

# Babel

Version 3.43.2002  
2020/05/09

*Original author*  
Johannes L. Braams

*Current maintainer*  
Javier Bezos

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

# Contents

<b>I</b>	<b>User guide</b>	<b>4</b>
<b>1</b>	<b>The user interface</b>	<b>4</b>
1.1	Monolingual documents . . . . .	4
1.2	Multilingual documents . . . . .	6
1.3	Mostly monolingual documents . . . . .	7
1.4	Modifiers . . . . .	8
1.5	Troubleshooting . . . . .	8
1.6	Plain . . . . .	9
1.7	Basic language selectors . . . . .	9
1.8	Auxiliary language selectors . . . . .	10
1.9	More on selection . . . . .	10
1.10	Shorthands . . . . .	12
1.11	Package options . . . . .	15
1.12	The base option . . . . .	17
1.13	ini files . . . . .	18
1.14	Selecting fonts . . . . .	25
1.15	Modifying a language . . . . .	27
1.16	Creating a language . . . . .	28
1.17	Digits and counters . . . . .	31
1.18	Accessing language info . . . . .	33
1.19	Hyphenation and line breaking . . . . .	34
1.20	Selection based on BCP 47 tags . . . . .	36
1.21	Selecting scripts . . . . .	37
1.22	Selecting directions . . . . .	38
1.23	Language attributes . . . . .	42
1.24	Hooks . . . . .	42
1.25	Languages supported by babel with ldf files . . . . .	43
1.26	Unicode character properties in luatex . . . . .	45
1.27	Tweaking some features . . . . .	45
1.28	Tips, workarounds, known issues and notes . . . . .	45
1.29	Current and future work . . . . .	46
1.30	Tentative and experimental code . . . . .	47
<b>2</b>	<b>Loading languages with language.dat</b>	<b>47</b>
2.1	Format . . . . .	48
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>48</b>
3.1	Guidelines for contributed languages . . . . .	50
3.2	Basic macros . . . . .	50
3.3	Skeleton . . . . .	51
3.4	Support for active characters . . . . .	52
3.5	Support for saving macro definitions . . . . .	53
3.6	Support for extending macros . . . . .	53
3.7	Macros common to a number of languages . . . . .	53
3.8	Encoding-dependent strings . . . . .	54
<b>4</b>	<b>Changes</b>	<b>57</b>
4.1	Changes in babel version 3.9 . . . . .	57
<b>II</b>	<b>Source code</b>	<b>58</b>

<b>5</b>	<b>Identification and loading of required files</b>	<b>58</b>
<b>6</b>	<b>locale directory</b>	<b>58</b>
<b>7</b>	<b>Tools</b>	<b>59</b>
7.1	Multiple languages . . . . .	63
7.2	The Package File ( $\LaTeX$ , babel.sty) . . . . .	64
7.3	base . . . . .	65
7.4	Conditional loading of shorthands . . . . .	67
7.5	Cross referencing macros . . . . .	69
7.6	Marks . . . . .	72
7.7	Preventing clashes with other packages . . . . .	73
7.7.1	ifthen . . . . .	73
7.7.2	varioref . . . . .	73
7.7.3	hhline . . . . .	74
7.7.4	hyperref . . . . .	74
7.7.5	fancyhdr . . . . .	74
7.8	Encoding and fonts . . . . .	75
7.9	Basic bidi support . . . . .	77
7.10	Local Language Configuration . . . . .	82
<b>8</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>85</b>
8.1	Tools . . . . .	86
<b>9</b>	<b>Multiple languages</b>	<b>86</b>
9.1	Selecting the language . . . . .	89
9.2	Errors . . . . .	98
9.3	Hooks . . . . .	100
9.4	Setting up language files . . . . .	102
9.5	Shorthands . . . . .	104
9.6	Language attributes . . . . .	114
9.7	Support for saving macro definitions . . . . .	116
9.8	Short tags . . . . .	117
9.9	Hyphens . . . . .	117
9.10	Multiencoding strings . . . . .	119
9.11	Macros common to a number of languages . . . . .	124
9.12	Making glyphs available . . . . .	125
9.12.1	Quotation marks . . . . .	125
9.12.2	Letters . . . . .	126
9.12.3	Shorthands for quotation marks . . . . .	127
9.12.4	Umlauts and tremas . . . . .	128
9.13	Layout . . . . .	129
9.14	Load engine specific macros . . . . .	130
9.15	Creating and modifying languages . . . . .	130
<b>10</b>	<b>Adjusting the Babel bahavior</b>	<b>146</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>147</b>
<b>12</b>	<b>Font handling with fontspec</b>	<b>152</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>156</b>
13.1	XeTeX . . . . .	156
13.2	Layout . . . . .	158
13.3	LuaTeX . . . . .	160
13.4	Southeast Asian scripts . . . . .	165
13.5	CJK line breaking . . . . .	169
13.6	Automatic fonts and ids switching . . . . .	169
13.7	Layout . . . . .	177
13.8	Auto bidi with basic and basic-r . . . . .	179
<b>14</b>	<b>Data for CJK</b>	<b>190</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>190</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>191</b>
16.1	Not renaming hyphen.tex . . . . .	191
16.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	192
16.3	General tools . . . . .	192
16.4	Encoding related macros . . . . .	196
<b>17</b>	<b>Acknowledgements</b>	<b>199</b>

## Troubleshooting

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

## Part I

# User guide

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

## 1 The user interface

### 1.1 Monolingual documents

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

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

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[russian]{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

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

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

## 1.2 Multilingual documents

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

**EXAMPLE** In  $\LaTeX$ , the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

**EXAMPLE** A full bilingual document follows. The main language is french, which is activated when the document begins. The package `inputenc` may be omitted with  $\LaTeX \geq 2018-04-01$  if the encoding is UTF-8.

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

And an English paragraph, with a short text in
\foreignlanguage{french}{français}.

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

### 1.3 Mostly monolingual documents

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



loaded on the fly when the language is selected (and also when provided in the optional argument of `\babel font`, if used.)

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

**EXAMPLE** A trivial document is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

## 1.4 Modifiers

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

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

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

## 1.5 Troubleshooting

- Loading directly sty files in  $\text{\LaTeX}$  (ie, `\usepackage{<language>}`) is deprecated and you will get the error:<sup>2</sup>

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

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

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

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

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

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

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

## 1.6 Plain

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

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

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

## 1.7 Basic language selectors

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

## 1.8 Auxiliary language selectors

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

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

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

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

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

Spaces after the environment are ignored.

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

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

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

`\begin{hyphenrules}`  $\{\langle language \rangle\}$  ... `\end{hyphenrules}`

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

## 1.9 More on selection

---

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

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

**EXAMPLE** With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and  $\text{\today}$  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:

---

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

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

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

## 1.10 Shorthands

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

**NOTE** Note the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## `\languageshorthands` $\{\langle language \rangle\}$

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

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

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

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

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

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

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

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

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

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

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

**Basque** " ' ~

**Breton** : ; ? !

**Catalan** " ' `

**Czech** " -

**Esperanto** ^

**Estonian** " ~

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

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

**French** (all varieties) : ; ? !  
**Galician** " , ' ~ < >  
**Greek** ~  
**Hungarian** `  
**Kurmanji** ^  
**Latin** " ^ =  
**Slovak** " ^ ' -  
**Spanish** " , < > ' ~  
**Turkish** : ! =

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

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

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

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

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

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

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

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

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

## 1.11 Package options

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

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

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

**activegrave** Same for `.



**shorthands=**  $\langle char \rangle \langle char \rangle \dots$  | off

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

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

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

**safe=** none | ref | bib

Some  $\LaTeX$  macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in  $\epsilon\TeX$  based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

**math=** active | normal

Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like  $\{a\}$  (a closing brace after a shorthand) are not a source of trouble anymore.

**config=**  $\langle file \rangle$

Load  $\langle file \rangle$ .cfg instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).

**main=**  $\langle language \rangle$

Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.

**headfoot=**  $\langle language \rangle$

By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.

**noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected .cfg file. However, if the key config is set, this file is loaded.

**showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.

**nocase** **New 3.91** Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.

---

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

**silent** New 3.9l No warnings and no *infos* are written to the log file.<sup>9</sup>

**strings=** generic | unicode | encoded |  $\langle label \rangle$  |  $\langle font\ encoding \rangle$

Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUppercase and the like (this feature misuses some internal L<sup>A</sup>T<sub>E</sub>X tools, so use it only as a last resort).

**hyphenmap=** off | first | select | other | other\*

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

**off** deactivates this feature and no case mapping is applied;

**first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it's the default if a single language option has been stated;<sup>11</sup>

**select** sets it only at \selectlanguage;

**other** also sets it at otherlanguage;

**other\*** also sets it at otherlanguage\* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other\* for monolingual documents.<sup>12</sup>

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

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

**layout=**

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

## 1.12 The base option

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

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

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

<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.

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

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

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

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

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

### 1.13 ini files

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

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

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

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

LUATEX/XETEX

```
\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

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

