spaces
2944       \csname date#1\endcsname\relax
2945       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2946     \else
2947       \csname date#1\endcsname\relax
2948     \fi
2949   \fi
2950 \fi
2951 % switch extras
2952 \bb1@usehooks{beforeextras}{}%
2953 \csname extras#1\endcsname\relax
2954 \bb1@usehooks{afterextras}{}%
2955 % > babel-ensure
2956 % > babel-sh-<short>
2957 % > babel-bidi
2958 % > babel-fontspec
2959 % hyphenation - case mapping
2960 \ifcase\bb1@opt@hyphenmap\or
2961   \def\BabelLower##1##2{\lccode##1=##2\relax}%
2962   \ifnum\bb1@hymapsel>4\else
2963     \csname\language @bb1@hyphenmap\endcsname
2964   \fi
2965   \chardef\bb1@opt@hyphenmap\z@
2966 \else
2967   \ifnum\bb1@hymapsel>\bb1@opt@hyphenmap\else
2968     \csname\language @bb1@hyphenmap\endcsname
2969   \fi
2970 \fi
2971 \global\let\bb1@hymapsel\@cclv
2972 % hyphenation - patterns
2973 \bb1@patterns{#1}%
2974 % hyphenation - mins
2975 \babel@savevariable\lefthyphenmin
2976 \babel@savevariable\righthyphenmin
2977 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2978   \set@hyphenmins\tw@\thr@@\relax
2979 \else
2980   \expandafter\expandafter\expandafter\set@hyphenmins
2981   \csname #1hyphenmins\endcsname\relax
2982 \fi}

```

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

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

```
2983 \long\def\otherlanguage#1{%
2984   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
2985   \csname selectlanguage \endcsname{#1}%
2986   \ignorespaces}
```

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

```
2987 \long\def\endotherlanguage{%
2988   \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`.

```
2989 \expandafter\def\csname otherlanguage*\endcsname#1{%
2990   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2991   \foreign@language{#1}}
```

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

```
2992 \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`.

```
2993 \providecommand\bbl@beforeforeign{}
2994 \edef\foreignlanguage{%
2995   \noexpand\protect
2996   \expandafter\noexpand\csname foreignlanguage \endcsname}
2997 \expandafter\def\csname foreignlanguage \endcsname{%
2998   \@ifstar\bbl@foreign@s\bbl@foreign@x}
2999 \def\bbl@foreign@x#1#2{%
3000   \begingroup
3001   \let\BabelText\@firstofone
```

```

3002 \bbl@beforeforeign
3003 \foreign@language{#1}%
3004 \bbl@usehooks{foreign}{}%
3005 \BabelText{#2}% Now in horizontal mode!
3006 \endgroup}
3007 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
3008 \begingroup
3009 {\par}%
3010 \let\BabelText\@firstofone
3011 \foreign@language{#1}%
3012 \bbl@usehooks{foreign*}{}%
3013 \bbl@dirparastext
3014 \BabelText{#2}% Still in vertical mode!
3015 {\par}%
3016 \endgroup}

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the other `language*` 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`.

```

3017 \def\foreign@language#1{%
3018 % set name
3019 \edef\language#1}%
3020 \bbl@fixname\language
3021 \bbl@iflanguage\language{%
3022 \expandafter\ifx\csname date\language\endcsname\relax
3023 \bbl@warning % TODO - why a warning, not an error?
3024 {Unknown language `#1'. Either you have\\%
3025 misspelled its name, it has not been installed,\\%
3026 or you requested it in a previous run. Fix its name,\\%
3027 install it or just rerun the file, respectively. In\\%
3028 some cases, you may need to remove the aux file.\\%
3029 I'll proceed, but expect wrong results.\\%
3030 Reported}%
3031 \fi
3032 % set type
3033 \let\bbl@select@type\@ne
3034 \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\lcode'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.

```

3035 \let\bbl@hyphlist\@empty
3036 \let\bbl@hyphenation@\relax
3037 \let\bbl@pttnlist\@empty
3038 \let\bbl@patterns@\relax
3039 \let\bbl@hymapsel=\@cclv
3040 \def\bbl@patterns#1{%
3041 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3042 \csname l@#1\endcsname
3043 \edef\bbl@tempa{#1}%
3044 \else

```

```

3045 \csname l@#1:\f@encoding\endcsname
3046 \edef\bbl@tempa{#1:\f@encoding}%
3047 \fi
3048 \@expandtwoargs\bbl@usehooks{patterns}{\bbl@tempa}%
3049 % > luatex
3050 \@ifundefined{bbl@hyphenation@}{\% Can be \relax!
3051 \begingroup
3052 \bbl@xin@{\, \number\language,}{\, \bbl@hyphlist}%
3053 \ifin@else
3054 \@expandtwoargs\bbl@usehooks{hyphenation}{\bbl@tempa}%
3055 \hyphenation{%
3056 \bbl@hyphenation@
3057 \@ifundefined{bbl@hyphenation@#1}%
3058 \@empty
3059 {\space\csname bbl@hyphenation@#1\endcsname}}%
3060 \xdef\bbl@hyphlist{\bbl@hyphlist\language,}%
3061 \fi
3062 \endgroup}}

```

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

```

3063 \def\hyphenrules#1{%
3064 \edef\bbl@tempf{#1}%
3065 \bbl@fixname\bbl@tempf
3066 \bbl@iflanguage\bbl@tempf{%
3067 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
3068 \languageshortands{none}%
3069 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
3070 \set@hyphenmins\tw@\thr@@\relax
3071 \else
3072 \expandafter\expandafter\expandafter\set@hyphenmins
3073 \csname\bbl@tempf hyphenmins\endcsname\relax
3074 \fi}}
3075 \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.

```

3076 \def\providehyphenmins#1#2{%
3077 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3078 \@namedef{#1hyphenmins}{#2}%
3079 \fi}

```

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

```

3080 \def\set@hyphenmins#1#2{%
3081 \lefthyphenmin#1\relax
3082 \righthyphenmin#2\relax}

```

**\ProvidesLanguage** The identification code for each file is something that was introduced in L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by `babel`. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

```

3083 \ifx\ProvidesFile\@undefined
3084 \def\ProvidesLanguage#1[#2 #3 #4]{%

```

```

3085 \wlog{Language: #1 #4 #3 <#2>}%
3086 }
3087 \else
3088 \def\ProvidesLanguage#1{%
3089 \begingroup
3090 \catcode`\ 10 %
3091 \@makeother\/%
3092 \@ifnextchar[%]
3093 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
3094 \def\@provideslanguage#1[#2]{%
3095 \wlog{Language: #1 #2}%
3096 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
3097 \endgroup}
3098 \fi

```

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

The category code of the ampersand is restored and the macro calls itself again with the new definition from `babel.def`

```

3099 \def\LdfInit{%
3100 \chardef\atcatcode=\catcode`\@
3101 \catcode`\@=11\relax
3102 \input babel.def\relax
3103 \catcode`\@=\atcatcode \let\atcatcode\relax
3104 \LdfInit}

```

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

```
3105 \ifx\originalTeX\undefined\let\originalTeX\@empty\fi
```

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

```
3106 \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’:

```

3107 \providecommand\setlocale{%
3108 \bbl@error
3109 {Not yet available}%
3110 {Find an armchair, sit down and wait}}
3111 \let\uselocale\setlocale
3112 \let\locale\setlocale
3113 \let\selectlocale\setlocale
3114 \let\textlocale\setlocale
3115 \let\textlanguage\setlocale
3116 \let\languagetext\setlocale

```

## 11.2 Errors

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

`\@nopatterns`

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

```

3117 \edef\bbl@nulllanguage{\string\language=0}
3118 \ifx\PackageError\undefined
3119   \def\bbl@error#1#2{%
3120     \begingroup
3121       \newlinechar=`^^J
3122       \def\{^^J(babel) }%
3123       \errhelp{#2}\errmessage{\{#1}%
3124     \endgroup}
3125   \def\bbl@warning#1{%
3126     \begingroup
3127       \newlinechar=`^^J
3128       \def\{^^J(babel) }%
3129       \message{\{#1}%
3130     \endgroup}
3131   \let\bbl@infowarn\bbl@warning
3132   \def\bbl@info#1{%
3133     \begingroup
3134       \newlinechar=`^^J
3135       \def\{^^J}%
3136       \wlog{#1}%
3137     \endgroup}
3138 \else
3139   \def\bbl@error#1#2{%
3140     \begingroup
3141       \def\{\MessageBreak}%
3142       \PackageError{babel}{#1}{#2}%
3143     \endgroup}
3144   \def\bbl@warning#1{%
3145     \begingroup
3146       \def\{\MessageBreak}%
3147       \PackageWarning{babel}{#1}%
3148     \endgroup}
3149   \def\bbl@infowarn#1{%
3150     \begingroup
3151       \def\{\MessageBreak}%
3152       \GenericWarning
3153         {(babel) \@spaces\@spaces\@spaces}%
3154         {Package babel Info: #1}%
3155     \endgroup}
3156   \def\bbl@info#1{%
3157     \begingroup
3158       \def\{\MessageBreak}%
3159       \PackageInfo{babel}{#1}%
3160     \endgroup}
3161 \fi
3162 \@ifpackagewith{babel}{silent}
3163   {\let\bbl@info@gobble
3164   \let\bbl@infowarn@gobble
3165   \let\bbl@warning@gobble}
3166 {}
3167 \def\bbl@nocaption{\protect\bbl@nocaption@i}
3168 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
3169   \global\@namedef{#2}{\textbf{?#1?}}%
3170   \@nameuse{#2}%

```

```

3171 \bbl@warning{%
3172   \@backslashchar#2 not set. Please, define\\%
3173   it in the preamble with something like:\\%
3174   \string\renewcommand\@backslashchar#2{..}\\%
3175   Reported}}
3176 \def\bbl@tentative{\protect\bbl@tentative@i}
3177 \def\bbl@tentative@i#1{%
3178   \bbl@warning{%
3179     Some functions for '#1' are tentative.\\%
3180     They might not work as expected and their behavior\\%
3181     could change in the future.\\%
3182     Reported}}
3183 \def\@nolanerr#1{%
3184   \bbl@error
3185   {You haven't defined the language #1\space yet}%
3186   {Your command will be ignored, type <return> to proceed}}
3187 \def\@nopatterns#1{%
3188   \bbl@warning
3189   {No hyphenation patterns were preloaded for\\%
3190    the language '#1' into the format.\\%
3191    Please, configure your TeX system to add them and\\%
3192    rebuild the format. Now I will use the patterns\\%
3193    preloaded for \bbl@nulllanguage\space instead}}
3194 \let\bbl@usehooks\@gobbletwo
3195 \</kernel>
3196 \<*patterns>

```

## 12 Loading hyphenation patterns

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

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

```

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

```

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before  $\text{WTeX}$  fills the register. There are some problems with this approach though.

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

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



To make sure that L<sup>A</sup>T<sub>E</sub>X 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

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

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

```

3197 <<Make sure ProvidesFile is defined>>
3198 \ProvidesFile{hyphen.cfg}[\<<date>>] \<<version>> Babel hyphens]
3199 \xdef\bbl@format{\jobname}
3200 \ifx\AtBeginDocument\@undefined
3201   \def\@empty{}
3202   \let\orig@dump\dump
3203   \def\dump{%
3204     \ifx\@ztryfc\@undefined
3205       \else
3206         \toks0=\expandafter{\@preamblecmds}%
3207         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3208         \def\@begindocumenthook{}%
3209       \fi
3210       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3211 \fi
3212 <<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.

```

3213 \def\process@line#1#2 #3 #4 {%
3214   \ifx=#1%
3215     \process@synonym{#2}%
3216   \else
3217     \process@language{#1#2}{#3}{#4}%
3218   \fi
3219   \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.

```

3220 \toks@{}
3221 \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.

```

3222 \def\process@synonym#1{%
3223   \ifnum\last@language=\m@ne
3224     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3225   \else
3226     \expandafter\chardef\csname l@#1\endcsname\last@language
3227     \wlog{\string\l@#1=\string\language\the\last@language}%
3228     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
3229     \csname\language\endcsname hyphenmins\endcsname
3230     \let\bbl@elt\relax

```

```

3231 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
3232 \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.

```

3233 \def\process@language#1#2#3{%
3234 \expandafter\addlanguage\csname l@#1\endcsname
3235 \expandafter\language\csname l@#1\endcsname
3236 \edef\language#1{%
3237 \bbl@hook@everylanguage{#1}%
3238 % > luatex
3239 \bbl@get@enc#1::@@@
3240 \begingroup
3241 \lefthyphenmin@m@ne
3242 \bbl@hook@loadpatterns{#2}%
3243 % > luatex
3244 \ifnum\lefthyphenmin=\m@ne
3245 \else
3246 \expandafter\xdef\csname #1hyphenmins\endcsname{%
3247 \the\lefthyphenmin\the\righthyphenmin}%
3248 \fi
3249 \endgroup
3250 \def\bbl@tempa{#3}%
3251 \ifx\bbl@tempa\@empty\else
3252 \bbl@hook@loadexceptions{#3}%
3253 % > luatex
3254 \fi
3255 \let\bbl@elt\relax

```

```

3256 \edef\bbl@languages{%
3257   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3258 \ifnum\the\language=\z@
3259   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3260     \set@hyphenmins\tw@\thr@@\relax
3261   \else
3262     \expandafter\expandafter\expandafter\set@hyphenmins
3263       \csname #1hyphenmins\endcsname
3264   \fi
3265   \the\toks@
3266   \toks@{}}%
3267 \fi}

```

\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. It uses delimited arguments to achieve this.