```
\babelprovide[import,hyphenrules=+]{lao}  
\babeipatterns{lao}{\ln \l\ \l\ \l\ \l\ \l\ \l\} % Random
```

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

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

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

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

---

af	Afrikaans <sup>ul</sup>	am	Amharic <sup>ul</sup>
agq	Aghem	ar	Arabic <sup>ul</sup>
ak	Akan	ar-DZ	Arabic <sup>ul</sup>

ar-MA	Arabic <sup>ul</sup>	et	Estonian <sup>ul</sup>
ar-SY	Arabic <sup>ul</sup>	eu	Basque <sup>ul</sup>
as	Assamese	ewo	Ewondo
asa	Asu	fa	Persian <sup>ul</sup>
ast	Asturian <sup>ul</sup>	ff	Fulah
az-Cyrl	Azerbaijani	fi	Finnish <sup>ul</sup>
az-Latn	Azerbaijani	fil	Filipino
az	Azerbaijani <sup>ul</sup>	fo	Faroese
bas	Basaa	fr	French <sup>ul</sup>
be	Belarusian <sup>ul</sup>	fr-BE	French <sup>ul</sup>
bem	Bemba	fr-CA	French <sup>ul</sup>
bez	Bena	fr-CH	French <sup>ul</sup>
bg	Bulgarian <sup>ul</sup>	fr-LU	French <sup>ul</sup>
bm	Bambara	fur	Friulian <sup>ul</sup>
bn	Bangla <sup>ul</sup>	fy	Western Frisian
bo	Tibetan <sup>u</sup>	ga	Irish <sup>ul</sup>
brx	Bodo	gd	Scottish Gaelic <sup>ul</sup>
bs-Cyrl	Bosnian	gl	Galician <sup>ul</sup>
bs-Latn	Bosnian <sup>ul</sup>	grc	Ancient Greek <sup>ul</sup>
bs	Bosnian <sup>ul</sup>	gsw	Swiss German
ca	Catalan <sup>ul</sup>	gu	Gujarati
ce	Chechen	guz	Gusii
cgg	Chiga	gv	Manx
chr	Cherokee	ha-GH	Hausa
ckb	Central Kurdish	ha-NE	Hausa <sup>l</sup>
cop	Coptic	ha	Hausa
cs	Czech <sup>ul</sup>	haw	Hawaiian
cu	Church Slavic	he	Hebrew <sup>ul</sup>
cu-Cyrs	Church Slavic	hi	Hindi <sup>u</sup>
cu-Glag	Church Slavic	hr	Croatian <sup>ul</sup>
cy	Welsh <sup>ul</sup>	hsb	Upper Sorbian <sup>ul</sup>
da	Danish <sup>ul</sup>	hu	Hungarian <sup>ul</sup>
dav	Taita	hy	Armenian <sup>u</sup>
de-AT	German <sup>ul</sup>	ia	Interlingua <sup>ul</sup>
de-CH	German <sup>ul</sup>	id	Indonesian <sup>ul</sup>
de	German <sup>ul</sup>	ig	Igbo
dje	Zarma	ii	Sichuan Yi
dsb	Lower Sorbian <sup>ul</sup>	is	Icelandic <sup>ul</sup>
dua	Duala	it	Italian <sup>ul</sup>
dyo	Jola-Fonyi	ja	Japanese
dz	Dzongkha	jgo	Ngomba
ebu	Embu	jmc	Machame
ee	Ewe	ka	Georgian <sup>ul</sup>
el	Greek <sup>ul</sup>	kab	Kabyle
el-polyton	Polytonic Greek <sup>ul</sup>	kam	Kamba
en-AU	English <sup>ul</sup>	kde	Makonde
en-CA	English <sup>ul</sup>	kea	Kabuverdianu
en-GB	English <sup>ul</sup>	khq	Koyra Chiini
en-NZ	English <sup>ul</sup>	ki	Kikuyu
en-US	English <sup>ul</sup>	kk	Kazakh
en	English <sup>ul</sup>	kkj	Kako
eo	Esperanto <sup>ul</sup>	kl	Kalaallisut
es-MX	Spanish <sup>ul</sup>	kln	Kalenjin
es	Spanish <sup>ul</sup>	km	Khmer

kn	Kannada <sup>ul</sup>	pl	Polish <sup>ul</sup>
ko	Korean	pms	Piedmontese <sup>ul</sup>
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese <sup>ul</sup>
ksb	Shambala	pt-PT	Portuguese <sup>ul</sup>
ksf	Bafia	pt	Portuguese <sup>ul</sup>
ksh	Colognian	qu	Quechua
kw	Cornish	rm	Romansh <sup>ul</sup>
ky	Kyrgyz	rn	Rundi
lag	Langi	ro	Romanian <sup>ul</sup>
lb	Luxembourgish	rof	Rombo
lg	Ganda	ru	Russian <sup>ul</sup>
lkt	Lakota	rw	Kinyarwanda
ln	Lingala	rwk	Rwa
lo	Lao <sup>ul</sup>	sa-Beng	Sanskrit
lrc	Northern Luri	sa-Deva	Sanskrit
lt	Lithuanian <sup>ul</sup>	sa-Gujr	Sanskrit
lu	Luba-Katanga	sa-Knda	Sanskrit
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

tk	Turkmen <sup>ul</sup>	wae	Walser
to	Tongan	xog	Soga
tr	Turkish <sup>ul</sup>	yav	Yangben
twq	Tasawaq	yi	Yiddish
tzm	Central Atlas Tamazight	yo	Yoruba
ug	Uyghur	yue	Cantonese
uk	Ukrainian <sup>ul</sup>	zgh	Standard Moroccan Tamazight
ur	Urdu <sup>ul</sup>		
uz-Arab	Uzbek	zh-Hans-HK	Chinese
uz-Cyrl	Uzbek	zh-Hans-MO	Chinese
uz-Latn	Uzbek	zh-Hans-SG	Chinese
uz	Uzbek	zh-Hans	Chinese
vai-Latn	Vai	zh-Hant-HK	Chinese
vai-Vaii	Vai	zh-Hant-MO	Chinese
vai	Vai	zh-Hant	Chinese
vi	Vietnamese <sup>ul</sup>	zh	Chinese
vun	Vunjo	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	belarusian
akan	bemba
albanian	bena
american	bengali
amharic	bodo
ancientgreek	bosnian-cyrillic
arabic	bosnian-cyrl
arabic-algeria	bosnian-latin
arabic-DZ	bosnian-latn
arabic-morocco	bosnian
arabic-MA	brazilian
arabic-syria	breton
arabic-SY	british
armenian	bulgarian
assamese	burmese
asturian	canadian
asu	cantonese
australian	catalan
austrian	centralatlastamazight
azerbaijani-cyrillic	centralkurdish
azerbaijani-cyrl	chechen
azerbaijani-latin	cherokee
azerbaijani-latn	chiga
azerbaijani	chinese-hans-hk
bafia	chinese-hans-mo
bambara	chinese-hans-sg
basaa	chinese-hans
basque	chinese-hant-hk

chinese-hant-mo	fulah
chinese-hant	galician
chinese-simplified-hongkongsarchina	ganda
chinese-simplified-macausarchina	georgian
chinese-simplified-singapore	german-at
chinese-simplified	german-austria
chinese-traditional-hongkongsarchina	german-ch
chinese-traditional-macausarchina	german-switzerland
chinese-traditional	german
chinese	greek
churchslavic	gujarati
churchslavic-cyrs	gusii
churchslavic-oldcyrillic <sup>13</sup>	hausa-gh
churchsslavic-glag	hausa-ghana
churchsslavic-glagolitic	hausa-ne
cognian	hausa-niger
cornish	hausa
croatian	hawaiian
czech	hebrew
danish	hindi
duala	hungarian
dutch	icelandic
dzongkha	igbo
embu	inarisami
english-au	indonesian
english-australia	interlingua
english-ca	irish
english-canada	italian
english-gb	japanese
english-newzealand	jolafonyi
english-nz	kabuverdianu
english-unitedkingdom	kabyle
english-unitedstates	kako
english-us	kalaallisut
english	kalenjin
esperanto	kamba
estonian	kannada
ewe	kashmiri
ewondo	kazakh
faroeese	khmer
filipino	kikuyu
finnish	kinyarwanda
french-be	konkani
french-belgium	korean
french-ca	koyraborosenni
french-canada	koyrachiini
french-ch	kwasio
french-lu	kyrgyz
french-luxembourg	lakota
french-switzerland	langi
french	lao
friulian	latvian

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



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  
polytonicgreek

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	turkmen
shona	ukenglish
sichuanyi	ukrainian
sinhala	upporsorbian
slovak	urdu
slovene	usenglish
slovenian	usorbian
soga	uyghur
somali	uzbek-arab
spanish-mexico	uzbek-arabic
spanish-mx	uzbek-cyrillic
spanish	uzbek-cyrl
standardmoroccantamazight	uzbek-latin
swahili	uzbek-latn
swedish	uzbek
swissgerman	vai-latin
tachelhit-latin	vai-latn
tachelhit-latn	vai-vai
tachelhit-tfng	vai-vaii
tachelhit-tifinagh	vai
tachelhit	vietnam
taita	vietnamese
tamil	vunjo
tasawaq	walser
telugu	welsh
teso	westernfrisian
thai	yangben
tibetan	yiddish
tigrinya	yoruba
tongan	zarma
turkish	zulu afrikaans

---

### Modifying and adding values to ini files

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

## 1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עֵבְרִית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

## 1.15 Modifying a language

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

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

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

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

- The new way, which is found in bulgarian, azerbaijani, spanish, french, turkish, icelandic, vietnamese and a few more, as well as in languages created with `\babelprovide` and its key `import`, is:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 1.16 Creating a language

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

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

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

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

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

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

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

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

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

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

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

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

The main language is not changed (danish in this example). So, you must add

`\selectlanguage{arhinish}` or other selectors where necessary.

If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

`import=` *<language-tag>*

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

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

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

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

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

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

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

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

Loads only the strings. For example:

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

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

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

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

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

A special value is `+`, which allocates a new language (in the  $\text{T}_{\text{E}}\text{X}$  sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with `luatex`, because you can add some patterns with `\babelpatterns`, as for example:

```
\babelprovide[hyphenrules=+]{neo}  
\babelpatterns[neo]{a1 e1 i1 o1 u1}
```

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

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

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

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

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

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

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

**onchar=** ids | fonts

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

**mapfont=** direction

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

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

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

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

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

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

## 1.17 Digits and counters

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

For example:

```
\babelprovide[import]{telugu} % Telugu better with XeTeX
% Or also, if you want:
% \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami}
```



```
\begin{document}
\telugudigits{1234}
\telugucounter{section}
\end{document}
```

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

The styles are:

**Ancient Greek** `lower.ancient`, `upper.ancient`

**Arabic** `abjad`, `maghrebi.abjad`

**Belarusan, Bulgarian, Macedonian, Serbian** `lower`, `upper`

**Hebrew** `letters` (neither `geresh` nor `gershayim yet`)

**Hindi** `alphabetic`

**Armenian** `lower.letter`, `upper.letter`

**Japanese** `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`, `informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

**Georgian** `letters`

**Greek** `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with `keraia`)

**Khmer** `consonant`

**Korean** `consonant`, `syllabe`, `hanja.informal`, `hanja.formal`, `hangul.formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

**Persian** `abjad`, `alphabetic`

**Russian** lower, lower.full, upper, upper.full  
**Tamil** ancient  
**Thai** alphabetic  
**Ukrainian** lower, lower.full, upper, upper.full  
**Chinese** cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

## 1.18 Accessing language info

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

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

**\iflanguage** `{\language}{\true}{\false}`

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

**\localeinfo** `{\field}`

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

`name.english` as provided by the Unicode CLDR.  
`tag.ini` is the tag of the ini file (the way this file is identified in its name).  
`tag.bcp47` is the BCP 47 language tag.  
`tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).  
`script.name` as provided by the Unicode CLDR.  
`script.tag.bcp47` is the BCP 47 language tag of the script used by this locale.  
`script.tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).

**\getlocaleproperty** `{\macro}{\locale}{\property}`

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

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

the macro `\hechap` will contain the string פֶּרֶק.  
 Babel remembers which ini files have been loaded. There is a loop named `\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that `\LocaleForEach{\message{ **#1** }} just shows the loaded ini's.`

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

`\localeid`

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

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

## 1.19 Hyphenation and line breaking

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

`\babelhyphen` `*{\type}`

`\babelhyphen` `*{\text}`

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

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

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

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

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

There are also some differences with `LaTeX`: (1) the character used is that set for the current font, while in `LaTeX` it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in `LaTeX`, but it can be changed to another value by redefining `\babenullhyphen`; (3) a break after the hyphen is forbidden if preceded by a

glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

**\babelhyphenation** [*<language>*, *<language>*, ...]{*<exceptions>*}

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

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

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

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

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

**New 3.9m** In *luatex* only,<sup>15</sup> adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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

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

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

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

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

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

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

```

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

```

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

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

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

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

```

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

```

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

## 1.20 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

```

```

\babeladjust{ autoloading.bcp47 = on }

\begin{document}

\today

\selectlanguage{fr-CA}

\today

\end{document}

```

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

`autoloading.bcp47` with values on and off.

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

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

## 1.21 Selecting scripts

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

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

`\ensureascii` `{⟨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

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

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

example, if you load LY1, LGR, then it is set to LY1, but if you load LY1, T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

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

## 1.22 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```

\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

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

```

\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}

```

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

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

```

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

```

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



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

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

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

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

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

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

**WARNING** As of April 2019 there is a bug with `\parshape` in `luatex` (a `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>18</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 pdfTeX must be marked up explicitly (unlike LaTeX with `bidi=basic` or `bidi=basic-r` and, usually, `xetex`). 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.

## 1.23 Language attributes

### `\languageattribute`

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

## 1.24 Hooks

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

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

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

Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three  $\text{T}_{\text{E}}\text{X}$  parameters (`#1`, `#2`, `#3`), with the meaning given:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 1.25 Languages supported by babel with ldf files

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

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

**Dutch** dutch  
**English** english, USenglish, american, UKenglish, british, canadian, australian, newzealand  
**Esperanto** esperanto  
**Estonian** estonian  
**Finnish** finnish  
**French** french, francais, canadien, acadian  
**Galician** galician  
**German** austrian, german, germanb, ngerman, naustrian  
**Greek** greek, polutonikogreek  
**Hebrew** hebrew  
**Icelandic** icelandic  
**Indonesian** indonesian (bahasa, indon, bahasai)  
**Interlingua** interlingua  
**Irish Gaelic** irish  
**Italian** italian  
**Latin** latin  
**Lower Sorbian** lowersorbian  
**Malay** malay, melayu (bahasam)  
**North Sami** samin  
**Norwegian** norsk, nynorsk  
**Polish** polish  
**Portuguese** portuguese, brazilian (portuges, brazil)<sup>19</sup>  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** uppsorbian  
**Welsh** welsh

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

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

```

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

```

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

---

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

## 1.26 Unicode character properties in luatex

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

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

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

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

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

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

## 1.27 Tweaking some features

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

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

## 1.28 Tips, workarounds, known issues and notes

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

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

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

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

```
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrasrussian{\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>20</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of T<sub>E</sub>X, not of babel. Alternatively, you may use `\useshortands` to activate ' and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the .aux file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make 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.

## 1.29 Current and future work

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

<sup>20</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.



Useful additions would be, for example, time, currency, addresses and personal names.<sup>21</sup> But that is the easy part, because they don't require modifying the  $\LaTeX$  internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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

### 1.30 Tentative and experimental code

See the code section for  $\backslash\text{foreignlanguage}^*$  (a new starred version of  $\backslash\text{foreignlanguage}$ ).

#### Old and deprecated stuff

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

- $\backslash\text{babelFSstore}\{\langle\text{babel-language}\rangle\}$  sets the current three basic families (rm, sf, tt) as the default for the language given.
- $\backslash\text{babelFSdefault}\{\langle\text{babel-language}\rangle\}\{\langle\text{fontspec-features}\rangle\}$  patches  $\backslash\text{fontspec}$  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

$\text{T}_{\text{E}}\text{X}$  and most engines based on it ( $\text{pdfT}_{\text{E}}\text{X}$ ,  $\text{xetex}$ ,  $\epsilon\text{-T}_{\text{E}}\text{X}$ , the main exception being  $\text{luatex}$ ) require hyphenation patterns to be preloaded when a format is created (eg,  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ ,  $\text{XeL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ ,  $\text{pdfL}_{\text{A}}\text{T}_{\text{E}}\text{X}$ ).  $\text{babel}$  provides a tool which has become standard in many distributions and based on a “configuration file” named  $\text{language.dat}$ . 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  $\text{luatex}$ , however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).<sup>22</sup> Until 3.9n, this task was delegated to the package  $\text{luatex-hyphen}$ , by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named  $\text{language.dat.lua}$ , but now a new mechanism has been devised based solely on  $\text{language.dat}$ . **You must rebuild**

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

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



**the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).<sup>23</sup>

## 2.1 Format

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

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

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

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

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

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

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

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

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

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

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

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

The following assumptions are made:

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

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

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

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

Some recommendations:

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

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

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

### 3.1 Guidelines for contributed languages

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

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

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

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

### 3.2 Basic macros

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

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

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define this language as a ‘dialect’ of the language for which the patterns were loaded as `\language0`. Here “language” is used in the TeX sense of set of hyphenation patterns.

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

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

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

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *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.
<code>\noextras&lt;lang&gt;</code>	Because we want to let the user switch between languages, but we do not know what state $\TeX$ might be in after the execution of <code>\extras&lt;lang&gt;</code> , a macro that brings $\TeX$ into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras&lt;lang&gt;</code> .
<code>\bbl@declare@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the $\LaTeX$ command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, $\LaTeX$ can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions&lt;lang&gt;</code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct $\LaTeX$ to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

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

```

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

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

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

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

```

```

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

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

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

### 3.4 Support for active characters

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

`\initiate@active@char`

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

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

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

`\declare@shorthand`

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

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

The T<sub>E</sub>Xbook states: “Plain T<sub>E</sub>X includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. L<sup>A</sup>T<sub>E</sub>X adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special⟨char⟩` and `\bbl@remove@special⟨char⟩` add and remove the character `⟨char⟩` to these two sets.

### 3.5 Support for saving macro definitions

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

`\babel@save`

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

`\babel@savevariable`

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

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

### 3.6 Support for extending macros

`\addto`

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

### 3.7 Macros common to a number of languages

`\bbl@allowhyphens`

In several languages compound words are used. This means that when T<sub>E</sub>X has to hyphenate such a compound word, it only does so at the ‘-’ that is used in such words. To allow hyphenation in the rest of such a compound word, the macro `\bbl@allowhyphens` can be used.

`\allowhyphens`

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

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

`\set@low@box`

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

`\save@sfc@q`

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

`\bbl@frenchspacing`  
`\bbl@nonfrenchspacing`

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

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

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

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

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

```

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

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

\EndBabelCommands

```

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the



maintainers of the current languages to decide if using it is appropriate.<sup>29</sup>

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

**\AfterBabelCommands**  $\{\langle code \rangle\}$   
The code is delayed and executed at the global scope just after \EndBabelCommands.

**\SetString**  $\{\langle macro-name \rangle\}\{\langle string \rangle\}$   
Adds  $\langle macro-name \rangle$  to the current category, and defines globally  $\langle lang-macro-name \rangle$  to  $\langle code \rangle$  (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).  
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

**\SetStringLoop**  $\{\langle macro-name \rangle\}\{\langle string-list \rangle\}$   
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with abday):

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

#1 is replaced by the roman numeral.

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

```
\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]  
\SetCase  
{\uccode"10=`I\relax  
\lccode`I="10\relax}  
  
\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]  
\SetCase  
\uccode`i=`İ\relax  
\uccode`ı=`I\relax  
\lccode`İ=`i\relax  
\lccode`I=`ı\relax  
  
\StartBabelCommands{turkish}{}  
\SetCase  
\uccode`i="9D\relax  
\uccode"19=`I\relax  
\lccode"9D=`i\relax  
\lccode`I="19\relax
```

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

```
\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

`\SetHyphenMap` `{\to-lower-macros}`

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

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

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

```
\SetHyphenMap{\BabelLowerMM{"100}{\lccode}{\lccode}{\lccode}}
```

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 lead to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with babel were not recognized when called as global options.

## Part II

# Source code

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

## 5 Identification and loading of required files

*Code documentation is still under revision.*

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

The babel package after unpacking consists of the following files:

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

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

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

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

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

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

## 6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

**charset** the encoding used in the ini file.  
**version** of the ini file  
**level** “version” of the ini specification . which keys are available (they may grow in a compatible way) and how they should be read.  
**encodings** a descriptive list of font encodings.  
**[captions]** section of captions in the file charset  
**[captions.licr]** same, but in pure ASCII using the LICR  
**date.long** fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMMM for the month name) and anything outside is text. In addition, [ ] is a non breakable space and [ . ] is an abbreviation dot.

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

## 7 Tools

```
1 <<version=3.43.2002>>
2 <<date=2020/05/09>>
```

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

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

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

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

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

26 #2}}}

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

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

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

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

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

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

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

```
48 \begingroup
49 \gdef\bbl@ifunset#1{%
50   \expandafter\ifx\csname#1\endcsname\relax
51     \expandafter\@firstoftwo
52   \else
53     \expandafter\@secondoftwo
54   \fi}
55 \bbl@ifunset{ifcsname}%
56 {}%
57 {\gdef\bbl@ifunset#1{%
58   \ifcsname#1\endcsname
59     \expandafter\ifx\csname#1\endcsname\relax
```

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

```

60      \bbl@afterelse\expandafter\@firstoftwo
61      \else
62      \bbl@afterfi\expandafter\@secondoftwo
63      \fi
64      \else
65      \expandafter\@firstoftwo
66      \fi}}
67 \endgroup

```

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

```

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

```

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

```

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

```

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

```

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

```

`\bbl@replace`

```

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

```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an

example where it does *not* work is in `\bbl@TG@@date`, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with `\bbl@replace`; I'm not sure ckecking the replacement is really necessary or just paranoia).

```

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

```

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

```

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

```

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

```

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

```

## 7.1 Multiple languages

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

```

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

```

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

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

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

```

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

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or  $\TeX$  2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is



used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).  
 Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

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

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

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

The first two options are for debugging.

```

181 (*package)
182 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
183 \ProvidesPackage{babel}[\langle\date\rangle\ \langle\version\rangle\ The Babel package]
184 \@ifpackagewith{babel}{debug}
185   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
186   \let\bbl@debug\@firstofone}
187   {\providecommand\bbl@trace[1]{}}%
188   \let\bbl@debug\gobble}
189 \langle\Basic macros\rangle
190 % Temporarily repeat here the code for errors
191 \def\bbl@error#1#2{%
192   \begingroup
193     \def\{\MessageBreak}%
194     \PackageError{babel}{#1}{#2}%
195   \endgroup}
196 \def\bbl@warning#1{%
197   \begingroup
198     \def\{\MessageBreak}%
199     \PackageWarning{babel}{#1}%
200   \endgroup}
201 \def\bbl@infowarn#1{%
202   \begingroup
203     \def\{\MessageBreak}%
204     \GenericWarning
205       {(babel) \@spaces\@spaces\@spaces}%
206       {Package babel Info: #1}%
207   \endgroup}
208 \def\bbl@info#1{%
209   \begingroup
210     \def\{\MessageBreak}%
211     \PackageInfo{babel}{#1}%
212   \endgroup}
213 \def\bbl@nocaption{\protect\bbl@nocaption@i}
214 \def\bbl@nocaption@i#1#2% 1: text to be printed 2: caption macro \langXname
215   \global\@namedef{#2}{\textbf{?#1?}}%
216   \@nameuse{#2}%
217   \bbl@warning{%
218     \@backslashchar#2 not set. Please, define\%
219     it in the preamble with something like:\%
220     \string\renewcommand\@backslashchar#2{..\}%
221     Reported}}
222 \def\bbl@tentative{\protect\bbl@tentative@i}

```

```

223 \def\bbl@tentative@i#1{%
224   \bbl@warning{%
225     Some functions for '#1' are tentative.\%
226     They might not work as expected and their behavior\%
227     could change in the future.\%
228     Reported}}
229 \def\@nolanerr#1{%
230   \bbl@error
231   {You haven't defined the language #1\space yet.\%
232     Perhaps you misspelled it or your installation\%
233     is not complete}%
234   {Your command will be ignored, type <return> to proceed}}
235 \def\@nopatterns#1{%
236   \bbl@warning
237   {No hyphenation patterns were preloaded for\%
238     the language '#1' into the format.\%
239     Please, configure your TeX system to add them and\%
240     rebuild the format. Now I will use the patterns\%
241     preloaded for \bbl@nulllanguage\space instead}}
242   % End of errors
243 \@ifpackagewith{babel}{silent}
244   {\let\bbl@info\@gobble
245    \let\bbl@infowarn\@gobble
246    \let\bbl@warning\@gobble}
247   {}
248 %
249 \def\AfterBabelLanguage#1{%
250   \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

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

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

```

### 7.3 base

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

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

```

269 \bbl@trace{Defining option 'base'}
270 \@ifpackagewith{babel}{base}{%
271   \let\bbl@onlyswitch\@empty
272   \let\bbl@provide@locale\relax
273   \input babel.def
274   \let\bbl@onlyswitch\@undefined
275   \ifx\directlua\@undefined
276     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
277   \else
278     \input luababel.def
279     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
280   \fi
281   \DeclareOption{base}{}%
282   \DeclareOption{showlanguages}{}%
283   \ProcessOptions
284   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
285   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
286   \global\let\@ifl@ter@\@ifl@ter
287   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
288   \endinput}{}%
289 % \end{macrocode}
290 %
291 % \subsection{\texttt{key=value} options and other general option}
292 %
293 % The following macros extract language modifiers, and only real
294 % package options are kept in the option list. Modifiers are saved
295 % and assigned to |\BabelModifiers| at |\bbl@load@language|; when
296 % no modifiers have been given, the former is |\relax|. How
297 % modifiers are handled are left to language styles; they can use
298 % |\in@|, loop them with |\@for| or load |keyval|, for example.
299 %
300 % \begin{macrocode}
301 \bbl@trace{key=value and another general options}
302 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
303 \def\bbl@tempb#1.#2{%
304   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
305 \def\bbl@tempd#1.#2\@nnil{%
306   \ifx\@empty#2%
307     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
308   \else
309     \in@{=}{#1}\ifin@
310     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
311   \else
312     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
313     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
314   \fi
315 \fi}
316 \let\bbl@tempc\@empty
317 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
318 \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.

```

319 \DeclareOption{KeepShorthandsActive}{}
320 \DeclareOption{activeacute}{}
321 \DeclareOption{activegrave}{}
322 \DeclareOption{debug}{}

```

```

323 \DeclareOption{noconfigs}{}
324 \DeclareOption{showlanguages}{}
325 \DeclareOption{silent}{}
326 \DeclareOption{mono}{}
327 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
328 % Don't use. Experimental. TODO.
329 \newif\ifbbl@single
330 \DeclareOption{selectors=off}{\bbl@singletrue}
331 <<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.

```

332 \let\bbl@opt@shorthands\@nnil
333 \let\bbl@opt@config\@nnil
334 \let\bbl@opt@main\@nnil
335 \let\bbl@opt@headfoot\@nnil
336 \let\bbl@opt@layout\@nnil

```

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

```

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

```

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

```

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

```

356 \ProcessOptions*

```

## 7.4 Conditional loading of shorthands

If there is no `shorthands=<chars>`, the original babel macros are left untouched, but if there is, these macros are wrapped (in `babel.def`) to define only those given.

A bit of optimization: if there is no `shorthands=`, then `\bbl@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

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

```

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

```

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

```

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

```

378 \edef\bbl@opt@shorthands{%
379   \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.

```

380 \bbl@ifshorthand{'}%
381   {\PassOptionsToPackage{activeacute}{babel}}{}
382 \bbl@ifshorthand`}%
383   {\PassOptionsToPackage{activegrave}{babel}}{}
384 \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.

```

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

```

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

```

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

```

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

```

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

```
411 \input babel.def
```

## 7.5 Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

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

```

412 << *More package options >> ≡
413 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
414 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
415 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
416 << /More package options >>

```

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

```

417 \bbl@trace{Cross referencing macros}
418 \ifx\bbl@opt@safe\@empty\else
419   \def\@newl@bel#1#2#3{%
420     {\@safe@activestrue
421       \bbl@ifunset{#1@#2}%
422       \relax
423       {\gdef\@multiplelabels{%
424         \@latex@warning@no@line{There were multiply-defined labels}}%
425       \@latex@warning@no@line{Label `#2' multiply defined}}%
426       \global\@namedef{#1@#2}{#3}}}%

```

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

```

427 \CheckCommand*\@testdef[3]{%
428   \def\reserved@a{#3}%

```

```

429 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
430 \else
431 \@tempswatrue
432 \fi}

```

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

```

433 \def\@testdef#1#2#3{% TODO. With @samestring?
434 \@safe@activetrue
435 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
436 \def\bbl@tempb{#3}%
437 \@safe@activesfalse
438 \ifx\bbl@tempa\relax
439 \else
440 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
441 \fi
442 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
443 \ifx\bbl@tempa\bbl@tempb
444 \else
445 \@tempswatrue
446 \fi}
447 \fi

```

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

```

448 \bbl@xin@{R}\bbl@opt@safe
449 \ifin@
450 \bbl@redefineroobust\ref#1{%
451 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
452 \bbl@redefineroobust\pageref#1{%
453 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
454 \else
455 \let\org@ref\ref
456 \let\org@pageref\pageref
457 \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.

```

458 \bbl@xin@{B}\bbl@opt@safe
459 \ifin@
460 \bbl@redefine\@citex[#1]#2{%
461 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
462 \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.

```

463 \AtBeginDocument{%
464 \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.)

```
465 \def\@citex[#1][#2]#3{%
466   \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
467   \org@@citex[#1][#2]{\@tempa}}%
468 }{}}
```

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

```
469 \AtBeginDocument{%
470   \ifpackageloaded{cite}{%
471     \def\@citex[#1]#2{%
472       \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
473   }}
```

`\nocite` The macro `\nocite` which is used to instruct  $\text{\LaTeX}$  to extract uncited references from the database.

```
474 \bbl@redefine\nocite#1{%
475   \@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.