```

3268 \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.

```

3269 \def\bbl@hook@everylanguage#1{}
3270 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3271 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3272 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
3273 \begingroup
3274   \def\AddBabelHook#1#2{%
3275     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3276       \def\next{\toks1}%
3277     \else
3278       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
3279     \fi
3280     \next}
3281 \ifx\directlua\@undefined
3282   \ifx\XeTeXinputencoding\@undefined\else
3283     \input xebabel.def
3284   \fi
3285 \else
3286   \input luababel.def
3287 \fi
3288 \openin1 = babel-\bbl@format.cfg
3289 \ifeof1
3290 \else
3291   \input babel-\bbl@format.cfg\relax
3292 \fi
3293 \closein1
3294 \endgroup
3295 \bbl@hook@loadkernel{switch.def}

```

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

```

3296 \openin1 = language.dat

```

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

```

3297 \def\languagename{english}%
3298 \ifeof1
3299   \message{I couldn't find the file language.dat,\space
3300     I will try the file hyphen.tex}
3301   \input hyphen.tex\relax
3302   \chardef\l@english\z@
3303 \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`.

```
3304 \last@language\m@ne
```

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

```
3305 \loop
```

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

```
3306 \endlinechar\m@ne
```

```
3307 \read1 to \bbl@line
```

```
3308 \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.

```
3309 \if T\ifeof1F\fi T\relax
```

```
3310 \ifx\bbl@line\@empty\else
```

```
3311 \edef\bbl@line{\bbl@line\space\space\space}%
```

```
3312 \expandafter\process@line\bbl@line\relax
```

```
3313 \fi
```

```
3314 \repeat
```

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

```
3315 \begingroup
```

```
3316 \def\bbl@elt#1#2#3#4{%
```

```
3317 \global\language=#2\relax
```

```
3318 \gdef\language#1}%
```

```
3319 \def\bbl@elt##1##2##3##4{}}%
```

```
3320 \bbl@languages
```

```
3321 \endgroup
```

```
3322 \fi
```

and close the configuration file.

```
3323 \closein1
```

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

```
3324 \if/\the\toks@\else
```

```
3325 \errhelp{language.dat loads no language, only synonyms}
```

```
3326 \errmessage{Orphan language synonym}
```

```
3327 \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.

```
3328 \let\bbl@line\@undefined
```

```
3329 \let\process@line\@undefined
```

```
3330 \let\process@synonym\@undefined
```

```
3331 \let\process@language\@undefined
```

```
3332 \let\bbl@get@enc\@undefined
```

```
3333 \let\bbl@hyph@enc\@undefined
```

```
3334 \let\bbl@tempa\@undefined
```

```
3335 \let\bbl@hook@loadkernel\@undefined
```

```
3336 \let\bbl@hook@everylanguage\@undefined
```

```

3337 \let\bbl@hook@loadpatterns\@undefined
3338 \let\bbl@hook@loadexceptions\@undefined
3339 \end{patterns}

```

Here the code for `iniTEX` ends.

## 13 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].

```

3340 <<{*More package options}>> ≡
3341 \ifodd\bbl@engine
3342   \DeclareOption{bidi=basic-r}%
3343   {\ExecuteOptions{bidi=basic}}
3344   \DeclareOption{bidi=basic}%
3345   {\let\bbl@beforeforeign\leavevmode
3346    % TODO - to locale_props, not as separate attribute
3347    \newattribute\bbl@attr@dir
3348    % I don't like it, hackish:
3349    \frozen@everymath\expandafter{%
3350      \expandafter\bbl@mathboxdir\the\frozen@everymath}%
3351    \frozen@everydisplay\expandafter{%
3352      \expandafter\bbl@mathboxdir\the\frozen@everydisplay}%
3353    \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
3354    \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
3355 \else
3356   \DeclareOption{bidi=basic-r}%
3357   {\ExecuteOptions{bidi=basic}}
3358   \DeclareOption{bidi=basic}%
3359   {\bbl@error
3360    {The bidi method 'basic' is available only in\\
3361     luatex. I'll continue with 'bidi=default', so\\
3362     expect wrong results}%
3363    {See the manual for further details.}%
3364    \let\bbl@beforeforeign\leavevmode
3365    \AtEndOfPackage{%
3366      \EnableBabelHook{babel-bidi}%
3367      \bbl@xebidipar}}
3368   \def\bbl@loadxebidi#1{%
3369     \ifx\RTLfootnotetext\@undefined
3370       \AtEndOfPackage{%
3371         \EnableBabelHook{babel-bidi}%
3372         \ifx\fontspec\@undefined
3373           \usepackage{fontspec}% bidi needs fontspec
3374         \fi
3375         \usepackage#1{bidi}}%
3376       \fi}
3377   \DeclareOption{bidi=bidi}%
3378   {\bbl@tentative{bidi=bidi}%
3379    \bbl@loadxebidi{}}
3380   \DeclareOption{bidi=bidi-r}%
3381   {\bbl@tentative{bidi=bidi-r}%
3382    \bbl@loadxebidi{[rldocument]}}
3383   \DeclareOption{bidi=bidi-l}%
3384   {\bbl@tentative{bidi=bidi-l}%
3385    \bbl@loadxebidi{}}
3386 \fi

```

```

3387 \DeclareOption{bidi=default}%
3388 {\let\bbl@beforeforeign\leavevmode
3389 \ifodd\bbl@engine
3390 \newattribute\bbl@attr@dir
3391 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
3392 \fi
3393 \AtEndOfPackage{%
3394 \EnableBabelHook{babel-bidi}%
3395 \ifodd\bbl@engine\else
3396 \bbl@xebidipar
3397 \fi}}
3398 <</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`.

```

3399 << *Font selection>> ≡
3400 \bbl@trace{Font handling with fontspec}
3401 \@onlypreamble\babelfont
3402 \newcommand\babelfont[2][% 1=langs/scripts 2=fam
3403 \edef\bbl@tempa{#1}%
3404 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3405 \ifx\fontspec\undefined
3406 \usepackage{fontspec}%
3407 \fi
3408 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3409 \bbl@bblfont}
3410 \newcommand\bbl@bblfont[2][% 1=features 2=fontname, @font=rm|sf|tt
3411 \bbl@ifunset{\bbl@tempb family}%
3412 {\bbl@providedefam{\bbl@tempb}}%
3413 {\bbl@exp{%
3414 \\\bbl@sreplace\<\bbl@tempb family >%
3415 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3416 % For the default font, just in case:
3417 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
3418 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3419 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save \bbl@rmdflt@
3420 \bbl@exp{%
3421 \let\<\bbl@tempb dflt@\languagename>\<\bbl@tempb dflt@>%
3422 \\\bbl@font@set\<\bbl@tempb dflt@\languagename>%
3423 \<\bbl@tempb default>\<\bbl@tempb family>}}%
3424 {\bbl@foreach\bbl@tempa{% ie \bbl@rmdflt@lang / *scrt
3425 \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:

```

3426 \def\bbl@providedefam#1{%
3427 \bbl@exp{%
3428 \\\newcommand\<#1default>{}% Just define it
3429 \\\bbl@add@list\<\bbl@font@fams{#1}%
3430 \\\DeclareRobustCommand\<#1family>%
3431 \\\not@math@alphabet\<#1family>\relax
3432 \\\fontfamily\<#1default>\selectfont}%
3433 \\\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.

```

3434 \def\bbl@nostdfont#1{%
3435 \bbl@ifunset{\bbl@WFF@\f@family}%
3436 {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns

```

```

3437 \bbl@infowarn{The current font is not a babel standard family:\%
3438 #1%
3439 \fontname\font\%
3440 There is nothing intrinsically wrong with this warning, and\%
3441 you can ignore it altogether if you do not need these\%
3442 families. But if they are used in the document, you should be\%
3443 aware 'babel' will no set Script and Language for them, so\%
3444 you may consider defining a new family with \string\babelfont.\%
3445 See the manual for further details about \string\babelfont.\%
3446 Reported}}
3447 {}}%
3448 \gdef\bbl@switchfont{%
3449 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}}%
3450 \bbl@exp{% eg Arabic -> arabic
3451 \lowercase{\edef\bbbl@tempa{\bbl@cs{sname@\language}}}%
3452 \bbl@foreach\bbl@font@fams{%
3453 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
3454 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
3455 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
3456 {}% 123=F - nothing!
3457 {\bbl@exp{% 3=T - from generic
3458 \global\let<\bbl@##1dflt@\language>%
3459 \<\bbl@##1dflt@>}}}%
3460 {\bbl@exp{% 2=T - from script
3461 \global\let<\bbl@##1dflt@\language>%
3462 \<\bbl@##1dflt@*\bbl@tempa>}}}%
3463 {}}% 1=T - language, already defined
3464 \def\bbl@tempa{\bbl@nostdfont}}}%
3465 \bbl@foreach\bbl@font@fams{% don't gather with prev for
3466 \bbl@ifunset{\bbl@##1dflt@\language}%
3467 {\bbl@cs{famrst@##1}%
3468 \global\bbl@csarg\let{famrst@##1}\relax}%
3469 {\bbl@exp{% order is relevant
3470 \bbl@add\originalTeX{%
3471 \bbl@font@rst{\bbl@cs{##1dflt@\language}}%
3472 \<##1default>\<##1family>{##1}}%
3473 \bbl@font@set<\bbl@##1dflt@\language>% the main part!
3474 \<##1default>\<##1family>}}}%
3475 \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.

```

3476 \ifx\f@family\undefined\else % if latex
3477 \ifcase\bbl@engine % if pdftex
3478 \let\bbl@ckeckstdfonts\relax
3479 \else
3480 \def\bbl@ckeckstdfonts{%
3481 \begingroup
3482 \global\let\bbl@ckeckstdfonts\relax
3483 \let\bbl@tempa\empty
3484 \bbl@foreach\bbl@font@fams{%
3485 \bbl@ifunset{\bbl@##1dflt@}%
3486 {\nameuse{##1family}%
3487 \bbl@csarg\gdef{WFF@f@family}}}% Flag
3488 \bbl@exp{\bbl@add\bbbl@tempa{* \<##1family>= \f@family\\}%
3489 \space\space\fontname\font\\}%
3490 \bbl@csarg\xdef{##1dflt@}{\f@family}%
3491 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
3492 {}}%

```

```

3493     \ifx\bbbl@tempa\@empty\else
3494     \bbbl@infowarn{The following fonts are not babel standard families:\%
3495     \bbbl@tempa
3496     There is nothing intrinsically wrong with it, but\%
3497     'babel' will no set Script and Language. Consider\%
3498     defining a new family with \string\babelfont.\%
3499     Reported}%
3500     \fi
3501 \endgroup}
3502 \fi
3503 \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 \bbbl@mapselect because \selectfont is called internally when a font is defined.

```

3504 \def\bbbl@font@set#1#2#3{% eg \bbbl@rmdflt@lang \rmdefault \rmfamily
3505 \bbbl@xin@{<>}{#1}%
3506 \ifin@
3507 \bbbl@exp{\bbbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
3508 \fi
3509 \bbbl@exp{%
3510 \def\#2{#1}% eg, \rmdefault{\bbbl@rmdflt@lang}
3511 \bbbl@ifsamestring{#2}{\f@family}{\#3\let\bbbl@tempa\relax}{}}
3512 % TODO - next should be global?, but even local does its job. I'm
3513 % still not sure -- must investigate:
3514 \def\bbbl@fontspec@set#1#2#3#4{% eg \bbbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
3515 \let\bbbl@tempa\bbbl@mapselect
3516 \let\bbbl@mapselect\relax
3517 \let\bbbl@temp@fam#4% eg, '\rmfamily', to be restored below
3518 \let#4\relax % So that can be used with \newfontfamily
3519 \bbbl@exp{%
3520 \let\bbbl@temp@pfam<\bbbl@stripslash#4\space>% eg, '\rmfamily '
3521 <\keys_if_exist:nnF>{fontspec-opentype}%
3522 {Script/\bbbl@cs{sname@languagename}}%
3523 {\newfontscript{\bbbl@cs{sname@languagename}}%
3524 {\bbbl@cs{sotf@languagename}}}%
3525 <\keys_if_exist:nnF>{fontspec-opentype}%
3526 {Language/\bbbl@cs{lname@languagename}}%
3527 {\newfontlanguage{\bbbl@cs{lname@languagename}}%
3528 {\bbbl@cs{lotf@languagename}}}%
3529 \newfontfamily\#4%
3530 [\bbbl@cs{lsys@languagename},#2]{#3}% ie \bbbl@exp{.}{#3}
3531 \begingroup
3532 #4%
3533 \xdef#1{\f@family}% eg, \bbbl@rmdflt@lang{FreeSerif(0)}
3534 \endgroup
3535 \let#4\bbbl@temp@fam
3536 \bbbl@exp{\let<\bbbl@stripslash#4\space>\bbbl@temp@pfam
3537 \let\bbbl@mapselect\bbbl@tempa}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```

3538 \def\bbbl@font@rst#1#2#3#4{%
3539 \bbbl@csarg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.



```
3540 \def\bbl@font@fams{rm,sf,tt}
```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for `\babelFSfeatures`. The reason is explained in the user guide, but essentially – that was not the way to go :-).