```
476 \bbl@redefine\bibcite{%
477   \bbl@cite@choice
478   \bibcite}
```

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

```
479 \def\bbl@bibcite#1#2{%
480   \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.

```
481 \def\bbl@cite@choice{%
482   \global\let\bibcite\bbl@bibcite
483   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
484   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
485   \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.

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

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

```
487 \bbl@redefine\@bibitem#1{%
```



```

488 \safe@activetrue\org@@bibitem{#1}\safe@activesfalse}
489 \else
490 \let\org@nocite\nocite
491 \let\org@@citex\citex
492 \let\org@bibtex\@bibtex
493 \let\org@@bibitem\@bibitem
494 \fi

```

## 7.6 Marks

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

```

495 \bbl@trace{Marks}
496 \IfBabelLayout{sectioning}
497   {\ifx\bbl@opt@headfoot\@nnil
498     \g@addto@macro\@resetactivechars{%
499       \set@typeset@protect
500       \expandafter\select@language@x\expandafter{\bbl@main@language}%
501       \let\protect\@noexpand
502       \edef\thepage{% TODO. Only with bidi. See also above
503         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
504     \fi}
505   {\ifbbl@single\else
506     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
507     \markright#1{%
508       \bbl@ifblank{#1}%
509       {\org@markright{}}}%
510     {\toks@{#1}%
511       \bbl@exp{%
512         \org@markright{\protect\foreignlanguage{\language}%
513           {\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 macro, so it's not necessary anymore, but it's preserved for older versions.)

```

514 \ifx\@mkboth\markboth
515   \def\bbl@tempc{\let\@mkboth\markboth}
516 \else
517   \def\bbl@tempc{}
518 \fi
519 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
520 \markboth#1#2{%
521   \protected@edef\bbl@tempb##1{%
522     \protect\foreignlanguage
523       {\language}{\protect\bbl@restore@actives##1}}%
524   \bbl@ifblank{#1}%
525   {\toks@{}}%
526   {\toks@\expandafter{\bbl@tempb{#1}}}%
527   \bbl@ifblank{#2}%
528   {\@temptokena{}}%

```

```

529      {\@temptokena\expandafter{\bbl@tempb{#2}}}%
530      \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}}}
531      \bbl@tempc
532      \fi} % end ifbbl@single, end \IfBabelLayout

```

## 7.7 Preventing clashes with other packages

### 7.7.1 ifthen

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

```

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

```

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

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

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

```

533 \bbl@trace{Preventing clashes with other packages}
534 \bbl@xin@{R}\bbl@opt@safe
535 \ifin@
536   \AtBeginDocument{%
537     \@ifpackageloaded{ifthen}{%
538       \bbl@redefine@long\ifthenelse#1#2#3{%
539         \let\bbl@temp@pref\pageref
540         \let\pageref\org@pageref
541         \let\bbl@temp@ref\ref
542         \let\ref\org@ref
543         \@safe@activestrue
544         \org@ifthenelse{#1}%
545           {\let\pageref\bbl@temp@pref
546            \let\ref\bbl@temp@ref
547            \@safe@activesfalse
548            #2}%
549           {\let\pageref\bbl@temp@pref
550            \let\ref\bbl@temp@ref
551            \@safe@activesfalse
552            #3}%
553         }%
554       }{}%
555     }

```

### 7.7.2 varioref

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

```

556   \AtBeginDocument{%
557     \@ifpackageloaded{varioref}{%
558       \bbl@redefine\@@vpageref#1[#2]#3{%

```

```

559     \@safe@activetrue
560     \org@@@vpageref{#1}{#2}{#3}%
561     \@safe@activesfalse}%
562     \bbl@redefine\hrefpagenum#1#2{%
563     \@safe@activetrue
564     \org@vrefpagenum{#1}{#2}%
565     \@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.

```

566     \expandafter\def\csname Ref \endcsname#1{%
567     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
568     }{}%
569 }
570 \fi

```

### 7.7.3 `hhline`

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

```

571 \AtEndOfPackage{%
572 \AtBeginDocument{%
573 \ifpackageloaded{hhline}%
574 {\expandafter\ifx\csname normal@char\string\endcsname\relax
575 \else
576 \makeatletter
577 \def\@currname{hhline}\input{hhline.sty}\makeatother
578 \fi}%
579 {}}

```

### 7.7.4 `hyperref`

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

```

580 \AtBeginDocument{%
581 \ifx\pdfstringdefDisableCommands\undefined\else
582 \pdfstringdefDisableCommands{\languageshorthands{system}}%
583 \fi}

```

### 7.7.5 `fancyhdr`

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

```

584 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
585 \lowercase{\foreignlanguage{#1}}}

```

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

```

586 \def\substitutefontfamily#1#2#3{%
587   \lowercase{\immediate\openout15=#1#2.fd\relax}%
588   \immediate\write15{%
589     \string\ProvidesFile{#1#2.fd}%
590     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
591     \space generated font description file]^^J
592     \string\DeclareFontFamily{#1}{#2}{^^J
593     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
594     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
595     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
596     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
597     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
598     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
599     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
600     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
601   }%
602   \closeout15
603 }
604 \@onlypreamble\substitutefontfamily

```

## 7.8 Encoding and fonts

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

`\ensureascii`

```

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

```

```

628         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
629         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
630     \else
631         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
632     \fi}%
633 \fi}%
634 \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@ de mas??
635 \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
636 \ifin@else
637     \edef\ensureascii#1{{%
638         \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
639 \fi
640 \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.

```

641 \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.

```

642 \AtBeginDocument{%
643     \@ifpackageloaded{fontspec}%
644     {\xdef\latinencoding{%
645         \ifx\UTFencname\@undefined
646             EU\ifcase\bbl@engine\or2\or1\fi
647         \else
648             \UTFencname
649         \fi}}%
650     {\gdef\latinencoding{OT1}%
651         \ifx\cf@encoding\bbl@t@one
652             \xdef\latinencoding{\bbl@t@one}%
653         \else
654             \ifx\@fontenc@load@list\@undefined
655                 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
656             \else
657                 \def\@elt#1{,#1,}%
658                 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
659                 \let\@elt\relax
660                 \bbl@xin@{,T1,}\bbl@tempa
661                 \ifin@
662                     \xdef\latinencoding{\bbl@t@one}%
663                 \fi
664             \fi
665         \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.

```

666 \DeclareRobustCommand{\latintext}{%
667     \fontencoding{\latinencoding}\selectfont
668     \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.

```
669 \ifx\@undefined\DeclareTextFontCommand
670   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
671 \else
672   \DeclareTextFontCommand{\textlatin}{\latintext}
673 \fi
```

## 7.9 Basic bidi support

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

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

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

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

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

```
674 \ifodd\bb1@engine
675   \def\bb1@activate@preotf{%
676     \let\bb1@activate@preotf\relax % only once
677     \directlua{
678       Babel = Babel or {}
679       %
680       function Babel.pre_otfload_v(head)
681         if Babel.numbers and Babel.digits_mapped then
682           head = Babel.numbers(head)
683         end
684         if Babel.bidi_enabled then
685           head = Babel.bidi(head, false, dir)
686         end
687         return head
688       end
689       %
690       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
691         if Babel.numbers and Babel.digits_mapped then
692           head = Babel.numbers(head)
693         end
694       end
695     }
```

```

693     end
694     if Babel.bidi_enabled then
695         head = Babel.bidi(head, false, dir)
696     end
697     return head
698 end
699 %
700 luatexbase.add_to_callback('pre_linebreak_filter',
701     Babel.pre_otfload_v,
702     'Babel.pre_otfload_v',
703     luatexbase.priority_in_callback('pre_linebreak_filter',
704         'luaotfload.node_processor') or nil)
705 %
706 luatexbase.add_to_callback('hpack_filter',
707     Babel.pre_otfload_h,
708     'Babel.pre_otfload_h',
709     luatexbase.priority_in_callback('hpack_filter',
710         'luaotfload.node_processor') or nil)
711 }}
712 \fi

```

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

```

713 \bbl@trace{Loading basic (internal) bidi support}
714 \ifodd\bbl@engine
715   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
716     \let\bbl@beforeforeign\leavevmode
717     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
718     \RequirePackage{luatexbase}
719     \bbl@activate@preotf
720     \directlua{
721       require('babel-data-bidi.lua')
722       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
723         require('babel-bidi-basic.lua')
724       \or
725         require('babel-bidi-basic-r.lua')
726     \fi}
727     % TODO - to locale_props, not as separate attribute
728     \newattribute\bbl@attr@dir
729     % TODO. I don't like it, hackish:
730     \bbl@exp{\output{\bodydir\pagedir\the\output}}
731     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
732   \fi\fi
733 \else
734   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
735     \bbl@error
736     {The bidi method `basic' is available only in\\%
737       luatex. I'll continue with `bidi=default', so\\%
738       expect wrong results}%
739     {See the manual for further details.}%
740     \let\bbl@beforeforeign\leavevmode
741     \AtEndOfPackage{%
742       \EnableBabelHook{babel-bidi}%
743       \bbl@xebidipar}
744   \fi\fi
745   \def\bbl@loadxebidi#1{%
746     \ifx\RTLfootnotetext\undefined
747       \AtEndOfPackage{%
748         \EnableBabelHook{babel-bidi}%

```

```

749     \ifx\fontspec\@undefined
750     \usepackage{fontspec}% bidi needs fontspec
751     \fi
752     \usepackage#1{bidi}}%
753   \fi}
754 \ifnum\bbbl@bidimode>200
755   \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
756     \bbbl@tentative{bidi=bidi}
757     \bbbl@loadxebidi{}
758   \or
759     \bbbl@tentative{bidi=bidi-r}
760     \bbbl@loadxebidi{[rldocument]}
761   \or
762     \bbbl@tentative{bidi=bidi-l}
763     \bbbl@loadxebidi{}
764   \fi
765 \fi
766 \fi
767 \ifnum\bbbl@bidimode=\@ne
768   \let\bbbl@beforeforeign\leavevmode
769   \ifodd\bbbl@engine
770     \newattribute\bbbl@attr@dir
771     \bbbl@exp{\output{\bodydir\pagedir\the\output}}%
772   \fi
773   \AtEndOfPackage{%
774     \EnableBabelHook{babel-bidi}%
775     \ifodd\bbbl@engine\else
776       \bbbl@xebidipar
777     \fi}
778 \fi

```

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

```

779 \bbbl@trace{Macros to switch the text direction}
780 \def\bbbl@alscripts{,Arabic,Syriac,Thaana,}
781 \def\bbbl@rscripts{% TODO. Base on codes ??
782   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
783   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
784   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
785   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
786   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
787   Old South Arabian,}%
788 \def\bbbl@provide@dirs#1{%
789   \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
790   \ifin@
791     \global\bbbl@csarg\chardef{wdir@#1}\@ne
792     \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
793     \ifin@
794       \global\bbbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
795     \fi
796   \else
797     \global\bbbl@csarg\chardef{wdir@#1}\z@
798   \fi
799   \ifodd\bbbl@engine
800     \bbbl@csarg\ifcase{wdir@#1}%
801       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
802     \or
803       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
804     \or

```



```

805 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
806 \fi
807 \fi}
808 \def\bbl@switchdir{%
809 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}}%
810 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}}%
811 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
812 \def\bbl@setdirs#1{% TODO - math
813 \ifcase\bbl@select@type % TODO - strictly, not the right test
814 \bbl@bodydir{#1}%
815 \bbl@pdir{#1}%
816 \fi
817 \bbl@texmdir{#1}}
818 % TODO. Only if \bbl@bidimode > 0?:
819 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
820 \DisableBabelHook{babel-bidi}

```

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

```

821 \ifodd\bbl@engine % luatex=1
822 \chardef\bbl@thetexmdir\z@
823 \chardef\bbl@thepardir\z@
824 \def\bbl@getluadir#1{%
825 \directlua{
826 if tex.#1dir == 'TLT' then
827 tex.sprint('0')
828 elseif tex.#1dir == 'TRT' then
829 tex.sprint('1')
830 end}}
831 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
832 \ifcase#3\relax
833 \ifcase\bbl@getluadir{#1}\relax\else
834 #2 TLT\relax
835 \fi
836 \else
837 \ifcase\bbl@getluadir{#1}\relax
838 #2 TRT\relax
839 \fi
840 \fi}
841 \def\bbl@texmdir#1{%
842 \bbl@setluadir{tex}\texmdir{#1}%
843 \chardef\bbl@thetexmdir#1\relax
844 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
845 \def\bbl@pdir#1{%
846 \bbl@setluadir{par}\pardir{#1}%
847 \chardef\bbl@thepardir#1\relax}
848 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
849 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
850 \def\bbl@dirparastext{\pardir\the\texmdir\relax}% %%%
851 % Sadly, we have to deal with boxes in math with basic.
852 % Activated every math with the package option bidi=:
853 \def\bbl@mathboxdir{%
854 \ifcase\bbl@thetexmdir\relax
855 \everyhbox{\texmdir TLT\relax}%
856 \else
857 \everyhbox{\texmdir TRT\relax}%
858 \fi}
859 \frozen@everymath\expandafter{%
860 \expandafter\bbl@mathboxdir\the\frozen@everymath}
861 \frozen@everydisplay\expandafter{%

```