```
3541 \newcommand\babelFSstore[2][{}]{%
3542   \bbl@ifblank{#1}%
3543   {\bbl@csarg\def{sname@#2}{Latin}}%
3544   {\bbl@csarg\def{sname@#2}{#1}}%
3545   \bbl@provide@dirs{#2}%
3546   \bbl@csarg\ifnum{wdir@#2}>\z@
3547     \let\bbl@beforeforeign\leavevmode
3548     \EnableBabelHook{babel-bidi}%
3549   \fi
3550   \bbl@foreach{#2}{%
3551     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
3552     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
3553     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
3554 \def\bbl@FSstore#1#2#3#4{%
3555   \bbl@csarg\edef{#2default#1}{#3}%
3556   \expandafter\addto\csname extras#1\endcsname{%
3557     \let#4#3%
3558     \ifx#3\f@family
3559       \edef#3{\csname bbl@#2default#1\endcsname}%
3560       \fontfamily{#3}\selectfont
3561     \else
3562       \edef#3{\csname bbl@#2default#1\endcsname}%
3563     \fi}%
3564   \expandafter\addto\csname noextras#1\endcsname{%
3565     \ifx#3\f@family
3566       \fontfamily{#4}\selectfont
3567     \fi
3568     \let#3#4}}
3569 \let\bbl@langfeatures\@empty
3570 \def\babelFSfeatures{% make sure \fontspec is redefined once
3571   \let\bbl@ori@fontspec\fontspec
3572   \renewcommand\fontspec[1][{}]{%
3573     \bbl@ori@fontspec[\bbl@langfeatures##1]}
3574   \let\babelFSfeatures\bbl@FSfeatures
3575   \babelFSfeatures}
3576 \def\bbl@FSfeatures#1#2{%
3577   \expandafter\addto\csname extras#1\endcsname{%
3578     \babel@save\bbl@langfeatures
3579     \edef\bbl@langfeatures{#2,}}
3580 \<</Font selection>>
```

## 14 Hooks for XeTeX and LuaTeX

### 14.1 XeTeX

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

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

```
3581 \<<*Restore Unicode catcodes before loading patterns>> \equiv
```

```

3582 \begingroup
3583   % Reset chars "80-"C0 to category "other", no case mapping:
3584   \catcode`\@=11 \count@=128
3585   \loop\ifnum\count@<192
3586     \global\uccode\count@=0 \global\lccode\count@=0
3587     \global\catcode\count@=12 \global\sffcode\count@=1000
3588     \advance\count@ by 1 \repeat
3589   % Other:
3590   \def\O ##1 {%
3591     \global\uccode"##1=0 \global\lccode"##1=0
3592     \global\catcode"##1=12 \global\sffcode"##1=1000 }%
3593   % Letter:
3594   \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3595     \global\uccode"##1="##2
3596     \global\lccode"##1="##3
3597     % Uppercase letters have sffcode=999:
3598     \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
3599   % Letter without case mappings:
3600   \def\l ##1 {\L ##1 ##1 ##1 }%
3601   \l 00AA
3602   \L 00B5 039C 00B5
3603   \l 00BA
3604   \O 00D7
3605   \l 00DF
3606   \O 00F7
3607   \L 00FF 0178 00FF
3608 \endgroup
3609 \input #1\relax
3610 <</Restore Unicode catcodes before loading patterns>>

```

Some more common code.

```

3611 <<{*Footnote changes}>> ≡
3612 \bbl@trace{Bidi footnotes}
3613 \ifx\bbl@beforeforeign\leavevmode
3614   \def\bbl@footnote#1#2#3{%
3615     \@ifnextchar[%
3616       {\bbl@footnote@o{#1}{#2}{#3}}%
3617       {\bbl@footnote@x{#1}{#2}{#3}}}
3618   \def\bbl@footnote@x#1#2#3#4{%
3619     \bgroup
3620     \select@language@x{\bbl@main@language}%
3621     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3622     \egroup}
3623   \def\bbl@footnote@o#1#2#3[#4]#5{%
3624     \bgroup
3625     \select@language@x{\bbl@main@language}%
3626     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3627     \egroup}
3628   \def\bbl@footnotetext#1#2#3{%
3629     \@ifnextchar[%
3630       {\bbl@footnotetext@o{#1}{#2}{#3}}%
3631       {\bbl@footnotetext@x{#1}{#2}{#3}}}
3632   \def\bbl@footnotetext@x#1#2#3#4{%
3633     \bgroup
3634     \select@language@x{\bbl@main@language}%
3635     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3636     \egroup}
3637   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3638     \bgroup

```

```

3639 \select@language@x{\bbl@main@language}%
3640 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3641 \egroup}
3642 \def\BabelFootnote#1#2#3#4{%
3643 \ifx\bbl@fn@footnote\@undefined
3644 \let\bbl@fn@footnote\footnote
3645 \fi
3646 \ifx\bbl@fn@footnotetext\@undefined
3647 \let\bbl@fn@footnotetext\footnotetext
3648 \fi
3649 \bbl@ifblank{#2}%
3650 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3651 \@namedef{\bbl@stripslash#1text}%
3652 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3653 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}{#3}{#4}}%
3654 \@namedef{\bbl@stripslash#1text}%
3655 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}}
3656 \fi
3657 <</Footnote changes>>

```

Now, the code.

```

3658 (*xetex)
3659 \def\BabelStringsDefault{unicode}
3660 \let\xebbl@stop\relax
3661 \AddBabelHook{xetex}{encodedcommands}{%
3662 \def\bbl@tempa{#1}%
3663 \ifx\bbl@tempa\@empty
3664 \XeTeXinputencoding"bytes"%
3665 \else
3666 \XeTeXinputencoding"#1"%
3667 \fi
3668 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3669 \AddBabelHook{xetex}{stopcommands}{%
3670 \xebbl@stop
3671 \let\xebbl@stop\relax}
3672 \def\bbl@intraspace#1 #2 #3\@@{%
3673 \bbl@csarg\gdef{\xeisp@bbl@cs{sbcpr@languagename}}%
3674 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3675 \def\bbl@intrapenalty#1\@@{%
3676 \bbl@csarg\gdef{\xeipn@bbl@cs{sbcpr@languagename}}%
3677 {\XeTeXlinebreakpenalty #1\relax}}
3678 \def\bbl@provide@intraspace{%
3679 \bbl@xin@{\bbl@cs{sbcpr@languagename}}{Thai,Lao,Khmr}%
3680 \ifin@ % sea (currently ckj not handled)
3681 \bbl@ifunset{bbl@intsp@languagename}{%
3682 {\expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else
3683 \ifx\bbl@KVP@intraspace\@nil
3684 \bbl@exp{%
3685 \bbl@intraspace\bbl@cs{intsp@languagename}\@@}%
3686 \fi
3687 \ifx\bbl@KVP@intrapenalty\@nil
3688 \bbl@intrapenalty0\@@
3689 \fi
3690 \fi
3691 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
3692 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
3693 \fi
3694 \ifx\bbl@KVP@intrapenalty\@nil\else
3695 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@

```

```

3696      \fi
3697      \ifx\bbl@ispace\undefined
3698      \AtBeginDocument{%
3699      \expandafter\bbl@add
3700      \csname selectfont \endcsname{\bbl@ispace}%
3701      \def\bbl@ispace{\bbl@cs{xeisp@\bbl@cs{sbcpr@\language}}}%
3702      \fi}%
3703  \fi}
3704 \AddBabelHook{xetex}{loadkernel}{%
3705 <<Restore Unicode catcodes before loading patterns>>}}
3706 \ifx\DisableBabelHook\undefined\endinput\fi
3707 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3708 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
3709 \DisableBabelHook{babel-fontspec}
3710 <<Font selection>>
3711 \input txtbabel.def
3712 </xetex>

```

## 14.2 Layout

*In progress.*

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

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the  $\TeX$  expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdfTeX and xetex.

```

3713 < *texxet >
3714 \providecommand\bbl@provide@intraspace{}
3715 \bbl@trace{Redefinitions for bidi layout}
3716 \def\bbl@sspre@caption{%
3717   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
3718 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
3719 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3720 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3721 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3722   \def\@hangfrom#1{%
3723     \setbox\@tempboxa\hbox{#1}%
3724     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3725     \noindent\box\@tempboxa}
3726 \def\raggedright{%
3727   \let\@centercr
3728   \bbl@startskip\z@skip
3729   \@rightskip\@flushglue
3730   \bbl@endskip\@rightskip
3731   \parindent\z@
3732   \parfillskip\bbl@startskip}
3733 \def\raggedleft{%
3734   \let\@centercr
3735   \bbl@startskip\@flushglue
3736   \bbl@endskip\z@skip
3737   \parindent\z@
3738   \parfillskip\bbl@endskip}
3739 \fi
3740 \IfBabelLayout{lists}
3741 {\bbl@sreplace\list

```

```

3742     {\totalleftmargin\leftmargin}{\totalleftmargin\bbl@listleftmargin}%
3743 \def\bbl@listleftmargin{%
3744   \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
3745 \ifcase\bbl@engine
3746   \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
3747   \def\p@enumiii{\p@enumii}\theenumii{}\%
3748 \fi
3749 \bbl@sreplace\@verbatim
3750   {\leftskip\totalleftmargin}%
3751   {\bbl@startskip\textwidth
3752     \advance\bbl@startskip-\linewidth}%
3753 \bbl@sreplace\@verbatim
3754   {\rightskip\z@skip}%
3755   {\bbl@endskip\z@skip}}%
3756 {}
3757 \IfBabelLayout{contents}
3758   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
3759   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
3760 {}
3761 \IfBabelLayout{columns}
3762   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
3763   \def\bbl@outputbox#1{%
3764     \hb@xt@\textwidth{%
3765       \hskip\columnwidth
3766       \hfil
3767       {\normalcolor\vrule \@width\columnseprule}%
3768       \hfil
3769       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3770       \hskip-\textwidth
3771       \hb@xt@\columnwidth{\box\@outputbox \hss}%
3772       \hskip\columnsep
3773       \hskip\columnwidth}}}%
3774   {}
3775 \IfBabelLayout{footnotes}
3776   {\BabelFootnote\footnote\language\{}}%
3777   {\BabelFootnote\localfootnote\language\{}}%
3778   {\BabelFootnote\mainfootnote\{}}%
3779   {}
3780 {}

```

Implicitly reverses sectioning labels in `bidibasic`, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

3781 \IfBabelLayout{counters}%
3782   {\let\bbl@latinarabic=\@arabic
3783   \def\@arabic#1{\bbl@sublrbbl@latinarabic#1}}%
3784   {\let\bbl@asciroman=\@roman
3785   \def\@roman#1{\bbl@sublrbbl@asciroman#1}}%
3786   {\let\bbl@asciiRoman=\@Roman
3787   \def\@Roman#1{\bbl@sublrbbl@asciiRoman#1}}%
3788 \end{texet}

```

### 14.3 LuaTeX

The new loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for ‘english’, so that it’s available without further intervention from the user. To avoid duplicating it, the following rule applies: if the “0th” language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won’t at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn’t happen very often – with luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn’t work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like `ctablestack`). For the moment, a dangerous approach is used – just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

```

3789 (*luatex)
3790 \ifx\AddBabelHook\undefined
3791 \bbl@trace{Read language.dat}
3792 \begingroup
3793   \toks@{}
3794   \count@ \z@ % 0=start, 1=0th, 2=normal
3795   \def\bbl@process@line#1#2 #3 #4 {%
3796     \ifx=#1%
3797       \bbl@process@synonym{#2}%
3798     \else
3799       \bbl@process@language{#1#2}{#3}{#4}%
3800     \fi
3801     \ignorespaces}
3802   \def\bbl@manylang{%
3803     \ifnum\bbl@last>\@ne
3804       \bbl@info{Non-standard hyphenation setup}%
3805     \fi
3806     \let\bbl@manylang\relax}
3807   \def\bbl@process@language#1#2#3{%
3808     \ifcase\count@
3809       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
3810     \or
3811       \count@\tw@
3812     \fi
3813     \ifnum\count@=\tw@
3814       \expandafter\addlanguage\csname l@#1\endcsname
3815       \language\allocationnumber
3816       \chardef\bbl@last\allocationnumber
3817       \bbl@manylang
3818     \let\bbl@elt\relax

```

```

3819 \xdef\bbl@languages{%
3820 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3821 \fi
3822 \the\toks@
3823 \toks@{}}
3824 \def\bbl@process@synonym@aux#1#2{%
3825 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3826 \let\bbl@elt\relax
3827 \xdef\bbl@languages{%
3828 \bbl@languages\bbl@elt{#1}{#2}{}}}%
3829 \def\bbl@process@synonym#1{%
3830 \ifcase\count@
3831 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3832 \or
3833 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3834 \else
3835 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3836 \fi}
3837 \ifx\bbl@languages@\undefined % Just a (sensible?) guess
3838 \chardef\l@english\z@
3839 \chardef\l@USenglish\z@
3840 \chardef\bbl@last\z@
3841 \global\@namedef{\bbl@hyphendata@0}{\hyphen.tex}{}
3842 \gdef\bbl@languages{%
3843 \bbl@elt{english}{0}{\hyphen.tex}{}%
3844 \bbl@elt{USenglish}{0}{}}%
3845 \else
3846 \global\let\bbl@languages@format\bbl@languages
3847 \def\bbl@elt#1#2#3#4{% Remove all except language 0
3848 \ifnum#2>\z@\else
3849 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3850 \fi}%
3851 \xdef\bbl@languages{\bbl@languages}%
3852 \fi
3853 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3854 \bbl@languages
3855 \openin1=language.dat
3856 \ifeof1
3857 \bbl@warning{I couldn't find language.dat. No additional\\%
3858 patterns loaded. Reported}%
3859 \else
3860 \loop
3861 \endlinechar\m@ne
3862 \read1 to \bbl@line
3863 \endlinechar\^^M
3864 \if T\ifeof1F\fi T\relax
3865 \ifx\bbl@line\empty\else
3866 \edef\bbl@line{\bbl@line\space\space\space}%
3867 \expandafter\bbl@process@line\bbl@line\relax
3868 \fi
3869 \repeat
3870 \fi
3871 \endgroup
3872 \bbl@trace{Macros for reading patterns files}
3873 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
3874 \ifx\babelcatcodetablenum\undefined
3875 \def\babelcatcodetablenum{5211}
3876 \fi
3877 \def\bbl@luapatterns#1#2{%

```

```

3878 \bbl@get@enc#1::\@@@
3879 \setbox\z@\hbox\bgroup
3880 \begingroup
3881 \ifx\catcodetable\undefined
3882 \let\savecatcodetable\luatexsavecatcodetable
3883 \let\initcatcodetable\luatexinitcatcodetable
3884 \let\catcodetable\luatexcatcodetable
3885 \fi
3886 \savecatcodetable\babelcatcodetablenum\relax
3887 \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3888 \catcodetable\numexpr\babelcatcodetablenum+1\relax
3889 \catcode`\#6 \catcode`\$3 \catcode`\&4 \catcode`\^7
3890 \catcode`\_8 \catcode`\{1 \catcode`\}=2 \catcode`\~13
3891 \catcode`\@11 \catcode`\^^I10 \catcode`\^^J12
3892 \catcode`\<12 \catcode`\>12 \catcode`\*12 \catcode`\.=12
3893 \catcode`\-=12 \catcode`\/=12 \catcode`\[12 \catcode`\]=12
3894 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
3895 \input #1\relax
3896 \catcodetable\babelcatcodetablenum\relax
3897 \endgroup
3898 \def\bbl@tempa{#2}%
3899 \ifx\bbl@tempa@empty\else
3900 \input #2\relax
3901 \fi
3902 \egroup}%
3903 \def\bbl@patterns@lua#1{%
3904 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3905 \csname l@#1\endcsname
3906 \edef\bbl@tempa{#1}%
3907 \else
3908 \csname l@#1:\f@encoding\endcsname
3909 \edef\bbl@tempa{#1:\f@encoding}%
3910 \fi\relax
3911 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3912 \@ifundefined{bbl@hyphendata@the\language}%
3913 {\def\bbl@elt##1##2##3##4{%
3914 \ifnum##2=\csname l@#1:\f@encoding\endcsname % #2=spanish, dutch:OT1...
3915 \def\bbl@tempb{##3}%
3916 \ifx\bbl@tempb@empty\else % if not a synonymous
3917 \def\bbl@tempc{##3}{##4}%
3918 \fi
3919 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3920 \fi}%
3921 \bbl@languages
3922 \@ifundefined{bbl@hyphendata@the\language}%
3923 {\bbl@info{No hyphenation patterns were set for\the\language}%
3924 language '\bbl@tempa'. Reported}}%
3925 {\expandafter\expandafter\expandafter\bbl@luapatterns
3926 \csname bbl@hyphendata@the\language\endcsname}}}%
3927 \endinput\fi
3928 \begingroup
3929 \catcode`\%=12
3930 \catcode`\'=12
3931 \catcode`\`=12
3932 \catcode`\:=12
3933 \directlua{
3934 Babel = Babel or {}
3935 function Babel.bytes(line)
3936 return line:gsub(".",

```



```

3937     function (chr) return unicode.utf8.char(string.byte(chr)) end)
3938 end
3939 function Babel.begin_process_input()
3940     if luatexbase and luatexbase.add_to_callback then
3941         luatexbase.add_to_callback('process_input_buffer',
3942             Babel.bytes, 'Babel.bytes')
3943     else
3944         Babel.callback = callback.find('process_input_buffer')
3945         callback.register('process_input_buffer', Babel.bytes)
3946     end
3947 end
3948 function Babel.end_process_input ()
3949     if luatexbase and luatexbase.remove_from_callback then
3950         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3951     else
3952         callback.register('process_input_buffer', Babel.callback)
3953     end
3954 end
3955 function Babel.addpatterns(pp, lg)
3956     local lg = lang.new(lg)
3957     local pats = lang.patterns(lg) or ''
3958     lang.clear_patterns(lg)
3959     for p in pp:gmatch('[^%s]+') do
3960         ss = ''
3961         for i in string.utfcharacters(p:gsub('%d', '')) do
3962             ss = ss .. '%d?' .. i
3963         end
3964         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
3965         ss = ss:gsub('%.%%d%?$', '%%.')
3966         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3967         if n == 0 then
3968             tex.sprint(
3969                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
3970                 .. p .. [[{}]])
3971             pats = pats .. ' ' .. p
3972         else
3973             tex.sprint(
3974                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
3975                 .. p .. [[{}]])
3976         end
3977     end
3978     lang.patterns(lg, pats)
3979 end
3980 }
3981 \endgroup
3982 \ifx\newattribute\@undefined\else
3983     \newattribute\bbl@attr@locale
3984     \AddBabelHook{luatex}{beforeextras}{%
3985         \setattribute\bbl@attr@locale\localeid}
3986 \fi
3987 \def\BabelStringsDefault{unicode}
3988 \let\luabbl@stop\relax
3989 \AddBabelHook{luatex}{encodedcommands}{%
3990     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3991     \ifx\bbl@tempa\bbl@tempb\else
3992         \directlua{Babel.begin_process_input()}%
3993         \def\luabbl@stop{%
3994             \directlua{Babel.end_process_input()}}%
3995     \fi}%

```

```

3996 \AddBabelHook{luatex}{stopcommands}{%
3997   \luabb1@stop
3998   \let\luabb1@stop\relax}
3999 \AddBabelHook{luatex}{patterns}{%
4000   \@ifundefined{bbl@hyphendata@the\language}%
4001     {\def\bbl@elt##1##2##3##4{%
4002       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4003       \def\bbl@tempb{##3}%
4004       \ifx\bbl@tempb@empty\else % if not a synonymous
4005         \def\bbl@tempc{##3}{##4}}%
4006       \fi
4007       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4008     \fi}%
4009   \bbl@languages
4010   \@ifundefined{bbl@hyphendata@the\language}%
4011     {\bbl@info{No hyphenation patterns were set for\%
4012       language '#2'. Reported}}%
4013     {\expandafter\expandafter\expandafter\bbl@luapatterns
4014       \csname bbl@hyphendata@the\language\endcsname}}}%
4015   \@ifundefined{bbl@patterns@}{}%
4016   \begingroup
4017   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4018   \ifin@else
4019     \ifx\bbl@patterns@empty\else
4020       \directlua{ Babel.addpatterns(
4021         [[\bbl@patterns@]], \number\language) }%
4022     \fi
4023     \@ifundefined{bbl@patterns@#1}%
4024     \@empty
4025     {\directlua{ Babel.addpatterns(
4026       [[\space\csname bbl@patterns@#1\endcsname]],
4027       \number\language) }}%
4028     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4029   \fi
4030   \endgroup}}
4031 \AddBabelHook{luatex}{everylanguage}{%
4032   \def\process@language##1##2##3{%
4033     \def\process@line####1####2 ####3 ####4 {}}}
4034 \AddBabelHook{luatex}{loadpatterns}{%
4035   \input #1\relax
4036   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4037     {##1}}}%
4038 \AddBabelHook{luatex}{loadexceptions}{%
4039   \input #1\relax
4040   \def\bbl@tempb##1##2{##1}{##1}%
4041   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4042     {\expandafter\expandafter\expandafter\bbl@tempb
4043       \csname bbl@hyphendata@the\language\endcsname}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4044 \@onlypreamble\babelpatterns
4045 \AtEndOfPackage{%
4046   \newcommand\babelpatterns[2][\@empty]{%
4047     \ifx\bbl@patterns@relax
4048       \let\bbl@patterns@\@empty
4049     \fi

```

```

4050 \ifx\bbl@pttnlist\@empty\else
4051 \bbl@warning{%
4052 You must not intermingle \string\selectlanguage\space and\\%
4053 \string\babelpatterns\space or some patterns will not\\%
4054 be taken into account. Reported}%
4055 \fi
4056 \ifx\@empty#1%
4057 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4058 \else
4059 \edef\bbl@tempb{\zap@space#1 \@empty}%
4060 \bbl@for\bbl@tempa\bbl@tempb{%
4061 \bbl@fixname\bbl@tempa
4062 \bbl@iflanguage\bbl@tempa{%
4063 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4064 \@ifundefined{bbl@patterns@\bbl@tempa}%
4065 \@empty
4066 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4067 #2}}}%
4068 \fi}}

```

## 14.4 Southeast Asian scripts

*In progress.* Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4069 \directlua{
4070 Babel = Babel or {}
4071 Babel.linebreaking = Babel.linebreaking or {}
4072 Babel.linebreaking.before = {}
4073 Babel.linebreaking.after = {}
4074 Babel.locale = {} % Free to use, indexed with \localeid
4075 function Babel.linebreaking.add_before(func)
4076 tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4077 table.insert(Babel.linebreaking.before , func)
4078 end
4079 function Babel.linebreaking.add_after(func)
4080 tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4081 table.insert(Babel.linebreaking.after, func)
4082 end
4083 }
4084 \def\bbl@intraspace#1 #2 #3\@{%
4085 \directlua{
4086 Babel = Babel or {}
4087 Babel.intraspaces = Babel.intraspaces or {}
4088 Babel.intraspaces['\csname bbl@sbcpr@\languagename\endcsname'] = %
4089 {b = #1, p = #2, m = #3}
4090 Babel.locale_props[\the\localeid].intraspace = %
4091 {b = #1, p = #2, m = #3}
4092 }}
4093 \def\bbl@intrapenalty#1\@{%
4094 \directlua{
4095 Babel = Babel or {}
4096 Babel.intrapenalties = Babel.intrapenalties or {}
4097 Babel.intrapenalties['\csname bbl@sbcpr@\languagename\endcsname'] = #1
4098 Babel.locale_props[\the\localeid].intrapenalty = #1
4099 }}
4100 \begingroup

```

```

4101 \catcode`\%=12
4102 \catcode`\^=14
4103 \catcode`\'=12
4104 \catcode`\~=12
4105 \gdef\bbl@seaintraspace{^
4106 \let\bbl@seaintraspace\relax
4107 \directlua{
4108   Babel = Babel or {}
4109   Babel.sea_enabled = true
4110   Babel.sea_ranges = Babel.sea_ranges or {}
4111   function Babel.set_chrngs (script, chrng)
4112     local c = 0
4113     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
4114       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4115       c = c + 1
4116     end
4117   end
4118   function Babel.sea_disc_to_space (head)
4119     local sea_ranges = Babel.sea_ranges
4120     local last_char = nil
4121     local quad = 655360    ^^ 10 pt = 655360 = 10 * 65536
4122     for item in node.traverse(head) do
4123       local i = item.id
4124       if i == node.id'glyph' then
4125         last_char = item
4126       elseif i == 7 and item.subtype == 3 and last_char
4127         and last_char.char > 0x0C99 then
4128         quad = font.getfont(last_char.font).size
4129         for lg, rg in pairs(sea_ranges) do
4130           if last_char.char > rg[1] and last_char.char < rg[2] then
4131             lg = lg:sub(1, 4)
4132             local intraspace = Babel.intraspaces[lg]
4133             local intrapenalty = Babel.intrapenalties[lg]
4134             local n
4135             if intrapenalty ~= 0 then
4136               n = node.new(14, 0)    ^^ penalty
4137               n.penalty = intrapenalty
4138               node.insert_before(head, item, n)
4139             end
4140             n = node.new(12, 13)    ^^ (glue, spaceskip)
4141             node.setglue(n, intraspace.b * quad,
4142               intraspace.p * quad,
4143               intraspace.m * quad)
4144             node.insert_before(head, item, n)
4145             node.remove(head, item)
4146           end
4147         end
4148       end
4149     end
4150   end
4151 }^^
4152 \bbl@luahyphenate}
4153 \catcode`\%=14
4154 \gdef\bbl@cjkintraspacespace{%
4155 \let\bbl@cjkintraspacespace\relax
4156 \directlua{
4157   Babel = Babel or {}
4158   require'babel-data-cjk.lua'
4159   Babel.cjk_enabled = true

```