```

862 \expandafter\bb1@mathboxdir\the\frozen@everydisplay}
863 \else % pdftex=0, xetex=2
864 \newcount\bb1@dirlevel
865 \chardef\bb1@thetextdir\z@
866 \chardef\bb1@thepardir\z@
867 \def\bb1@textdir#1{%
868 \ifcase#1\relax
869 \chardef\bb1@thetextdir\z@
870 \bb1@textdir@i\beginL\endL
871 \else
872 \chardef\bb1@thetextdir\@ne
873 \bb1@textdir@i\beginR\endR
874 \fi}
875 \def\bb1@textdir@i#1#2{%
876 \ifhmode
877 \ifnum\currentgrouplevel>\z@
878 \ifnum\currentgrouplevel=\bb1@dirlevel
879 \bb1@error{Multiple bidi settings inside a group}%
880 {I'll insert a new group, but expect wrong results.}%
881 \bgroup\aftergroup#2\aftergroup\egroup
882 \else
883 \ifcase\currentgrouptype\or % 0 bottom
884 \aftergroup#2% 1 simple {}
885 \or
886 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
887 \or
888 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
889 \or\or\or % vbox vtop align
890 \or
891 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
892 \or\or\or\or\or\or\or % output math disc insert vcent mathchoice
893 \or
894 \aftergroup#2% 14 \begingroup
895 \else
896 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
897 \fi
898 \fi
899 \bb1@dirlevel\currentgrouplevel
900 \fi
901 #1%
902 \fi}
903 \def\bb1@pardir#1{\chardef\bb1@thepardir#1\relax}
904 \let\bb1@bodydir\@gobble
905 \let\bb1@pagedir\@gobble
906 \def\bb1@dirparastext{\chardef\bb1@thepardir\bb1@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

907 \def\bb1@xebidipar{%
908 \let\bb1@xebidipar\relax
909 \TeXeTstate\@ne
910 \def\bb1@xeverypar{%
911 \ifcase\bb1@thepardir
912 \ifcase\bb1@thetextdir\else\beginR\fi
913 \else
914 {\setbox\z@\lastbox\beginR\box\z@}%
915 \fi}%
916 \let\bb1@severypar\everypar

```

```

917 \newtoks\everypar
918 \everypar=\bbl@severypar
919 \bbl@severypar{\bbl@xeverypar\the\everypar}}
920 \ifnum\bbl@bidimode>200
921 \let\bbl@textdir@i@gobbletwo
922 \let\bbl@xebidipar@empty
923 \AddBabelHook{bidi}{foreign}{%
924   \def\bbl@tempa{\def\BabelText###1}%
925   \ifcase\bbl@thetextdir
926     \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
927   \else
928     \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
929   \fi}
930 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
931 \fi
932 \fi

```

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

```

933 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
934 \AtBeginDocument{%
935   \ifx\pdfstringdefDisableCommands\@undefined\else
936     \ifx\pdfstringdefDisableCommands\relax\else
937       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
938     \fi
939   \fi}

```

## 7.10 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `nor.sk.cfg` will be loaded when the language definition file `nor.sk.ldf` is loaded.

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

```

940 \bbl@trace{Local Language Configuration}
941 \ifx\loadlocalcfg\@undefined
942   \@ifpackagewith{babel}{noconfigs}%
943   {\let\loadlocalcfg\@gobble}%
944   {\def\loadlocalcfg#1{%
945     \InputIfFileExists{#1.cfg}%
946     {\typeout{*****^J%
947               * Local config file #1.cfg used^^J%
948               *}}%
949     \@empty}}
950 \fi

```

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

```

951 \ifx\@unexpandable@protect\@undefined
952   \def\@unexpandable@protect{\noexpand\protect\noexpand}
953   \long\def\protected@write#1#2#3{%
954     \begingroup
955       \let\thepage\relax
956       #2%
957       \let\protect\@unexpandable@protect
958       \edef\reserved@a{\write#1{#3}}%
959       \reserved@a

```

```

960 \endgroup
961 \if@nobreak\ifvmode\nobreak\fi\fi}
962 \fi
963 %
964 % \subsection{Language options}
965 %
966 % Languages are loaded when processing the corresponding option
967 % \textit{except} if a |main| language has been set. In such a
968 % case, it is not loaded until all options has been processed.
969 % The following macro inputs the ldf file and does some additional
970 % checks (\input works, too, but possible errors are not caught).
971 %
972 % \begin{macrocode}
973 \bbl@trace{Language options}
974 \let\bbl@afterlang\relax
975 \let\BabelModifiers\relax
976 \let\bbl@loaded@empty
977 \def\bbl@load@language#1{%
978 \InputIfFileExists{#1.ldf}%
979 {\edef\bbl@loaded{\CurrentOption
980 \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
981 \expandafter\let\expandafter\bbl@afterlang
982 \csname\CurrentOption.ldf-h@k\endcsname
983 \expandafter\let\expandafter\BabelModifiers
984 \csname bbl@mod@\CurrentOption\endcsname}%
985 {\bbl@error{%
986 Unknown option '\CurrentOption'. Either you misspelled it\\
987 or the language definition file \CurrentOption.ldf was not found}{%
988 Valid options are: shorthands=, KeepShorthandsActive,\\
989 activeacute, activegrave, noconfigs, safe=, main=, math=\\
990 headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

991 \def\bbl@try@load@lang#1#2#3{%
992 \IfFileExists{\CurrentOption.ldf}%
993 {\bbl@load@language{\CurrentOption}}%
994 {#1\bbl@load@language{#2}#3}}
995 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
996 \DeclareOption{hebrew}{%
997 \input{rlbabel.def}%
998 \bbl@load@language{hebrew}}
999 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1000 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1001 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1002 \DeclareOption{polutonikogreek}{%
1003 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1004 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1005 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1006 \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.

```

1007 \ifx\bbl@opt@config@nnil

```

```

1008 \ifpackagewith{babel}{noconfigs}{}%
1009   {\InputIfFileExists{bblopts.cfg}%
1010     {\typeout{*****^J%
1011               * Local config file bblopts.cfg used^^J%
1012               *}}}%
1013   }{}%
1014 \else
1015   \InputIfFileExists{\bbl@opt@config.cfg}%
1016   {\typeout{*****^J%
1017             * Local config file \bbl@opt@config.cfg used^^J%
1018             *}}}%
1019   {\bbl@error{%
1020     Local config file '\bbl@opt@config.cfg' not found}{%
1021     Perhaps you misspelled it.}}}%
1022 \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.

```

1023 \bbl@for\bbl@tempa\bbl@language@opts{%
1024   \bbl@ifunset{ds@\bbl@tempa}%
1025   {\edef\bbl@tempb{%
1026     \noexpand\DeclareOption
1027     {\bbl@tempa}%
1028     {\noexpand\bbl@load@language{\bbl@tempa}}}%
1029     \bbl@tempb}%
1030   \@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.

```

1031 \bbl@foreach\@classoptionslist{%
1032   \bbl@ifunset{ds@#1}%
1033   {\IfFileExists{#1.ldf}%
1034     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1035     {}}%
1036   {}}

```

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

```

1037 \ifx\bbl@opt@main\@nnil\else
1038   \expandafter
1039   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1040   \DeclareOption{\bbl@opt@main}{}
1041 \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):

```

1042 \def\AfterBabelLanguage#1{%
1043   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
1044   \DeclareOption*{}
1045   \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`.

```

1046 \bbl@trace{Option 'main'}
1047 \ifx\bbl@opt@main\@nnil
1048 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1049 \let\bbl@tempc\@empty
1050 \bbl@for\bbl@tempb\bbl@tempa{%
1051   \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1052   \ifin@ \edef\bbl@tempc{\bbl@tempb}\fi}
1053 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1054 \expandafter\bbl@tempa\bbl@loaded,\@nnil
1055 \ifx\bbl@tempb\bbl@tempc\else
1056   \bbl@warning{%
1057     Last declared language option is '\bbl@tempc',\%
1058     but the last processed one was '\bbl@tempb'.\%
1059     The main language cannot be set as both a global\%
1060     and a package option. Use 'main=\bbl@tempc' as\%
1061     option. Reported}%
1062 \fi
1063 \else
1064 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1065 \ExecuteOptions{\bbl@opt@main}
1066 \DeclareOption*{}
1067 \ProcessOptions*
1068 \fi
1069 \def\AfterBabelLanguage{%
1070   \bbl@error
1071   {Too late for \string\AfterBabelLanguage}%
1072   {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.

```

1073 \ifx\bbl@main@language\undefined
1074   \bbl@info{%
1075     You haven't specified a language. I'll use 'nil'\%
1076     as the main language. Reported}
1077   \bbl@load@language{nil}
1078 \fi
1079 \</package>
1080 \<*core>

```

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

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

## 8.1 Tools

```
1081 \ifx\ldf@quit\@undefined\else
1082 \endinput\fi % Same line!
1083 <<Make sure ProvidesFile is defined>>
1084 \ProvidesFile{babel.def}[\<date>] <<version>> Babel common definitions]
```

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

```
1085 \ifx\AtBeginDocument\@undefined % TODO. change test.
1086 <<Emulate LaTeX>>
1087 \def\language#1{english}%
1088 \let\bbl@opt@shorthands\@nnil
1089 \def\bbl@ifshorthand#1#2#3{#2}%
1090 \let\bbl@language@opts\@empty
1091 \ifx\babeloptionstrings\@undefined
1092   \let\bbl@opt@strings\@nnil
1093 \else
1094   \let\bbl@opt@strings\babeloptionstrings
1095 \fi
1096 \def\BabelStringsDefault{generic}
1097 \def\bbl@tempa{normal}
1098 \ifx\babeloptionmath\bbl@tempa
1099   \def\bbl@mathnormal{\noexpand\textormath}
1100 \fi
1101 \def\AfterBabelLanguage#1#2{}
1102 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1103 \let\bbl@afterlang\relax
1104 \def\bbl@opt@safe{BR}
1105 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1106 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1107 \expandafter\newif\csname ifbbl@single\endcsname
1108 \chardef\bbl@bidimode\z@
1109 \fi
```

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

```
1110 \ifx\bbl@trace\@undefined
1111   \let\ldf@init\endinput
1112   \def\ProvidesLanguage#1{\endinput}
1113 \endinput\fi % Same line!
```

And continue.

## 9 Multiple languages

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

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

```
1114 <<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.

```
1115 \def\bbl@version{\<version>}
```

```

1116 \def\bbl@date{<<date>>}}
1117 \def\adddialect#1#2{%
1118   \global\chardef#1#2\relax
1119   \bbl@usehooks{adddialect}{#1}{#2}}%
1120 \begingroup
1121   \count@#1\relax
1122   \def\bbl@elt##1##2##3##4{%
1123     \ifnum\count@=##2\relax
1124       \bbl@info{\string#1 = using hyphenrules for ##1\\%
1125         (\string\language\the\count@)}%
1126       \def\bbl@elt####1####2####3####4{%
1127         \fi}%
1128       \bbl@cs{languages}%
1129     \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.

```

1130 \def\bbl@fixname#1{%
1131   \begingroup
1132     \def\bbl@tempe{l@}%
1133     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1134     \bbl@tempd
1135     {\lowercase\expandafter{\bbl@tempd}%
1136      {\uppercase\expandafter{\bbl@tempd}%
1137       \@empty
1138       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1139        \uppercase\expandafter{\bbl@tempd}}}%
1140      {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1141       \lowercase\expandafter{\bbl@tempd}}}%
1142     \@empty
1143     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1144   \bbl@tempd
1145   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1146 \def\bbl@iflanguage#1{%
1147   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

1148 \def\bbl@bcpcase#1#2#3#4\@#5{%
1149   \ifx\@empty#3%
1150     \uppercase{\def#5{#1#2}}%
1151   \else
1152     \uppercase{\def#5{#1}}%
1153     \lowercase{\edef#5{#5#2#3#4}}%
1154   \fi}
1155 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1156   \let\bbl@bcp\relax
1157   \lowercase{\def\bbl@tempa{#1}}%

```



```

1158 \ifx\@empty#2%
1159 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1160 \else\ifx\@empty#3%
1161 \bbl@bcpcase#2\@empty\@empty\@empty\@empty\bbl@tempb
1162 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1163 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1164 {}%
1165 \ifx\bbl@bcp\relax
1166 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1167 \fi
1168 \else
1169 \bbl@bcpcase#2\@empty\@empty\@empty\@empty\bbl@tempb
1170 \bbl@bcpcase#3\@empty\@empty\@empty\@empty\bbl@tempc
1171 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1172 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1173 {}%
1174 \ifx\bbl@bcp\relax
1175 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1176 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1177 {}%
1178 \fi
1179 \ifx\bbl@bcp\relax
1180 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1181 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1182 {}%
1183 \fi
1184 \ifx\bbl@bcp\relax
1185 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1186 \fi
1187 \fi\fi}
1188 \let\bbl@autoload@options\@empty
1189 \let\bbl@initoload\relax
1190 \def\bbl@provide@locale{%
1191 \ifx\babelprovide\undefined
1192 \bbl@error{For a language to be defined on the fly 'base'\\%
1193 is not enough, and the whole package must be\\%
1194 loaded. Either delete the 'base' option or\\%
1195 request the languages explicitly}%
1196 {See the manual for further details.}%
1197 \fi
1198 % TODO. Option to search if loaded, with \LocaleForEach
1199 \let\bbl@auxname\language\language % Still necessary. TODO
1200 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1201 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1202 \ifbbl@bcpallowed
1203 \expandafter\ifx\csname date\language\endcsname\relax
1204 \expandafter
1205 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@empty
1206 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1207 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1208 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1209 \expandafter\ifx\csname date\language\endcsname\relax
1210 \let\bbl@initoload\bbl@bcp
1211 \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1212 \let\bbl@initoload\relax
1213 \fi
1214 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1215 \fi
1216 \fi

```

```

1217 \fi
1218 \expandafter\ifx\csname date\language\endcsname\relax
1219 \IfFileExists{babel-\language.tex}%
1220 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
1221 {}%
1222 \fi}

```

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

```

1223 \def\iflanguage#1{%
1224 \bbl@iflanguage{#1}{%
1225 \ifnum\csname l@#1\endcsname=\language
1226 \expandafter\@firstoftwo
1227 \else
1228 \expandafter\@secondoftwo
1229 \fi}}

```

## 9.1 Selecting the language

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

```

1230 \let\bbl@select@type\z@
1231 \edef\selectlanguage{%
1232 \noexpand\protect
1233 \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`.

```

1234 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

1235 \let\xstring\string

```

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

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

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

```

1236 \def\bbl@language@stack{}

```

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

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

```
1237 \def\bbl@push@language{%
1238   \ifx\language\@undefined\else
1239     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1240   \fi}
```

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

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

```
1241 \def\bbl@pop@lang#1+#2#3{%
1242   \edef\language{#1}\xdef#3{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before `\bbl@pop@lang` is executed  $\TeX$  first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a ‘+’-sign (zero language names won’t occur as this macro will only be called after something has been pushed on the stack) followed by the ‘&’-sign and finally the reference to the stack.