```

4160 function Babel.cjk_linebreak(head)
4161     local GLYPH = node.id'glyph'
4162     local last_char = nil
4163     local quad = 655360      % 10 pt = 655360 = 10 * 65536
4164     local last_class = nil
4165     local last_lang = nil
4166
4167     for item in node.traverse(head) do
4168         if item.id == GLYPH then
4169
4170             local lang = item.lang
4171
4172             local LOCALE = node.get_attribute(item,
4173                 luatexbase.registernumber'bbl@attr@locale')
4174             local props = Babel.locale_props[LOCALE]
4175
4176             class = Babel.cjk_class[item.char].c
4177
4178             if class == 'cp' then class = 'cl' end % )] as CL
4179             if class == 'id' then class = 'I' end
4180
4181             if class and last_class and Babel.cjk_breaks[last_class][class] then
4182                 br = Babel.cjk_breaks[last_class][class]
4183             else
4184                 br = 0
4185             end
4186
4187             if br == 1 and props.linebreak == 'c' and
4188                 lang ~= \the\l@nohyphenation\space and
4189                 last_lang ~= \the\l@nohyphenation then
4190                 local intrapenalty = props.intrapenalty
4191                 if intrapenalty ~= 0 then
4192                     local n = node.new(14, 0)      % penalty
4193                     n.penalty = intrapenalty
4194                     node.insert_before(head, item, n)
4195                 end
4196                 local intraspace = props.intraspace
4197                 local n = node.new(12, 13)          % (glue, spaceskip)
4198                 node.setglue(n, intraspace.b * quad,
4199                     intraspace.p * quad,
4200                     intraspace.m * quad)
4201                 node.insert_before(head, item, n)
4202             end
4203
4204             quad = font.getfont(item.font).size
4205             last_class = class
4206             last_lang = lang
4207         else % if penalty, glue or anything else
4208             last_class = nil
4209         end
4210     end
4211     lang.hyphenate(head)
4212 end
4213 }%
4214 \bbl@luahyphenate}
4215 \gdef\bbl@luahyphenate{%
4216     \let\bbl@luahyphenate\relax
4217     \directlua{
4218         luatexbase.add_to_callback('hyphenate',

```

```

4219 function (head, tail)
4220   if Babel.cjk_enabled then
4221     Babel.cjk_linebreak(head)
4222   end
4223   if Babel.linebreaking.before then
4224     for k, func in ipairs(Babel.linebreaking.before) do
4225       func(head)
4226     end
4227   end
4228   lang.hyphenate(head)
4229   if Babel.linebreaking.after then
4230     for k, func in ipairs(Babel.linebreaking.after) do
4231       func(head)
4232     end
4233   end
4234   if Babel.sea_enabled then
4235     Babel.sea_disc_to_space(head)
4236   end
4237 end,
4238 'Babel.hyphenate')
4239 }
4240 }
4241 \endgroup
4242 \def\bbbl@provide@intraspace{%
4243   \bbbl@ifunset{\bbbl@intsp@\languagename}{}%
4244   {\xexpandafter\ifx\csname \bbbl@intsp@\languagename\endcsname\@empty\else
4245     \bbbl@xin@\bbbl@cs{sbcpr@\languagename}}{Hant,Hans,Jpan,Kore,Kana}%
4246     \ifin@           % cjk
4247     \bbbl@cjk@intraspace
4248     \directlua{
4249       Babel = Babel or {}
4250       Babel.locale_props = Babel.locale_props or {}
4251       Babel.locale_props[\the\localeid].linebreak = 'c'
4252     }%
4253     \bbbl@exp{\bbbl@intraspace\bbbl@cs{intsp@\languagename}\@}%
4254     \ifx\bbbl@KVP@intrapenalty\@nil
4255       \bbbl@intrapenalty0\@
4256     \fi
4257   \else           % sea
4258     \bbbl@seaintraspace
4259     \bbbl@exp{\bbbl@intraspace\bbbl@cs{intsp@\languagename}\@}%
4260     \directlua{
4261       Babel = Babel or {}
4262       Babel.sea_ranges = Babel.sea_ranges or {}
4263       Babel.set_chranges('\bbbl@cs{sbcpr@\languagename}',
4264         '\bbbl@cs{chrng@\languagename}')
4265     }%
4266     \ifx\bbbl@KVP@intrapenalty\@nil
4267       \bbbl@intrapenalty0\@
4268     \fi
4269   \fi
4270 \fi
4271 \ifx\bbbl@KVP@intrapenalty\@nil\else
4272   \xexpandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@
4273 \fi}}

```

## 14.5 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.

```
4274 \AddBabelHook{luatex}{loadkernel}{%
4275 \<<Restore Unicode catcodes before loading patterns>>}
4276 \ifx\DisableBabelHook\undefined\endinput\fi
4277 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4278 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4279 \DisableBabelHook{babel-fontspec}
4280 \<<Font selection>>
```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```
4281 \newcommand\babelcharproperty[1]{%
4282   \count@=#1\relax
4283   \ifvmode
4284     \expandafter\bbl@chprop
4285   \else
4286     \bbl@error{\string\babelcharproperty\space can be used only in\%
4287               vertical mode (preamble or between paragraphs)}%
4288               {See the manual for futher info}%
4289   \fi}
4290 \newcommand\bbl@chprop[3][\the\count@]{%
4291   \@tempcnta=#1\relax
4292   \bbl@ifunset{\bbl@chprop@#2}%
4293   {\bbl@error{No property named '#2'. Allowed values are\%
4294             direction (bc), mirror (bmg), and linebreak (lb)}%
4295             {See the manual for futher info}}%
4296   {%
4297   \loop
4298     \@nameuse{\bbl@chprop@#2}{#3}%
4299     \ifnum\count@<\@tempcnta
4300       \advance\count@\@ne
4301     \repeat}
4302 \def\bbl@chprop@direction#1{%
4303   \directlua{
4304     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4305     Babel.characters[\the\count@]['d'] = '#1'
4306   }}
4307 \let\bbl@chprop@bc\bbl@chprop@direction
4308 \def\bbl@chprop@mirror#1{%
4309   \directlua{
4310     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4311     Babel.characters[\the\count@]['m'] = '\number#1'
4312   }}
4313 \let\bbl@chprop@bmg\bbl@chprop@mirror
4314 \def\bbl@chprop@linebreak#1{%
4315   \directlua{
4316     Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4317     Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4318   }}
```

```

4318 }}
4319 \let\bbl@chprop@lb\bbl@chprop@linebreak

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f.

```

4320 \begingroup
4321 \catcode`\#=12
4322 \catcode`\%=12
4323 \catcode`\&=14
4324 \directlua{
4325   function Babel.str_to_nodes(text, base)
4326     local n, head, last
4327     for s in string.utfvalues(text) do
4328       if base.id == 7 then
4329         base = base.replace
4330       end
4331       n = node.copy(base)
4332       n.char = s
4333       if not head then
4334         head = n
4335       else
4336         last.next = n
4337       end
4338       last = n
4339     end
4340     return head
4341   end
4342
4343   function Babel.fetch_word(head, funct)
4344     local word_string = ''
4345     local word_nodes = {}
4346     local lang
4347
4348     for item in node.traverse(head) do
4349
4350       if item.id == 29
4351         and not(item.char == 124) && ie, not |
4352         and not(item.char == 61) && ie, not =
4353         and (item.lang == lang or lang == nil) then
4354         lang = lang or item.lang
4355         word_string = word_string .. unicode.utf8.char(item.char)
4356         word_nodes[#word_nodes+1] = item
4357
4358       elseif item.id == 7 and item.subtype == 2 then
4359         word_string = word_string .. '='
4360         word_nodes[#word_nodes+1] = item
4361
4362       elseif item.id == 7 and item.subtype == 3 then
4363         word_string = word_string .. '|'
4364         word_nodes[#word_nodes+1] = item
4365
4366       elseif word_string == '' then
4367         && pass
4368
4369       else
4370         return word_string, word_nodes, item, lang
4371       end
4372     end
4373   end
4374

```



```

4375 function Babel.capture_func(key, cap)
4376     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
4377     ret = ret:gsub("%[%[%]%]%.%", '')
4378     ret = ret:gsub("%.%[%[%]%]%", '')
4379     return key .. "[[=function(m) return ]] .. ret .. [[ end]]
4380 end
4381
4382 Babel.linebreaking.replacements = {}
4383
4384 function Babel.post_hyphenate_replace(head)
4385     local u = unicode.utf8
4386     local lbk = Babel.linebreaking
4387     local word_head = head
4388
4389     while true do
4390         local w, wn, nw, lang = Babel.fetch_word(word_head)
4391         if not lang then return head end
4392
4393         if not lbk.replacements[lang] then
4394             break
4395         end
4396
4397         for k=1, #lbk.replacements[lang] do
4398             local r = lbk.replacements[lang][k].replace
4399             local p = lbk.replacements[lang][k].pattern
4400
4401             while true do
4402                 local matches = { u.match(w, p) }
4403                 if #matches < 2 then break end
4404
4405                 local first = table.remove(matches, 1)
4406                 local last = table.remove(matches, #matches)
4407
4408                 %% Fix offsets, from bytes to unicode
4409                 first = u.len(w:sub(1, first-1)) + 1
4410                 last = u.len(w:sub(1, last-1))
4411
4412                 local new %% used when inserting and removing nodes
4413                 local changed = 0
4414
4415                 %% This loop is somewhat dirty. To refactor.
4416                 for q = first, last do
4417                     local rep_i = r[q-first+1]
4418                     local char_node = wn[q]
4419                     local char_base = char_node
4420
4421                     if rep_i and rep_i.data then
4422                         char_base = wn[rep_i.data+first-1]
4423                     end
4424
4425                     if rep_i == nil then
4426                         rep_i = { string = function(m) return '' end }
4427                     end
4428
4429                     if rep_i and (rep_i.pre or rep_i.no or rep_i.post) then
4430                         changed = changed + 1
4431                         d = node.new(7, 0) %% (disc, discretionary)
4432                         local prepre = rep_i.pre and rep_i.pre(matches) or ''
4433                         d.pre = Babel.str_to_nodes(prepre, char_base)

```

```

4434         d.post = Babel.str_to_nodes(
4435             rep_i.post and rep_i.post(matches) or '', char_base)
4436         d.replace = Babel.str_to_nodes(
4437             rep_i.no and rep_i.no(matches) or '', char_base)
4438         d.attr = char_base.attr
4439         if prepre == '' then    %% TeXbook p96
4440             d.penalty = rep_i.penalty or tex.hyphenpenalty
4441         else
4442             d.penalty = rep_i.penalty or tex.exhyphenpenalty
4443         end
4444         head, new = node.insert_before(head, char_node, d)
4445         node.remove(head, char_node)
4446         if q == 1 then
4447             word_head = new
4448         end
4449     elseif rep_i and rep_i.string then
4450         changed = changed + 1
4451         local str = rep_i.string(matches)
4452         if str == '' then
4453             if q == 1 then
4454                 word_head = char_node.next
4455             end
4456             head, new = node.remove(head, char_node)
4457         elseif char_node.id == 29 and u.len(str) == 1 then
4458             char_node.char = string.utfvalue(str)
4459         else
4460             local n
4461             for s in string.utfvalues(str) do
4462                 if char_node.id == 7 then
4463                     log('Automatic hyphens cannot be replaced, just removed.')
4464                 else
4465                     n = node.copy(char_base)
4466                 end
4467                 n.char = s
4468                 if q == 1 then
4469                     head, new = node.insert_before(head, char_node, n)
4470                     word_head = new
4471                 else
4472                     node.insert_before(head, char_node, n)
4473                 end
4474             end
4475         end
4476         node.remove(head, char_node)
4477     end    %% string length
4478 end    %% if char and char.string
4479 end    %% for char in match
4480 if changed > 20 then
4481     texio.write('Too many changes. Ignoring the rest.')
4482 elseif changed > 0 then
4483     w, wn, nw = Babel.fetch_word(word_head)
4484 end
4485
4486 end    %% for match
4487 end    %% for patterns
4488 word_head = nw
4489 end    %% for words
4490 return head
4491 end
4492 }

```

```

4493 \catcode`\#=6
4494 \gdef\babelposthyphenation#1#2#3{&%
4495 \begingroup
4496 \def\babeltempa{\bbl@add@list\babeltempb}&%
4497 \let\babeltempb\@empty
4498 \bbl@foreach{#3}{&%
4499 \bbl@ifsamestring{##1}{remove}&%
4500 {\bbl@add@list\babeltempb{nil}}&%
4501 {\directlua{
4502 local rep = [[#1]]
4503 rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
4504 rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
4505 rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
4506 rep = rep:gsub( '(string)%s*=%s*([^\s,]*)', Babel.capture_func)
4507 tex.print([[\\string\babeltempa{}}] .. rep .. [[}}]])
4508 }}&%
4509 \directlua{
4510 local lbk = Babel.linebreaking
4511 local u = unicode.utf8
4512 local patt = string.gsub([[#2]], '%s', '')
4513 if not u.find(patt, '()', nil, true) then
4514 patt = '()' .. patt .. '()'
4515 end
4516 patt = u.gsub(patt, '{(.)}',
4517 function (n)
4518 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
4519 end)
4520 lbk.replacements[\\the\\csname l@#1\\endcsname] =
4521 lbk.replacements[\\the\\csname l@#1\\endcsname] or {}
4522 table.insert(lbk.replacements[\\the\\csname l@#1\\endcsname],
4523 { pattern = patt, replace = { \babeltempb } })
4524 }&%
4525 \endgroup}
4526 \endgroup
4527 %
4528 % \subsection{Layout}
4529 %
4530 % \textbf{Work in progress}.
4531 %
4532 % Unlike \xetex{}, \luatex{} requires only minimal changes for
4533 % right-to-left layouts, particularly in monolingual documents (the
4534 % engine itself reverses boxes -- including column order or headings
4535 % --, margins, etc.) with |bidi=basic|, without having to patch
4536 % almost any macro where text direction is relevant.
4537 %
4538 % |\@hangfrom| is useful in many contexts and it is redefined always
4539 % with the |layout| option.
4540 %
4541 % There are, however, a number of issues when the text direction is not
4542 % the same as the box direction (as set by |\bodydir|), and when
4543 % |\parbox| and |\hangindent| are involved. Fortunately, latest
4544 % releases of \luatex{} simplify a lot the solution with |\shapemode|.
4545 %
4546 % With the issue \#15 I realized commands are best patched, instead of
4547 % redefined. With a few lines, a modification could be applied to
4548 % several classes and packages. Now, |tabular| seems to work (at least
4549 % in simple cases) with \textsf{array}, \textsf{tabularx},
4550 % \textsf{hhline}, \textsf{colortbl}, \textsf{longtable},
4551 % \textsf{booktabs}, etc. However, \textsf{dcolumn} still fails.

```