```
1243 \let\bbl@ifrestoring\@secondoftwo
1244 \def\bbl@pop@language{%
1245   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1246   \let\bbl@ifrestoring\@firstoftwo
1247   \expandafter\bbl@set@language\expandafter{\language}%
1248   \let\bbl@ifrestoring\@secondoftwo}
```

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

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

```
1249 \chardef\localeid\z@
1250 \def\bbl@id@last{0} % No real need for a new counter
1251 \def\bbl@id@assign{%
1252   \bbl@ifunset{bbl@id@\language}%
1253   {\count@bbl@id@last\relax
1254    \advance\count@\@ne
1255    \bbl@csarg\chardef{id@\language}\count@
1256    \edef\bbl@id@last{\the\count@}%
1257    \ifcase\bbl@engine\or
1258      \directlua{
1259        Babel = Babel or {}
1260        Babel.locale_props = Babel.locale_props or {}
1261        Babel.locale_props[\bbl@id@last] = {}
1262        Babel.locale_props[\bbl@id@last].name = '\language'
1263      }%
1264    \fi}%
1265  }%
1266  \chardef\localeid\bbl@c{l{id@}}
```

The unprotected part of `\selectlanguage`.

```

1267 \expandafter\def\csname selectlanguage \endcsname#1{%
1268   \ifnum\bbl@hymapsel=\@cc1v\let\bbl@hymapsel\tw@ \fi
1269   \bbl@push@language
1270   \aftergroup\bbl@pop@language
1271   \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.

```

1272 \def\BabelContentsFiles{toc,lof,lot}
1273 \def\bbl@set@language#1{% from selectlanguage, pop@
1274   % The old buggy way. Preserved for compatibility.
1275   \edef\language{%
1276     \ifnum\escapechar=\expandafter`\string#1\@empty
1277     \else\string#1\@empty\fi}%
1278   \ifcat\relax\noexpand#1%
1279     \expandafter\ifx\csname date\language\endcsname\relax
1280       \edef\language{#1}%
1281       \let\localname\language
1282     \else
1283       \bbl@info{Using '\string\language' instead of 'language' is\\%
1284         deprecated. If what you want is to use a\\%
1285         macro containing the actual locale, make\\%
1286         sure it does not not match any language.\\%
1287         Reported}%
1288       I'll\\%
1289       try to fix '\string\localname', but I cannot promise\\%
1290       anything. Reported}%
1291     \ifx\scantokens\undefined
1292       \def\localname{??}%
1293     \else
1294       \scantokens\expandafter{\expandafter
1295         \def\expandafter\localname\expandafter{\language}}%
1296     \fi
1297   \fi
1298 \else
1299   \def\localname{#1}% This one has the correct catcodes
1300 \fi
1301 \select@language{\language}%
1302 % write to auxs
1303 \expandafter\ifx\csname date\language\endcsname\relax\else
1304   \if@filesw
1305     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1306       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname{}}%
1307     \fi
1308     \bbl@usehooks{write}{}%
1309   \fi
1310 \fi}
1311 %
1312 \newif\ifbbl@bcpallowed
1313 \bbl@bcpallowedfalse
1314 \def\select@language#1{% from set@, babel@aux
1315   % set hymap

```

```

1316 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
1317 % set name
1318 \edef\language{#1}%
1319 \bbl@fixname\language
1320 % TODO. name@map must be here?
1321 \bbl@provide@locale
1322 \bbl@iflanguage\language{%
1323   \expandafter\ifx\csname date\language\endcsname\relax
1324     \bbl@error
1325     {Unknown language '\language'. Either you have\\%
1326      misspelled its name, it has not been installed,\\%
1327      or you requested it in a previous run. Fix its name,\\%
1328      install it or just rerun the file, respectively. In\\%
1329      some cases, you may need to remove the aux file}%
1330     {You may proceed, but expect wrong results}%
1331   \else
1332     % set type
1333     \let\bbl@select@type\z@
1334     \expandafter\bbl@switch\expandafter{\language}%
1335   \fi}}
1336 \def\babel@aux#1#2{%
1337   \select@language{#1}%
1338   \bbl@foreach\BabelContentsFiles{%
1339     \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1340 \def\babel@toc#1#2{%
1341   \select@language{#1}}

```

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

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

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

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

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

```

1342 \newif\ifbbl@usedategroup
1343 \def\bbl@switch#1{% from select@, foreign@
1344   % make sure there is info for the language if so requested
1345   \bbl@ensureinfo{#1}%
1346   % restore
1347   \originalTeX
1348   \expandafter\def\expandafter\originalTeX\expandafter{%
1349     \csname noextras#1\endcsname
1350     \let\originalTeX\@empty
1351     \babel@beginsave}%
1352   \bbl@usehooks{afterreset}}}%
1353 \languageshorthands{none}%
1354 % set the locale id
1355 \bbl@id@assign
1356 % switch captions, date
1357 \ifcase\bbl@select@type

```

```

1358 \ifhmode
1359 \hskip\z@skip % trick to ignore spaces
1360 \csname captions#1\endcsname\relax
1361 \csname date#1\endcsname\relax
1362 \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1363 \else
1364 \csname captions#1\endcsname\relax
1365 \csname date#1\endcsname\relax
1366 \fi
1367 \else
1368 \ifhmode
1369 \hskip\z@skip % trick to ignore spaces
1370 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1371 \ifin@
1372 \csname captions#1\endcsname\relax
1373 \fi
1374 \bbl@xin@{,date,}{,\bbl@select@opts,}%
1375 \ifin@ % if \foreign... within \<lang>date
1376 \csname date#1\endcsname\relax
1377 \fi
1378 \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1379 \else
1380 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1381 \ifin@
1382 \csname captions#1\endcsname\relax
1383 \fi
1384 \bbl@xin@{,date,}{,\bbl@select@opts,}%
1385 \ifin@
1386 \csname date#1\endcsname\relax
1387 \fi
1388 \fi
1389 \fi
1390 % switch extras
1391 \bbl@usehooks{beforeextras}{}%
1392 \csname extras#1\endcsname\relax
1393 \bbl@usehooks{afterextras}{}%
1394 % > babel-ensure
1395 % > babel-sh-<short>
1396 % > babel-bidi
1397 % > babel-fontspec
1398 % hyphenation - case mapping
1399 \ifcase\bbl@opt@hyphenmap\or
1400 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1401 \ifnum\bbl@hymapsel>4\else
1402 \csname\languagenam @bbl@hyphenmap\endcsname
1403 \fi
1404 \chardef\bbl@opt@hyphenmap\z@
1405 \else
1406 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1407 \csname\languagenam @bbl@hyphenmap\endcsname
1408 \fi
1409 \fi
1410 \global\let\bbl@hymapsel\@cclv
1411 % hyphenation - patterns
1412 \bbl@patterns{#1}%
1413 % hyphenation - mins
1414 \babel@savevariable\lefthyphenmin
1415 \babel@savevariable\righthyphenmin
1416 \expandafter\ifx\csname #1hyphenmins\endcsname\relax

```

```

1417 \set@hyphenmins\tw@\thr@@\relax
1418 \else
1419 \expandafter\expandafter\expandafter\set@hyphenmins
1420 \csname #1hyphenmins\endcsname\relax
1421 \fi}

```

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

```

1422 \long\def\otherlanguage#1{%
1423 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1424 \csname selectlanguage\endcsname{#1}%
1425 \ignorespaces}

```

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

```

1426 \long\def\endotherlanguage{%
1427 \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`.

```

1428 \expandafter\def\csname otherlanguage*\endcsname{%
1429 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1430 \def\bbl@otherlanguage@s[#1]#2{%
1431 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1432 \def\bbl@select@opts{#1}%
1433 \foreign@language{#2}}

```

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

```

1434 \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`.

```

1435 \providecommand\bbl@beforeforeign{
1436 \edef\foreignlanguage{%
1437   \noexpand\protect
1438   \expandafter\noexpand\csname foreignlanguage \endcsname}
1439 \expandafter\def\csname foreignlanguage \endcsname{%
1440   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1441 \newcommand\bbl@foreign@x[3][{}]{%
1442   \begingroup
1443     \def\bbl@select@opts{#1}%
1444     \let\BabelText\@firstofone
1445     \bbl@beforeforeign
1446     \foreign@language{#2}%
1447     \bbl@usehooks{foreign}{}%
1448     \BabelText{#3}% Now in horizontal mode!
1449   \endgroup}
1450 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
1451   \begingroup
1452     {\par}%
1453     \let\BabelText\@firstofone
1454     \foreign@language{#1}%
1455     \bbl@usehooks{foreign*}{}%
1456     \bbl@dirparastext
1457     \BabelText{#2}% Still in vertical mode!
1458     {\par}%
1459   \endgroup}

```

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

```

1460 \def\foreign@language#1{%
1461   % set name
1462   \edef\language{#1}%
1463   \ifbbl@usedategroup
1464     \bbl@add\bbl@select@opts{,date,}%
1465     \bbl@usedategroupfalse
1466   \fi
1467   \bbl@fixname\language
1468   % TODO. name@map here?
1469   \bbl@provide@locale
1470   \bbl@iflanguage\language{%
1471     \expandafter\ifx\csname date\language\endcsname\relax
1472       \bbl@warning % TODO - why a warning, not an error?
1473       {Unknown language `#1'. Either you have\\%
1474        misspelled its name, it has not been installed,\\%
1475        or you requested it in a previous run. Fix its name,\\%
1476        install it or just rerun the file, respectively. In\\%
1477        some cases, you may need to remove the aux file.\\%
1478        I'll proceed, but expect wrong results.\\%
1479        Reported}%
1480     \fi
1481     % set type
1482     \let\bbl@select@type\@ne
1483     \expandafter\bbl@switch\expandafter{\language}}

```

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



instead of the default.

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

```

1484 \let\bbl@hyphlist\@empty
1485 \let\bbl@hyphenation@relax
1486 \let\bbl@pttnlist\@empty
1487 \let\bbl@patterns@relax
1488 \let\bbl@hymapsel=\@ccclv
1489 \def\bbl@patterns#1{%
1490   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1491     \csname l@#1\endcsname
1492     \edef\bbl@tempa{#1}%
1493   \else
1494     \csname l@#1:\f@encoding\endcsname
1495     \edef\bbl@tempa{#1:\f@encoding}%
1496   \fi
1497   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1498   % > luatex
1499   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
1500     \begingroup
1501       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1502       \ifin@else
1503         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1504         \hyphenation{%
1505           \bbl@hyphenation@
1506           \@ifundefined{bbl@hyphenation@#1}%
1507             \@empty
1508             {\space\csname bbl@hyphenation@#1\endcsname}}%
1509         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1510       \fi
1511     \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\*.

```

1512 \def\hyphenrules#1{%
1513   \edef\bbl@tempf{#1}%
1514   \bbl@fixname\bbl@tempf
1515   \bbl@iflanguage\bbl@tempf{%
1516     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1517     \languageshortands{none}%
1518     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1519       \set@hyphenmins\tw@\thr@@\relax
1520     \else
1521       \expandafter\expandafter\expandafter\set@hyphenmins
1522       \csname\bbl@tempf hyphenmins\endcsname\relax
1523     \fi}}
1524 \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.

```

1525 \def\providehyphenmins#1#2{%
1526   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1527     \@namedef{#1hyphenmins}{#2}%
1528   \fi}

```

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

```

1529 \def\set@hyphenmins#1#2{%
1530   \lefthyphenmin#1\relax
1531   \righthyphenmin#2\relax}

```

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

```

1532 \ifx\ProvidesFile\@undefined
1533   \def\ProvidesLanguage#1[#2 #3 #4]{%
1534     \wlog{Language: #1 #4 #3 <#2>}%
1535   }
1536 \else
1537   \def\ProvidesLanguage#1{%
1538     \begingroup
1539     \catcode`\ 10 %
1540     \@makeother\/%
1541     \@ifnextchar[%]
1542       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1543   \def\@provideslanguage#1[#2]{%
1544     \wlog{Language: #1 #2}%
1545     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1546     \endgroup}
1547 \fi

```

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

```

1548 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi

```

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

```

1549 \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’:

```

1550 \providecommand\setlocale{%
1551   \bbl@error
1552   {Not yet available}%
1553   {Find an armchair, sit down and wait}}
1554 \let\uselocale\setlocale
1555 \let\locale\setlocale
1556 \let\selectlocale\setlocale
1557 \let\localename\setlocale
1558 \let\textlocale\setlocale
1559 \let\textlanguage\setlocale
1560 \let\languagetext\setlocale

```

## 9.2 Errors

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

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

```

1561 \edef\bbl@nulllanguage{\string\language=0}
1562 \ifx\PackageError\@undefined % TODO. Move to Plain
1563   \def\bbl@error#1#2{%
1564     \begingroup
1565       \newlinechar=`^^J
1566       \def\{^^J(babel) }%
1567       \errhelp{#2}\errmessage{\#1}%
1568     \endgroup}
1569   \def\bbl@warning#1{%
1570     \begingroup
1571       \newlinechar=`^^J
1572       \def\{^^J(babel) }%
1573       \message{\#1}%
1574     \endgroup}
1575   \let\bbl@infowarn\bbl@warning
1576   \def\bbl@info#1{%
1577     \begingroup
1578       \newlinechar=`^^J
1579       \def\{^^J}%
1580       \wlog{#1}%
1581     \endgroup}
1582 \fi
1583 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1584 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1585   \global\@namedef{#2}{\textbf{?#1?}}%
1586   \@nameuse{#2}%
1587   \bbl@warning{%
1588     \@backslashchar#2 not set. Please, define\\%
1589     it in the preamble with something like:\\%
1590     \string\renewcommand\@backslashchar#2{..}\\%
1591     Reported}}
1592 \def\bbl@tentative{\protect\bbl@tentative@i}
1593 \def\bbl@tentative@i#1{%
1594   \bbl@warning{%
1595     Some functions for '#1' are tentative.\\%
1596     They might not work as expected and their behavior\\%
1597     could change in the future.\\%
1598     Reported}}
1599 \def\@nolanerr#1{%
1600   \bbl@error
1601   {You haven't defined the language #1\space yet.\\%
1602     Perhaps you misspelled it or your installation\\%
1603     is not complete}%
1604   {Your command will be ignored, type <return> to proceed}}

```

```

1605 \def\@nopatterns#1{%
1606   \bbl@warning
1607   {No hyphenation patterns were preloaded for\%
1608     the language `#1' into the format.\%
1609     Please, configure your TeX system to add them and\%
1610     rebuild the format. Now I will use the patterns\%
1611     preloaded for \bbl@nulllanguage\space instead}}
1612 \let\bbl@usehooks\@gobbletwo
1613 \ifx\bbl@onlyswitch\@empty\endinput\fi
1614 % Here ended switch.def

Here ended switch.def.

1615 \ifx\directlua\@undefined\else
1616   \ifx\bbl@luapatterns\@undefined
1617     \input luababel.def
1618   \fi
1619 \fi
1620 <<Basic macros>>
1621 \bbl@trace{Compatibility with language.def}
1622 \ifx\bbl@languages\@undefined
1623   \ifx\directlua\@undefined
1624     \openin1 = language.def % TODO. Remove hardcoded number
1625     \ifeof1
1626       \closein1
1627       \message{I couldn't find the file language.def}
1628     \else
1629       \closein1
1630       \begingroup
1631         \def\addlanguage#1#2#3#4#5{%
1632           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1633             \global\expandafter\let\csname l@#1\endcsname
1634               \csname lang@#1\endcsname
1635           \fi}%
1636         \def\uselanguage#1{%
1637           \input language.def
1638         \endgroup
1639       \fi
1640     \fi
1641     \chardef\l@english\z@
1642 \fi

```

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

```

1643 \def\addto#1#2{%
1644   \ifx#1\@undefined
1645     \def#1{#2}%
1646   \else
1647     \ifx#1\relax
1648       \def#1{#2}%
1649     \else
1650       {\toks@\expandafter{#1#2}}%
1651       \xdef#1{\the\toks@}%
1652     \fi
1653   \fi}

```

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

```
1654 \def\bbl@withactive#1#2{%
1655   \begingroup
1656   \lccode`~=`#2\relax
1657   \lowercase{\endgroup#1~}}
```

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

```
1658 \def\bbl@redefine#1{%
1659   \edef\bbl@tempa{\bbl@stripslash#1}%
1660   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1661   \expandafter\def\csname\bbl@tempa\endcsname}
1662 \@onlypreamble\bbl@redefine
```

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

```
1663 \def\bbl@redefine@long#1{%
1664   \edef\bbl@tempa{\bbl@stripslash#1}%
1665   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1666   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1667 \@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_`.

```
1668 \def\bbl@redefineroobust#1{%
1669   \edef\bbl@tempa{\bbl@stripslash#1}%
1670   \bbl@ifunset{\bbl@tempa\space}%
1671   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1672    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1673   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1674   \@namedef{\bbl@tempa\space}
1675 \@onlypreamble\bbl@redefineroobust
```

### 9.3 Hooks

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

```
1676 \bbl@trace{Hooks}
1677 \newcommand\AddBabelHook[3][{}]{%
1678   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1679   \def\bbl@tempa##1,##3=\empty{\def\bbl@tempb{##2}}%
1680   \expandafter\bbl@tempa\bbl@evargs,##3=\empty
1681   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1682   {\bbl@csarg\bbl@add{\ev@#3@#1}{\bbl@elt{#2}}}%
1683   {\bbl@csarg\let{\ev@#2@#3@#1}\relax}%
1684   \bbl@csarg\newcommand{\ev@#2@#3@#1}{\bbl@tempb}}
```

```

1685 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1686 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1687 \def\bbl@usehooks#1#2{%
1688   \def\bbl@elt##1{%
1689     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1}#2}}%
1690   \bbl@cs{ev@#1@}%
1691   \ifx\language\@undefined\else % Test required for Plain (?)
1692     \def\bbl@elt##1{%
1693       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1694     \bbl@cl{ev@#1}%
1695   \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).

```

1696 \def\bbl@evargs{% <- don't delete this comma
1697   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1698   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1699   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1700   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1701   beforestart=0,language=2}

```

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

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

```

1702 \bbl@trace{Defining babelensure}
1703 \newcommand\babelensure[2][{}]{% TODO - revise test files
1704   \AddBabelHook{babel-ensure}{afterextras}{%
1705     \ifcase\bbl@select@type
1706       \bbl@cl{e}%
1707     \fi}%
1708   \begingroup
1709     \let\bbl@ens@include\@empty
1710     \let\bbl@ens@exclude\@empty
1711     \def\bbl@ens@fontenc{\relax}%
1712     \def\bbl@tempb##1{%
1713       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1714     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1715     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1716     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1717     \def\bbl@tempc{\bbl@ensure}%
1718     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1719       \expandafter{\bbl@ens@include}}%
1720     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1721       \expandafter{\bbl@ens@exclude}}%
1722     \toks@{\expandafter{\bbl@tempc}}%
1723     \bbl@exp{%
1724   \endgroup
1725   \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1726 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc

```

```

1727 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1728 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1729 \edef##1{\noexpand\bbl@nocaption
1730 {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1731 \fi
1732 \ifx##1\@empty\else
1733 \in@{##1}{#2}%
1734 \ifin\else
1735 \bbl@ifunset{\bbl@ensure@\language}%
1736 {\bbl@exp{%
1737 \\\DeclareRobustCommand\bbl@ensure@<\language>[1]{%
1738 \\\foreignlanguage{\language}%
1739 {\ifx\relax#3\else
1740 \\\fontencoding{#3}\selectfont
1741 \fi
1742 #####1}}}%
1743 }%
1744 \toks@\expandafter{##1}%
1745 \edef##1{%
1746 \bbl@csarg\noexpand{ensure@\language}%
1747 {\the\toks@}}%
1748 \fi
1749 \expandafter\bbl@tempb
1750 \fi}%
1751 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1752 \def\bbl@tempa##1{% elt for include list
1753 \ifx##1\@empty\else
1754 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1755 \ifin\else
1756 \bbl@tempb##1\@empty
1757 \fi
1758 \expandafter\bbl@tempa
1759 \fi}%
1760 \bbl@tempa#1\@empty}
1761 \def\bbl@captionslist{%
1762 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1763 \contentsname\listfigurename\listtablename\indexname\figurename
1764 \tablename\partname\encname\ccname\headtoname\pagename\seename
1765 \alsoname\proofname\glossaryname}

```

## 9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1766 \bbl@trace{Macros for setting language files up}
1767 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1768   \let\bbl@screset\@empty
1769   \let\BabelStrings\bbl@opt@string
1770   \let\BabelOptions\@empty
1771   \let\BabelLanguages\relax
1772   \ifx\originalTeX\@undefined
1773     \let\originalTeX\@empty
1774   \else
1775     \originalTeX
1776   \fi}
1777 \def\LdfInit#1#2{%
1778   \chardef\atcatcode=\catcode`\@
1779   \catcode`\@=11\relax
1780   \chardef\eqcatcode=\catcode`\=
1781   \catcode`\==12\relax
1782   \expandafter\if\expandafter\@backslashchar
1783     \expandafter\@car\string#2\@nil
1784     \ifx#2\@undefined\else
1785       \ldf@quit{#1}%
1786     \fi
1787   \else
1788     \expandafter\ifx\csname#2\endcsname\relax\else
1789       \ldf@quit{#1}%
1790     \fi
1791   \fi
1792   \bbl@ldfinit}

```

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

```

1793 \def\ldf@quit#1{%
1794   \expandafter\main@language\expandafter{#1}%
1795   \catcode`\@=\atcatcode \let\atcatcode\relax
1796   \catcode`\==\eqcatcode \let\eqcatcode\relax
1797   \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.

```

1798 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1799   \bbl@afterlang
1800   \let\bbl@afterlang\relax
1801   \let\BabelModifiers\relax
1802   \let\bbl@screset\relax}%
1803 \def\ldf@finish#1{%
1804   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1805     \loadlocalcfg{#1}%
1806   \fi
1807   \bbl@afterldf{#1}%
1808   \expandafter\main@language\expandafter{#1}%
1809   \catcode`\@=\atcatcode \let\atcatcode\relax
1810   \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  $\LaTeX$ .

```
1811 \@onlypreamble\LdfInit
1812 \@onlypreamble\ldf@quit
1813 \@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.

```
1814 \def\main@language#1{%
1815   \def\bbl@main@language{#1}%
1816   \let\language\main@language % TODO. Set localename
1817   \bbl@id@assign
1818   \bbl@patterns{\language}}
```

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

```
1819 \def\bbl@beforestart{%
1820   \bbl@usehooks{beforestart}{}%
1821   \global\let\bbl@beforestart\relax}
1822 \AtBeginDocument{%
1823   \@nameuse{bbl@beforestart}%
1824   \if@filesw
1825     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1826   \fi
1827   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1828   \ifbbl@single % must go after the line above.
1829     \renewcommand\selectlanguage[1]{}%
1830     \renewcommand\foreignlanguage[2]{#2}%
1831     \global\let\babel@aux\@gobbletwo % Also as flag
1832   \fi
1833   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1834 \def\select@language@x#1{%
1835   \ifcase\bbl@select@type
1836     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1837   \else
1838     \select@language{#1}%
1839   \fi}
```

## 9.5 Shorthands

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

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

```
1840 \bbl@trace{Shorhands}
1841 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1842   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1843   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
```

```

1844 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1845 \beginingroup
1846 \catcode`#1\active
1847 \nfss@catcodes
1848 \ifnum\catcode`#1=\active
1849 \endgroup
1850 \bbl@add\nfss@catcodes{\@makeother#1}%
1851 \else
1852 \endgroup
1853 \fi
1854 \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.

```

1855 \def\bbl@remove@special#1{%
1856 \beginingroup
1857 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1858 \else\noexpand##1\noexpand##2\fi}%
1859 \def\do{\x\do}%
1860 \def\@makeother{\x\@makeother}%
1861 \edef\x{\endgroup
1862 \def\noexpand\dospecials{\dospecials}%
1863 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1864 \def\noexpand\@sanitize{\@sanitize}%
1865 \fi}%
1866 \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).

```

1867 \def\bbl@active@def#1#2#3#4{%
1868 \@namedef{#3#1}{%
1869 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1870 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1871 \else
1872 \bbl@afterfi\csname#2@sh@#1\endcsname
1873 \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.

```

1874 \long\@namedef{#3@arg#1}##1{%
1875   \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1876     \bbl@afterelse\csname#4#1\endcsname##1%
1877   \else
1878     \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1879   \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.

```

1880 \def\initiate@active@char#1{%
1881   \bbl@ifunset{active@char\string#1}%
1882   {\bbl@withactive
1883     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1884   {}}

```

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).