```

4552 %
4553 % \changes{babel-3.18}{2018/02/14}{\cs{bbl@nextfake}, similar to the old
4554 %   \cs{nextfakemath} in Omega}
4555 % \changes{babel-3.19}{2018/04/23}{Patch \cs{eqnnum}, somewhat ad hoc. To be
4556 %   improved.}
4557 % \changes{babel-3.21}{2018/05/09}{The ad hoc \cs{eqnnum} was buggy. Fixed.}
4558 % \changes{babel-3.31}{2019/05/04}{\cs{tabular} and \cs{list} are patched,
4559 %   instead of redefined.}
4560 %
4561 %   \begin{macrocode}
4562 \bbl@trace{Redefinitions for bidi layout}
4563 \ifx\@eqnnum\undefined\else
4564   \ifx\bbl@attr@dir\undefined\else
4565     \edef\@eqnnum{%
4566       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
4567       \unexpanded\expandafter{\@eqnnum}}
4568   \fi
4569 \fi
4570 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
4571 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4572   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
4573     \bbl@exp{%
4574       \mathdir\the\bodydir
4575       #1%           Once entered in math, set boxes to restore values
4576       \<ifmmode>%
4577       \everyvbox{%
4578         \the\everyvbox
4579         \bodydir\the\bodydir
4580         \mathdir\the\mathdir
4581         \everyhbox{\the\everyhbox}%
4582         \everyvbox{\the\everyvbox}}%
4583       \everyhbox{%
4584         \the\everyhbox
4585         \bodydir\the\bodydir
4586         \mathdir\the\mathdir
4587         \everyhbox{\the\everyhbox}%
4588         \everyvbox{\the\everyvbox}}%
4589       \<fi>}}%
4590   \def\@hangfrom#1{%
4591     \setbox\@tempboxa\hbox{#1}%
4592     \hangindent\wd\@tempboxa
4593     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4594       \shapemode\@ne
4595     \fi
4596     \noindent\box\@tempboxa}
4597 \fi
4598 \IfBabelLayout{tabular}
4599   {\let\bbl@OL@tabular\@tabular
4600    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
4601    \let\bbl@NL@tabular\@tabular
4602    \AtBeginDocument{%
4603      \ifx\bbl@NL@tabular\@tabular\else
4604        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
4605        \let\bbl@NL@tabular\@tabular
4606      \fi}}
4607   {}
4608 \IfBabelLayout{lists}
4609   {\let\bbl@OL@list\list
4610    \bbl@replace\list{\parshape}{\bbl@listparshape}%

```

```

4611 \let\bbl@NL@list\list
4612 \def\bbl@listparshape#1#2#3{%
4613   \parshape #1 #2 #3 %
4614   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4615     \shapemode\tw@
4616   \fi}}
4617 {}
4618 \IfBabelLayout{graphics}
4619 {\let\bbl@pictresetdir\relax
4620 \def\bbl@pictsetdir{%
4621   \ifcase\bbl@thetextdir
4622     \let\bbl@pictresetdir\relax
4623   \else
4624     \textdir TLT\relax
4625   \def\bbl@pictresetdir{\textdir TRT\relax}%
4626   \fi}%
4627 \let\bbl@OL@@picture\@picture
4628 \let\bbl@OL@put\put
4629 \bbl@sreplace\@picture{\hskip-}\bbl@pictsetdir\hskip-}%
4630 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
4631   \@killglue
4632   \raise#2\unitlength
4633   \hb@xt@#3\kern#1\unitlength{\bbl@pictresetdir#3}\hss}}%
4634 \AtBeginDocument
4635 {\ifx\tikz@atbegin@node\undefined\else
4636   \let\bbl@OL@pgfpicture\pgfpicture
4637   \bbl@sreplace\pgfpicture{\pgfpicturetrue}{\bbl@pictsetdir\pgfpicturetrue}%
4638   \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir}%
4639   \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
4640   \fi}}
4641 {}

```

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.

```

4642 \IfBabelLayout{counters}%
4643 {\let\bbl@OL@@textsuperscript\textsuperscript
4644 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
4645 \let\bbl@latinarabic=\@arabic
4646 \let\bbl@OL@@arabic\@arabic
4647 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4648 \@ifpackagewith{babel}{bidi=default}%
4649 {\let\bbl@asciroman=\@roman
4650 \let\bbl@OL@@roman\@roman
4651 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4652 \let\bbl@asciiRoman=\@Roman
4653 \let\bbl@OL@@roman\@Roman
4654 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
4655 \let\bbl@OL@labelenumii\labelenumii
4656 \def\labelenumii{}\theenumii}%
4657 \let\bbl@OL@p@enumiii\p@enumiii
4658 \def\p@enumiii{\p@enumii}\theenumii{}\{}\{}\}
4659 <<Footnote changes>>
4660 \IfBabelLayout{footnotes}%
4661 {\let\bbl@OL@footnote\footnote
4662 \BabelFootnote\footnote\language\language{}{}%
4663 \BabelFootnote\localfootnote\language\language{}{}%
4664 \BabelFootnote\mainfootnote{}\{}\{}\}
4665 {}

```

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.

```

4666 \IfBabelLayout{extras}%
4667   {\let\bbl@OL@underline\underline
4668    \bbl@sreplace\underline{$\@@underline}\bbl@nextfake$\@@underline}%
4669   \let\bbl@OL@LaTeX2e\LaTeX2e
4670   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
4671     \if b\expandafter\@car\@series\@nil\boldmath\fi
4672     \babelsublr{%
4673       \LaTeX\kern.15em\bbl@nextfake$_{\textstyle\varepsilon}$}}}
4674   {}
4675 \</luatex>

```

## 14.6 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.

```

4676 (*basic-r)
4677 Babel = Babel or {}
4678
4679 Babel.bidi_enabled = true
4680
4681 require('babel-data-bidi.lua')
4682
4683 local characters = Babel.characters
4684 local ranges = Babel.ranges
4685
4686 local DIR = node.id("dir")
4687
4688 local function dir_mark(head, from, to, outer)
4689   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
4690   local d = node.new(DIR)
4691   d.dir = '+' .. dir
4692   node.insert_before(head, from, d)
4693   d = node.new(DIR)
4694   d.dir = '-' .. dir
4695   node.insert_after(head, to, d)
4696 end
4697
4698 function Babel.bidi(head, ispar)
4699   local first_n, last_n          -- first and last char with nums
4700   local last_es                  -- an auxiliary 'last' used with nums
4701   local first_d, last_d          -- first and last char in L/R block
4702   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):

```

4703   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
4704   local strong_lr = (strong == 'l') and 'l' or 'r'
4705   local outer = strong
4706
4707   local new_dir = false
4708   local first_dir = false
4709   local inmath = false
4710
4711   local last_lr
4712
4713   local type_n = ''
4714
4715   for item in node.traverse(head) do
4716
4717     -- three cases: glyph, dir, otherwise
4718     if item.id == node.id'glyph'
4719       or (item.id == 7 and item.subtype == 2) then
4720
4721       local itemchar
4722       if item.id == 7 and item.subtype == 2 then
4723         itemchar = item.replace.char
4724       else
4725         itemchar = item.char
4726       end
4727       local chardata = characters[itemchar]
4728       dir = chardata and chardata.d or nil
4729       if not dir then
4730         for nn, et in ipairs(ranges) do

```

```

4731         if itemchar < et[1] then
4732             break
4733         elseif itemchar <= et[2] then
4734             dir = et[3]
4735             break
4736         end
4737     end
4738 end
4739 dir = dir or 'l'
4740 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).

```

4741     if new_dir then
4742         attr_dir = 0
4743         for at in node.traverse(item.attr) do
4744             if at.number == luatexbase.registernumber'bbl@attr@dir' then
4745                 attr_dir = at.value % 3
4746             end
4747         end
4748         if attr_dir == 1 then
4749             strong = 'r'
4750         elseif attr_dir == 2 then
4751             strong = 'al'
4752         else
4753             strong = 'l'
4754         end
4755         strong_lr = (strong == 'l') and 'l' or 'r'
4756         outer = strong_lr
4757         new_dir = false
4758     end
4759
4760     if dir == 'nsm' then dir = strong end -- W1

```

**Numbers.** The dual <al>/<r> system for R is somewhat cumbersome.

```

4761     dir_real = dir -- We need dir_real to set strong below
4762     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:

```

4763     if strong == 'al' then
4764         if dir == 'en' then dir = 'an' end -- W2
4765         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
4766         strong_lr = 'r' -- W3
4767     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

4768     elseif item.id == node.id'dir' and not inmath then
4769         new_dir = true
4770         dir = nil
4771     elseif item.id == node.id'math' then
4772         inmath = (item.subtype == 0)
4773     else
4774         dir = nil -- Not a char
4775     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>.

```

4776   if dir == 'en' or dir == 'an' or dir == 'et' then
4777       if dir ~= 'et' then
4778           type_n = dir
4779       end
4780       first_n = first_n or item
4781       last_n = last_es or item
4782       last_es = nil
4783   elseif dir == 'es' and last_n then -- W3+W6
4784       last_es = item
4785   elseif dir == 'cs' then             -- it's right - do nothing
4786   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
4787       if strong_lr == 'r' and type_n ~= '' then
4788           dir_mark(head, first_n, last_n, 'r')
4789       elseif strong_lr == 'l' and first_d and type_n == 'an' then
4790           dir_mark(head, first_n, last_n, 'r')
4791           dir_mark(head, first_d, last_d, outer)
4792           first_d, last_d = nil, nil
4793       elseif strong_lr == 'l' and type_n ~= '' then
4794           last_d = last_n
4795       end
4796       type_n = ''
4797       first_n, last_n = nil, nil
4798   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:

```

4799   if dir == 'l' or dir == 'r' then
4800       if dir ~= outer then
4801           first_d = first_d or item
4802           last_d = item
4803       elseif first_d and dir ~= strong_lr then
4804           dir_mark(head, first_d, last_d, outer)
4805           first_d, last_d = nil, nil
4806       end
4807   end

```

**Mirroring.** Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last\_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

4808   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
4809       item.char = characters[item.char] and
4810           characters[item.char].m or item.char
4811   elseif (dir or new_dir) and last_lr ~= item then
4812       local mir = outer .. strong_lr .. (dir or outer)
4813       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
4814           for ch in node.traverse(node.next(last_lr)) do
4815               if ch == item then break end

```

```

4816         if ch.id == node.id'glyph' and characters[ch.char] then
4817             ch.char = characters[ch.char].m or ch.char
4818         end
4819     end
4820 end
4821 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).

```

4822     if dir == 'l' or dir == 'r' then
4823         last_lr = item
4824         strong = dir_real          -- Don't search back - best save now
4825         strong_lr = (strong == 'l') and 'l' or 'r'
4826     elseif new_dir then
4827         last_lr = nil
4828     end
4829 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

4830 if last_lr and outer == 'r' then
4831     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
4832         if characters[ch.char] then
4833             ch.char = characters[ch.char].m or ch.char
4834         end
4835     end
4836 end
4837 if first_n then
4838     dir_mark(head, first_n, last_n, outer)
4839 end
4840 if first_d then
4841     dir_mark(head, first_d, last_d, outer)
4842 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

4843 return node.prev(head) or head
4844 end
4845 </basic-r>

```

And here the Lua code for bidi=basic:

```

4846 (*basic)
4847 Babel = Babel or {}
4848
4849 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
4850
4851 Babel.fontmap = Babel.fontmap or {}
4852 Babel.fontmap[0] = {}          -- l
4853 Babel.fontmap[1] = {}          -- r
4854 Babel.fontmap[2] = {}          -- al/an
4855
4856 Babel.bidi_enabled = true
4857 Babel.mirroring_enabled = true
4858
4859 -- Temporary:
4860
4861 if harf then
4862     Babel.mirroring_enabled = false
4863 end
4864

```

```

4865 require('babel-data-bidi.lua')
4866
4867 local characters = Babel.characters
4868 local ranges = Babel.ranges
4869
4870 local DIR = node.id('dir')
4871 local GLYPH = node.id('glyph')
4872
4873 local function insert_implicit(head, state, outer)
4874   local new_state = state
4875   if state.sim and state.eim and state.sim ~= state.eim then
4876     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
4877     local d = node.new(DIR)
4878     d.dir = '+' .. dir
4879     node.insert_before(head, state.sim, d)
4880     local d = node.new(DIR)
4881     d.dir = '-' .. dir
4882     node.insert_after(head, state.eim, d)
4883   end
4884   new_state.sim, new_state.eim = nil, nil
4885   return head, new_state
4886 end
4887
4888 local function insert_numeric(head, state)
4889   local new
4890   local new_state = state
4891   if state.san and state.ean and state.san ~= state.ean then
4892     local d = node.new(DIR)
4893     d.dir = '+TLT'
4894     _, new = node.insert_before(head, state.san, d)
4895     if state.san == state.sim then state.sim = new end
4896     local d = node.new(DIR)
4897     d.dir = '-TLT'
4898     _, new = node.insert_after(head, state.ean, d)
4899     if state.ean == state.eim then state.eim = new end
4900   end
4901   new_state.san, new_state.ean = nil, nil
4902   return head, new_state
4903 end
4904
4905 -- TODO - \hbox with an explicit dir can lead to wrong results
4906 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
4907 -- was s made to improve the situation, but the problem is the 3-dir
4908 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
4909 -- well.
4910
4911 function Babel.bidi(head, ispar, hdir)
4912   local d -- d is used mainly for computations in a loop
4913   local prev_d = ''
4914   local new_d = false
4915
4916   local nodes = {}
4917   local outer_first = nil
4918   local inmath = false
4919
4920   local glue_d = nil
4921   local glue_i = nil
4922
4923   local has_en = false

```

```

4924 local first_et = nil
4925
4926 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
4927
4928 local save_outer
4929 local temp = node.get_attribute(head, ATDIR)
4930 if temp then
4931     temp = temp % 3
4932     save_outer = (temp == 0 and 'l') or
4933                 (temp == 1 and 'r') or
4934                 (temp == 2 and 'al')
4935 elseif ispar then -- Or error? Shouldn't happen
4936     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
4937 else -- Or error? Shouldn't happen
4938     save_outer = ('TRT' == hdir) and 'r' or 'l'
4939 end
4940 -- when the callback is called, we are just _after_ the box,
4941 -- and the textdir is that of the surrounding text
4942 -- if not ispar and hdir ~= tex.textdir then
4943 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
4944 -- end
4945 local outer = save_outer
4946 local last = outer
4947 -- 'al' is only taken into account in the first, current loop
4948 if save_outer == 'al' then save_outer = 'r' end
4949
4950 local fontmap = Babel.fontmap
4951
4952 for item in node.traverse(head) do
4953
4954     -- In what follows, #node is the last (previous) node, because the
4955     -- current one is not added until we start processing the neutrals.
4956
4957     -- three cases: glyph, dir, otherwise
4958     if item.id == GLYPH
4959         or (item.id == 7 and item.subtype == 2) then
4960
4961         local d_font = nil
4962         local item_r
4963         if item.id == 7 and item.subtype == 2 then
4964             item_r = item.replace -- automatic discs have just 1 glyph
4965         else
4966             item_r = item
4967         end
4968         local chardata = characters[item_r.char]
4969         d = chardata and chardata.d or nil
4970         if not d or d == 'nsm' then
4971             for nn, et in ipairs(ranges) do
4972                 if item_r.char < et[1] then
4973                     break
4974                 elseif item_r.char <= et[2] then
4975                     if not d then d = et[3]
4976                     elseif d == 'nsm' then d_font = et[3]
4977                 end
4978                 break
4979             end
4980         end
4981         end
4982         d = d or 'l'

```

```

4983
4984 -- A short 'pause' in bidi for mapfont
4985 d_font = d_font or d
4986 d_font = (d_font == 'l' and 0) or
4987           (d_font == 'nsm' and 0) or
4988           (d_font == 'r' and 1) or
4989           (d_font == 'al' and 2) or
4990           (d_font == 'an' and 2) or nil
4991 if d_font and fontmap and fontmap[d_font][item_r.font] then
4992   item_r.font = fontmap[d_font][item_r.font]
4993 end
4994
4995 if new_d then
4996   table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4997   if inmath then
4998     attr_d = 0
4999   else
5000     attr_d = node.get_attribute(item, ATDIR)
5001     attr_d = attr_d % 3
5002   end
5003   if attr_d == 1 then
5004     outer_first = 'r'
5005     last = 'r'
5006   elseif attr_d == 2 then
5007     outer_first = 'r'
5008     last = 'al'
5009   else
5010     outer_first = 'l'
5011     last = 'l'
5012   end
5013   outer = last
5014   has_en = false
5015   first_et = nil
5016   new_d = false
5017 end
5018
5019 if glue_d then
5020   if (d == 'l' and 'l' or 'r') ~= glue_d then
5021     table.insert(nodes, {glue_i, 'on', nil})
5022   end
5023   glue_d = nil
5024   glue_i = nil
5025 end
5026
5027 elseif item.id == DIR then
5028   d = nil
5029   new_d = true
5030
5031 elseif item.id == node.id'glue' and item.subtype == 13 then
5032   glue_d = d
5033   glue_i = item
5034   d = nil
5035
5036 elseif item.id == node.id'math' then
5037   inmath = (item.subtype == 0)
5038
5039 else
5040   d = nil
5041 end

```

```

5042
5043 -- AL <= EN/ET/ES      -- W2 + W3 + W6
5044 if last == 'al' and d == 'en' then
5045     d = 'an'            -- W3
5046 elseif last == 'al' and (d == 'et' or d == 'es') then
5047     d = 'on'            -- W6
5048 end
5049
5050 -- EN + CS/ES + EN      -- W4
5051 if d == 'en' and #nodes >= 2 then
5052     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5053         and nodes[#nodes-1][2] == 'en' then
5054         nodes[#nodes][2] = 'en'
5055     end
5056 end
5057
5058 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
5059 if d == 'an' and #nodes >= 2 then
5060     if (nodes[#nodes][2] == 'cs')
5061         and nodes[#nodes-1][2] == 'an' then
5062         nodes[#nodes][2] = 'an'
5063     end
5064 end
5065
5066 -- ET/EN                -- W5 + W7->l / W6->on
5067 if d == 'et' then
5068     first_et = first_et or (#nodes + 1)
5069 elseif d == 'en' then
5070     has_en = true
5071     first_et = first_et or (#nodes + 1)
5072 elseif first_et then    -- d may be nil here !
5073     if has_en then
5074         if last == 'l' then
5075             temp = 'l'    -- W7
5076         else
5077             temp = 'en'   -- W5
5078         end
5079     else
5080         temp = 'on'       -- W6
5081     end
5082     for e = first_et, #nodes do
5083         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5084     end
5085     first_et = nil
5086     has_en = false
5087 end
5088
5089 if d then
5090     if d == 'al' then
5091         d = 'r'
5092         last = 'al'
5093     elseif d == 'l' or d == 'r' then
5094         last = d
5095     end
5096     prev_d = d
5097     table.insert(nodes, {item, d, outer_first})
5098 end
5099
5100 outer_first = nil

```

```

5101
5102 end
5103
5104 -- TODO -- repeated here in case EN/ET is the last node. Find a
5105 -- better way of doing things:
5106 if first_et then -- dir may be nil here !
5107   if has_en then
5108     if last == 'l' then
5109       temp = 'l' -- W7
5110     else
5111       temp = 'en' -- W5
5112     end
5113   else
5114     temp = 'on' -- W6
5115   end
5116   for e = first_et, #nodes do
5117     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5118   end
5119 end
5120
5121 -- dummy node, to close things
5122 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5123
5124 ----- NEUTRAL -----
5125
5126 outer = save_outer
5127 last = outer
5128
5129 local first_on = nil
5130
5131 for q = 1, #nodes do
5132   local item
5133
5134   local outer_first = nodes[q][3]
5135   outer = outer_first or outer
5136   last = outer_first or last
5137
5138   local d = nodes[q][2]
5139   if d == 'an' or d == 'en' then d = 'r' end
5140   if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5141
5142   if d == 'on' then
5143     first_on = first_on or q
5144   elseif first_on then
5145     if last == d then
5146       temp = d
5147     else
5148       temp = outer
5149     end
5150     for r = first_on, q - 1 do
5151       nodes[r][2] = temp
5152       item = nodes[r][1] -- MIRRORING
5153       if Babel.mirroring_enabled and item.id == GLYPH
5154         and temp == 'r' and characters[item.char] then
5155         item.char = characters[item.char].m or item.char
5156       end
5157     end
5158     first_on = nil
5159   end

```

```

5160
5161     if d == 'r' or d == 'l' then last = d end
5162 end
5163
5164 ----- IMPLICIT, REORDER -----
5165
5166 outer = save_outer
5167 last = outer
5168
5169 local state = {}
5170 state.has_r = false
5171
5172 for q = 1, #nodes do
5173     local item = nodes[q][1]
5174
5175     outer = nodes[q][3] or outer
5176
5177     local d = nodes[q][2]
5178
5179     if d == 'nsm' then d = last end          -- W1
5180     if d == 'en' then d = 'an' end
5181     local isdir = (d == 'r' or d == 'l')
5182
5183     if outer == 'l' and d == 'an' then
5184         state.san = state.san or item
5185         state.ean = item
5186     elseif state.san then
5187         head, state = insert_numeric(head, state)
5188     end
5189
5190     if outer == 'l' then
5191         if d == 'an' or d == 'r' then      -- im -> implicit
5192             if d == 'r' then state.has_r = true end
5193             state.sim = state.sim or item
5194             state.eim = item
5195         elseif d == 'l' and state.sim and state.has_r then
5196             head, state = insert_implicit(head, state, outer)
5197         elseif d == 'l' then
5198             state.sim, state.eim, state.has_r = nil, nil, false
5199         end
5200     else
5201         if d == 'an' or d == 'l' then
5202             if nodes[q][3] then -- nil except after an explicit dir
5203                 state.sim = item -- so we move sim 'inside' the group
5204             else
5205                 state.sim = state.sim or item
5206             end
5207             state.eim = item
5208         elseif d == 'r' and state.sim then
5209             head, state = insert_implicit(head, state, outer)
5210         elseif d == 'r' then
5211             state.sim, state.eim = nil, nil
5212         end
5213     end
5214 end
5215
5216 if isdir then
5217     last = d          -- Don't search back - best save now
5218 elseif d == 'on' and state.san then

```



```

5219     state.san = state.san or item
5220     state.ean = item
5221   end
5222
5223 end
5224
5225 return node.prev(head) or head
5226 end
5227  $\langle$ /basic $\rangle$ 

```

## 15 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.

## 16 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.

```

5228  $\langle$ *nil $\rangle$ 
5229 \ProvidesLanguage{nil}[ $\langle$ date $\rangle$ ][ $\langle$ version $\rangle$ ] Nil language]
5230 \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.

```

5231 \ifx\l@nil\undefined
5232   \newlanguage\l@nil
5233   \@namedef{bbl@hyphendata@the\l@nil}{\{\}\{\}}% Remove warning
5234   \let\bbl@elt\relax
5235   \edef\bbl@languages{% Add it to the list of languages
5236     \bbl@languages\bbl@elt{nil}{the\l@nil}\{\}\{\}}
5237 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

5238 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```

\captionnil
\datenil
5239 \let\captionnil\@empty
5240 \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.

```
5241 \ldf@finish{nil}
5242 </nil>
```

## 17 Support for Plain T<sub>E</sub>X (plain.def)

### 17.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T<sub>E</sub>X-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
5243 <(*bplain | blplain)
5244 \catcode`\{=1 % left brace is begin-group character
5245 \catcode`\}=2 % right brace is end-group character
5246 \catcode`\#=6 % hash mark is macro parameter character
```

Now let’s see if a file called `hyphen.cfg` can be found somewhere on T<sub>E</sub>X’s input path by trying to open it for reading...

```
5247 \openin 0 hyphen.cfg
```

If the file wasn’t found the following test turns out true.

```
5248 \ifeof0
5249 \else
```

When `hyphen.cfg` could be opened we make sure that *it* will be read instead of the file `hyphen.tex` which should (according to Don Knuth’s ruling) contain the american English hyphenation patterns and nothing else.

We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
5250 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
5251 \def\input #1 {%
5252   \let\input\input
5253   \input hyphen.cfg
```

Once that’s done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
5254   \let\input\undefined
5255 }
5256 \fi
5257 </bplain | blplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
5258 \bplain\la plain.tex
5259 \bplain\la lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
5260 \bplain\def\fmtname{babel-plain}
5261 \bplain\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

## 17.2 Emulating some $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```
5262 \*plain\
5263 \def\@empty{}
5264 \def\loadlocalcfg#1{%
5265   \openin0#1.cfg
5266   \ifeof0
5267     \closein0
5268   \else
5269     \closein0
5270     {\immediate\write16{*****}%
5271      \immediate\write16{* Local config file #1.cfg used}%
5272      \immediate\write16{*}%
5273     }
5274     \input #1.cfg\relax
5275   \fi
5276   \@endoflfd}
```

## 17.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```
5277 \long\def\@firstofone#1{#1}
5278 \long\def\@firstoftwo#1#2{#1}
5279 \long\def\@secondoftwo#1#2{#2}
5280 \def\@nnil{\@nil}
5281 \def\@gobbletwo#1#2{}
5282 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
5283 \def\@star@or@long#1{%
5284   \@ifstar
5285   {\let\l@ngrel@x\relax#1}%
5286   {\let\l@ngrel@x\long#1}}
5287 \let\l@ngrel@x\relax
5288 \def\@car#1#2\@nil{#1}
5289 \def\@cdr#1#2\@nil{#2}
5290 \let\@typeset@protect\relax
5291 \let\protected@edef\edef
5292 \long\def\@gobble#1{}
5293 \edef\@backslashchar{\expandafter\@gobble\string\}
5294 \def\strip@prefix#1>{}
5295 \def\g@addto@macro#1#2{%
5296   \toks@\expandafter{#1#2}%
5297   \xdef#1{\the\toks@}}
5298 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
```

```

5299 \def\@nameuse#1{\csname #1\endcsname}
5300 \def\@ifundefined#1{%
5301   \expandafter\ifx\csname#1\endcsname\relax
5302     \expandafter\@firstoftwo
5303   \else
5304     \expandafter\@secondoftwo
5305   \fi}
5306 \def\@expandtwoargs#1#2#3{%
5307   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
5308 \def\zap@space#1 #2{%
5309   #1%
5310   \ifx#2\@empty\else\expandafter\zap@space\fi
5311   #2}

```

$\text{\LaTeX 2}_\epsilon$  has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

5312 \ifx\@preamblecmds\@undefined
5313   \def\@preamblecmds{}
5314 \fi
5315 \def\@onlypreamble#1{%
5316   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
5317     \@preamblecmds\do#1}}
5318 \@onlypreamble\@onlypreamble

```

Mimick  $\text{\LaTeX}$ 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

5319 \def\begin{document}{%
5320   \@begin{document}hook
5321   \global\let\@begin{document}hook\@undefined
5322   \def\do##1{\global\let##1\@undefined}%
5323   \@preamblecmds
5324   \global\let\do\noexpand}

5325 \ifx\@begin{document}hook\@undefined
5326   \def\@begin{document}hook{}
5327 \fi
5328 \@onlypreamble\@begin{document}hook
5329 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimick  $\text{\LaTeX}$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

5330 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
5331 \@onlypreamble\AtEndOfPackage
5332 \def\@endoflfd{}
5333 \@onlypreamble\@endoflfd
5334 \let\bbl@afterlang\@empty
5335 \chardef\bbl@opt@hyphenmap\z@

```

$\text{\LaTeX}$  needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

5336 \ifx\if@filesw\@undefined
5337   \expandafter\let\csname if@filesw\endcsname
5338     \csname iffalse\endcsname
5339 \fi

```

Mimick  $\text{\LaTeX}$ 's commands to define control sequences.

```

5340 \def\newcommand{\@star@or@long\new@command}
5341 \def\new@command#1{%
5342   \@testopt{\@newcommand#1}0}

```

```

5343 \def\@newcommand#1[#2]{%
5344   \@ifnextchar [{\@xargdef#1[#2]]}%
5345   {\@argdef#1[#2]}}
5346 \long\def\@argdef#1[#2]#3{%
5347   \@yargdef#1\@ne{#2}{#3}}
5348 \long\def\@xargdef#1[#2][#3]#4{%
5349   \expandafter\def\expandafter#1\expandafter{%
5350     \expandafter\@protected@testopt\expandafter #1%
5351     \csname\string#1\expandafter\endcsname{#3}}}%
5352   \expandafter\@yargdef \csname\string#1\endcsname
5353   \tw@{#2}{#4}}
5354 \long\def\@yargdef#1#2#3{%
5355   \@tempcnta#3\relax
5356   \advance \@tempcnta \@ne
5357   \let\@hash\relax
5358   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
5359   \@tempcntb #2%
5360   \@whilenum \@tempcntb < \@tempcnta
5361   \do{%
5362     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
5363     \advance\@tempcntb \@ne}%
5364   \let\@hash@###
5365   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
5366 \def\providecommand{\@star@or@long\provide@command}
5367 \def\provide@command#1{%
5368   \begingroup
5369   \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
5370   \endgroup
5371   \expandafter\@ifundefined\@gtempa
5372   {\def\reserved@a{\new@command#1}}%
5373   {\let\reserved@a\relax
5374     \def\reserved@a{\new@command\reserved@a}}%
5375   \reserved@a}%

5376 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
5377 \def\declare@robustcommand#1{%
5378   \edef\reserved@a{\string#1}%
5379   \def\reserved@b{#1}%
5380   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
5381   \edef#1{%
5382     \ifx\reserved@a\reserved@b
5383       \noexpand\x@protect
5384       \noexpand#1%
5385     \fi
5386     \noexpand\protect
5387     \expandafter\noexpand\csname
5388       \expandafter\@gobble\string#1 \endcsname
5389   }%
5390   \expandafter\new@command\csname
5391     \expandafter\@gobble\string#1 \endcsname
5392 }
5393 \def\x@protect#1{%
5394   \ifx\protect\@typeset@protect\else
5395     \@x@protect#1%
5396   \fi
5397 }
5398 \def\@x@protect#1\fi#2#3{%
5399   \fi\protect#1%
5400 }

```

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`.

```

5401 \def\bbl@tempa{\csname newif\endcsname\ifin@}
5402 \ifx\in@\@undefined
5403   \def\in@#1#2{%
5404     \def\in@##1##2##3\in@{%
5405       \ifx\in@##2\in@false\else\in@true\fi}%
5406     \in@#2#1\in@\in@}
5407 \else
5408   \let\bbl@tempa\@empty
5409 \fi
5410 \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).

```

5411 \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.

```

5412 \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.

```

5413 \ifx\@tempcnta\@undefined
5414   \csname newcount\endcsname\@tempcnta\relax
5415 \fi
5416 \ifx\@tempcntb\@undefined
5417   \csname newcount\endcsname\@tempcntb\relax
5418 \fi

```

To prevent wasting two counters in  $\text{\LaTeX 2.09}$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

5419 \ifx\bye\@undefined
5420   \advance\count10 by -2\relax
5421 \fi
5422 \ifx\@ifnextchar\@undefined
5423   \def\@ifnextchar#1#2#3{%
5424     \let\reserved@d=#1%
5425     \def\reserved@a{#2}\def\reserved@b{#3}%
5426     \futurelet\@let@token\@ifnch}
5427   \def\@ifnch{%
5428     \ifx\@let@token\@sptoken
5429       \let\reserved@c\@xifnch
5430     \else
5431       \ifx\@let@token\reserved@d
5432         \let\reserved@c\reserved@a
5433       \else
5434         \let\reserved@c\reserved@b
5435       \fi
5436     \fi
5437     \reserved@c}

```

```

5438 \def\:\let\@sptoken= }\: % this makes \@sptoken a space token
5439 \def\:\@xifnch\ \expandafter\def\:\{\futurelet\@let@token\@ifnch}
5440 \fi
5441 \def\@testopt#1#2{%
5442   \@ifnextchar[{\#1}{\#1[\#2]}}
5443 \def\@protected@testopt#1{%
5444   \ifx\protect\@typeset@protect
5445     \expandafter\@testopt
5446   \else
5447     \@x@protect#1%
5448   \fi}
5449 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{\#1\relax
5450   #2\relax}\fi}
5451 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
5452   \else\expandafter\@gobble\fi{\#1}}

```

## 17.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\TeX$  environment.

```

5453 \def\DeclareTextCommand{%
5454   \@dec@text@cmd\providecommand
5455 }
5456 \def\ProvideTextCommand{%
5457   \@dec@text@cmd\providecommand
5458 }
5459 \def\DeclareTextSymbol#1#2#3{%
5460   \@dec@text@cmd\chardef#1{\#2}\#3\relax
5461 }
5462 \def\@dec@text@cmd#1#2#3{%
5463   \expandafter\def\expandafter#2%
5464     \expandafter{%
5465       \csname#3-cmd\expandafter\endcsname
5466       \expandafter#2%
5467       \csname#3\string#2\endcsname
5468     }%
5469 %   \let\@ifdefinable\@rc@ifdefinable
5470   \expandafter#1\csname#3\string#2\endcsname
5471 }
5472 \def\@current@cmd#1{%
5473   \ifx\protect\@typeset@protect\else
5474     \noexpand#1\expandafter\@gobble
5475   \fi
5476 }
5477 \def\@changed@cmd#1#2{%
5478   \ifx\protect\@typeset@protect
5479     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
5480       \expandafter\ifx\csname ?\string#1\endcsname\relax
5481         \expandafter\def\csname ?\string#1\endcsname{%
5482           \@changed@x@err{\#1}%
5483         }%
5484       \fi
5485       \global\expandafter\let
5486         \csname\cf@encoding \string#1\expandafter\endcsname
5487         \csname ?\string#1\endcsname
5488     \fi
5489     \csname\cf@encoding\string#1%
5490       \expandafter\endcsname
5491   \else

```

```

5492     \noexpand#1%
5493     \fi
5494 }
5495 \def\@changed@x@err#1{%
5496     \errhelp{Your command will be ignored, type <return> to proceed}%
5497     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
5498 \def\DeclareTextCommandDefault#1{%
5499     \DeclareTextCommand#1?%
5500 }
5501 \def\ProvideTextCommandDefault#1{%
5502     \ProvideTextCommand#1?%
5503 }
5504 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
5505 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
5506 \def\DeclareTextAccent#1#2#3{%
5507     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
5508 }
5509 \def\DeclareTextCompositeCommand#1#2#3#4{%
5510     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
5511     \edef\reserved@b{\string##1}%
5512     \edef\reserved@c{%
5513         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
5514     \ifx\reserved@b\reserved@c
5515         \expandafter\expandafter\expandafter\ifx
5516             \expandafter\@car\reserved@a\relax\relax\@nil
5517             \@text@composite
5518         \else
5519             \edef\reserved@b##1{%
5520                 \def\expandafter\noexpand
5521                     \csname#2\string#1\endcsname####1{%
5522                         \noexpand\@text@composite
5523                         \expandafter\noexpand\csname#2\string#1\endcsname
5524                         ####1\noexpand\@empty\noexpand\@text@composite
5525                         {##1}%
5526                     }%
5527                 }%
5528                 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
5529             \fi
5530             \expandafter\def\csname\expandafter\string\csname
5531                 #2\endcsname\string#1-\string#3\endcsname{#4}
5532             \else
5533                 \errhelp{Your command will be ignored, type <return> to proceed}%
5534                 \errmessage{\string\DeclareTextCompositeCommand\space used on
5535                     inappropriate command \protect#1}
5536             \fi
5537         }
5538     \def\@text@composite#1#2#3\@text@composite{%
5539         \expandafter\@text@composite@x
5540         \csname\string#1-\string#2\endcsname
5541     }
5542     \def\@text@composite@x#1#2{%
5543         \ifx#1\relax
5544             #2%
5545         \else
5546             #1%
5547         \fi
5548     }
5549 %
5550 \def\@strip@args#1:#2-#3\@strip@args{#2}

```



```

5551 \def\DeclareTextComposite#1#2#3#4{%
5552   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
5553   \bgroup
5554     \lccode` \@=#4%
5555     \lowercase{%
5556   \egroup
5557     \reserved@a @%
5558   }%
5559 }
5560 %
5561 \def\UseTextSymbol#1#2{%
5562 %   \let\@curr@enc\cf@encoding
5563 %   \@use@text@encoding{#1}%
5564   #2%
5565 %   \@use@text@encoding\@curr@enc
5566 }
5567 \def\UseTextAccent#1#2#3{%
5568 %   \let\@curr@enc\cf@encoding
5569 %   \@use@text@encoding{#1}%
5570 %   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
5571 %   \@use@text@encoding\@curr@enc
5572 }
5573 \def\@use@text@encoding#1{%
5574 %   \edef\@f@encoding{#1}%
5575 %   \xdef\font@name{%
5576 %     \csname\curr@fontshape/\@f@size\endcsname
5577 %   }%
5578 %   \pickup@font
5579 %   \font@name
5580 %   \@@enc@update
5581 }
5582 \def\DeclareTextSymbolDefault#1#2{%
5583   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}{#1}}%
5584 }
5585 \def\DeclareTextAccentDefault#1#2{%
5586   \DeclareTextCommandDefault#1{\UseTextAccent{#2}{#1}}%
5587 }
5588 \def\cf@encoding{OT1}

```

Currently we only use the  $\LaTeX 2_{\epsilon}$  method for accents for those that are known to be made active in *some* language definition file.

```

5589 \DeclareTextAccent{"}{OT1}{127}
5590 \DeclareTextAccent{'}{OT1}{19}
5591 \DeclareTextAccent{^}{OT1}{94}
5592 \DeclareTextAccent`}{OT1}{18}
5593 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN  $\TeX$ .

```

5594 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
5595 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
5596 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
5597 \DeclareTextSymbol{\textquoteright}{OT1}{''}
5598 \DeclareTextSymbol{\i}{OT1}{16}
5599 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\LaTeX$ -control sequence `\scriptsize` to be available. Because plain  $\TeX$  doesn't have such a sophisticated font mechanism as  $\LaTeX$  has, we just `\let` it to `\sevenrm`.

```

5600 \ifx\scriptsize\undefined

```

5601 \let\scriptsize\sevenrm  
 5602 \fi  
 5603 </plain>

## 18 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Donald E. Knuth, *The T<sub>E</sub>Xbook*, Addison-Wesley, 1986.
- [3] Leslie Lamport, *L<sup>A</sup>T<sub>E</sub>X, A document preparation System*, Addison-Wesley, 1986.
- [4] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).
- [5] Hubert Partl, *German T<sub>E</sub>X*, *TUGboat* 9 (1988) #1, p. 70–72.
- [6] Leslie Lamport, in: T<sub>E</sub>Xhax Digest, Volume 89, #13, 17 February 1989.
- [7] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national L<sup>A</sup>T<sub>E</sub>X styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [8] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [9] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [10] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [11] Joachim Schrod, *International L<sup>A</sup>T<sub>E</sub>X is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L<sup>A</sup>T<sub>E</sub>X*, Springer, 2002, p. 301–373.