```

1885 \def\@initiate@active@char#1#2#3{%
1886   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1887   \ifx#1\@undefined
1888     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1889   \else
1890     \bbl@csarg\let{oridef@#2}#1%
1891     \bbl@csarg\edef{oridef@#2}{%
1892       \let\noexpand#1%
1893       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1894   \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*").

```

1895   \ifx#1#3\relax
1896     \expandafter\let\csname normal@char#2\endcsname#3%
1897   \else
1898     \bbl@info{Making #2 an active character}%
1899     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1900     \@namedef{normal@char#2}{%
1901       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1902   \else
1903     \@namedef{normal@char#2}{#3}%
1904   \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).

```

1905   \bbl@restoreactive{#2}%
1906   \AtBeginDocument{%
1907     \catcode`#2\active
1908     \if@files

```

```

1909      \immediate\write\@mainaux{\catcode`\string#2\active}%
1910      \fi}%
1911      \expandafter\bb1@add@special\csname#2\endcsname
1912      \catcode`\#2\active
1913      \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⟩`).

```

1914      \let\bb1@tempa\@firstoftwo
1915      \if\string^#2%
1916        \def\bb1@tempa{\noexpand\textormath}%
1917      \else
1918        \ifx\bb1@mathnormal\@undefined\else
1919          \let\bb1@tempa\bb1@mathnormal
1920        \fi
1921      \fi
1922      \expandafter\edef\csname active@char#2\endcsname{%
1923        \bb1@tempa
1924        {\noexpand\if@safe@actives
1925          \noexpand\expandafter
1926          \expandafter\noexpand\csname normal@char#2\endcsname
1927        \noexpand\else
1928          \noexpand\expandafter
1929          \expandafter\noexpand\csname bbl@doactive#2\endcsname
1930        \noexpand\fi}%
1931      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1932      \bb1@csarg\edef{doactive#2}{%
1933        \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!).

```

1934      \bb1@csarg\edef{active@#2}{%
1935        \noexpand\active@prefix\noexpand#1%
1936        \expandafter\noexpand\csname active@char#2\endcsname}%
1937      \bb1@csarg\edef{normal@#2}{%
1938        \noexpand\active@prefix\noexpand#1%
1939        \expandafter\noexpand\csname normal@char#2\endcsname}%
1940      \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.

```

1941      \bb1@active@def#2\user@group{user@active}{language@active}%
1942      \bb1@active@def#2\language@group{language@active}{system@active}%
1943      \bb1@active@def#2\system@group{system@active}{normal@char}%

```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as ‘ ’ ends up in a heading T<sub>E</sub>X would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```

1944      \expandafter\edef\csname\user@group @sh@#2@@\endcsname

```

```

1945 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1946 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1947 {\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.

```

1948 \if\string'#2%
1949 \let\prim@s\bbl@prim@s
1950 \let\active@math@prime#1%
1951 \fi
1952 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

1953 <<(*More package options)>> ≡
1954 \DeclareOption{math=active}{}
1955 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1956 <</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*.

```

1957 \@ifpackagewith{babel}{KeepShorthandsActive}%
1958 {\let\bbl@restoreactive\@gobble}%
1959 {\def\bbl@restoreactive#1{%
1960 \bbl@exp{%
1961 \\\AfterBabelLanguage\\CurrentOption
1962 {\catcode`#1=\the\catcode`#1\relax}%
1963 \\\AtEndOfPackage
1964 {\catcode`#1=\the\catcode`#1\relax}}}%
1965 \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.

```

1966 \def\bbl@sh@select#1#2{%
1967 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1968 \bbl@afterelse\bbl@scndcs
1969 \else
1970 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1971 \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.

```

1972 \begingroup
1973 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1974 {\gdef\active@prefix#1{%
1975 \ifx\protect\@typeset@protect

```

```

1976     \else
1977         \ifx\protect\@unexpandable@protect
1978             \noexpand#1%
1979         \else
1980             \protect#1%
1981         \fi
1982         \expandafter\@gobble
1983     \fi}}
1984 {\gdef\active@prefix#1{%
1985     \ifincsname
1986         \string#1%
1987         \expandafter\@gobble
1988     \else
1989         \ifx\protect\@typeset@protect
1990         \else
1991             \ifx\protect\@unexpandable@protect
1992                 \noexpand#1%
1993             \else
1994                 \protect#1%
1995             \fi
1996             \expandafter\expandafter\expandafter\@gobble
1997         \fi
1998     \fi}}
1999 \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⟩`.

```

2000 \newif\if@safe@actives
2001 \@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.

```

2002 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

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

```

2003 \def\bbl@activate#1{%
2004     \bbl@withactive{\expandafter\let\expandafter}#1%
2005     \csname bbl@active@\string#1\endcsname}
2006 \def\bbl@deactivate#1{%
2007     \bbl@withactive{\expandafter\let\expandafter}#1%
2008     \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2009 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2010 \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.

```

2011 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2012 \def\@decl@short#1#2#3\@nil#4{%
2013   \def\bbl@tempa{#3}%
2014   \ifx\bbl@tempa\@empty
2015     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2016     \bbl@ifunset{#1@sh@\string#2@}{}%
2017     {\def\bbl@tempa{#4}%
2018       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2019       \else
2020         \bbl@info
2021           {Redefining #1 shorthand \string#2\%
2022             in language \CurrentOption}%
2023         \fi}%
2024     \@namedef{#1@sh@\string#2@}{#4}%
2025   \else
2026     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2027     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2028     {\def\bbl@tempa{#4}%
2029       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2030       \else
2031         \bbl@info
2032           {Redefining #1 shorthand \string#2\string#3\%
2033             in language \CurrentOption}%
2034         \fi}%
2035     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2036   \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.

```

2037 \def\textormath{%
2038   \ifmmode
2039     \expandafter\@secondoftwo
2040   \else
2041     \expandafter\@firstoftwo
2042   \fi}

```

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

```

2043 \def\user@group{user}
2044 \def\language@group{english} % TODO. I don't like defaults
2045 \def\system@group{system}

```

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

```

2046 \def\useshorthands{%
2047   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2048 \def\bbl@usesh@s#1{%
2049   \bbl@usesh@x
2050   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2051   {#1}}
2052 \def\bbl@usesh@x#1#2{%
2053   \bbl@ifshorthand{#2}%
2054   {\def\user@group{user}%

```

```

2055 \initiate@active@char{#2}%
2056 #1%
2057 \bbl@activate{#2}}%
2058 {\bbl@error
2059 {Cannot declare a shorthand turned off (\string#2)}
2060 {Sorry, but you cannot use shorthands which have been\\%
2061 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.

```

2062 \def\user@language@group{user@\language@group}
2063 \def\bbl@set@user@generic#1#2{%
2064 \bbl@ifunset{user@generic@active#1}%
2065 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2066 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2067 \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2068 \expandafter\noexpand\csname normal@char#1\endcsname}%
2069 \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
2070 \expandafter\noexpand\csname user@active#1\endcsname}}%
2071 \@empty}
2072 \newcommand\defineshorthand[3][user]{%
2073 \edef\bbl@tempa{\zap@space#1 \@empty}%
2074 \bbl@for\bbl@tempb\bbl@tempa{%
2075 \if*\expandafter\@car\bbl@tempb\@nil
2076 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2077 \@expandtwoargs
2078 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2079 \fi
2080 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```

2081 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

2082 \def\aliasshorthand#1#2{%
2083 \bbl@ifshorthand{#2}%
2084 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2085 \ifx\document\@notprerr
2086 \@notshorthand{#2}%
2087 \else
2088 \initiate@active@char{#2}%
2089 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2090 \csname active@char\string#1\endcsname
2091 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2092 \csname normal@char\string#1\endcsname
2093 \bbl@activate{#2}%
2094 \fi
2095 \fi}%
2096 {\bbl@error
2097 {Cannot declare a shorthand turned off (\string#2)}

```



```

2098      {Sorry, but you cannot use shorthands which have been\\%
2099      turned off in the package options}}

\@notshorthand
2100 \def\@notshorthand#1{%
2101   \bbl@error{%
2102     The character '\string #1' should be made a shorthand character;\\%
2103     add the command \string\usesshorthands\string{#1\string} to
2104     the preamble.\\%
2105     I will ignore your instruction}%
2106   {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.

2107 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2108 \DeclareRobustCommand*\shorthandoff{%
2109   \@ifstar{\bbl@shorthandoff\tw}{\bbl@shorthandoff\z@}}
2110 \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.

2111 \def\bbl@switch@sh#1#2{%
2112   \ifx#2\@nnil\else
2113     \bbl@ifunset{\bbl@active@\string#2}%
2114     {\bbl@error
2115       {I cannot switch '\string#2' on or off--not a shorthand}%
2116       {This character is not a shorthand. Maybe you made\\%
2117         a typing mistake? I will ignore your instruction}}%
2118     {\ifcase#1%
2119       \catcode`#212\relax
2120       \or
2121       \catcode`#2\active
2122       \or
2123       \csname bbl@oricat@\string#2\endcsname
2124       \csname bbl@oridef@\string#2\endcsname
2125       \fi}%
2126     \bbl@afterfi\bbl@switch@sh#1%
2127   \fi}

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

2128 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2129 \def\bbl@putsh#1{%
2130   \bbl@ifunset{\bbl@active@\string#1}%
2131   {\bbl@putsh@i#1\@empty\@nnil}%
2132   {\csname bbl@active@\string#1\endcsname}}
2133 \def\bbl@putsh@i#1#2\@nnil{%
2134   \csname\languagename @sh@\string#1@%
2135     \ifx\@empty#2\else\string#2@\fi\endcsname}
2136 \ifx\bbl@opt@shorthands\@nnil\else

```

```

2137 \let\bbl@s@initiate@active@char\initiate@active@char
2138 \def\initiate@active@char#1{%
2139   \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2140 \let\bbl@s@switch@sh\bbl@switch@sh
2141 \def\bbl@switch@sh#1#2{%
2142   \ifx#2\@nnil\else
2143     \bbl@afterfi
2144     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2145   \fi}
2146 \let\bbl@s@activate\bbl@activate
2147 \def\bbl@activate#1{%
2148   \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2149 \let\bbl@s@deactivate\bbl@deactivate
2150 \def\bbl@deactivate#1{%
2151   \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2152 \fi

```

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

```

2153 \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.

```

2154 \def\bbl@prim@s{%
2155   \prime\futurelet\@let@token\bbl@pr@m@s}
2156 \def\bbl@if@primes#1#2{%
2157   \ifx#1\@let@token
2158     \expandafter\@firstoftwo
2159   \else\ifx#2\@let@token
2160     \bbl@afterelse\expandafter\@firstoftwo
2161   \else
2162     \bbl@afterfi\expandafter\@secondoftwo
2163   \fi\fi}
2164 \begingroup
2165 \catcode`\^=7 \catcode`\*= \active \lccode`\^=\^
2166 \catcode`\'=12 \catcode`\="= \active \lccode`\="=\ '
2167 \lowercase{%
2168   \gdef\bbl@pr@m@s{%
2169     \bbl@if@primes"%
2170       \pr@@s
2171       {\bbl@if@primes*\pr@@@t\egroup}}
2172 \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).

```

2173 \initiate@active@char{~}
2174 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2175 \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.

```

2176 \expandafter\def\csname OT1dqqpos\endcsname{127}
2177 \expandafter\def\csname T1dqqpos\endcsname{4}

```

When the macro `\f@encoding` is undefined (as it is in plain  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ ) we define it here to expand to OT1

```

2178 \ifx\f@encoding\undefined
2179   \def\f@encoding{OT1}
2180 \fi

```

## 9.6 Language attributes

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

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

```

2181 \bbl@trace{Language attributes}
2182 \newcommand\languageattribute[2]{%
2183   \def\bbl@tempc{#1}%
2184   \bbl@fixname\bbl@tempc
2185   \bbl@iflanguage\bbl@tempc{%
2186     \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.

```

2187     \ifx\bbl@known@attribs\undefined
2188       \in@false
2189     \else
2190       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2191     \fi
2192     \ifin@
2193       \bbl@warning{%
2194         You have more than once selected the attribute '##1'\%
2195         for language #1. Reported}%
2196     \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  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ -code.

```

2197       \bbl@exp{%
2198         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2199       \edef\bbl@tempa{\bbl@tempc-##1}%
2200       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2201       {\csname\bbl@tempc @attr##1\endcsname}%
2202       {\@attrerr{\bbl@tempc}{##1}}%
2203     \fi}}}
2204 \@onlypreamble\languageattribute

```

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

```

2205 \newcommand*{\@attrerr}[2]{%
2206   \bbl@error
2207   {The attribute #2 is unknown for language #1.}%
2208   {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}`.

```
2209 \def\bbl@declare@ttribute#1#2#3{%
2210   \bbl@xin@{,#2,},{,\BabelModifiers,}%
2211   \ifin@
2212     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2213   \fi
2214   \bbl@add@list\bbl@attributes{#1-#2}%
2215   \expandafter\def\csname#1@attr#2\endcsname{#3}}
```

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

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

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

```
2216 \def\bbl@ifattributeset#1#2#3#4{%
2217   \ifx\bbl@known@attribs\undefined
2218     \in@false
2219   \else
2220     \bbl@xin@{,#1-#2,},{,\bbl@known@attribs,}%
2221   \fi
2222   \ifin@
2223     \bbl@afterelse#3%
2224   \else
2225     \bbl@afterfi#4%
2226   \fi
2227 }
```

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

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

```
2228 \def\bbl@ifknown@ttrib#1#2{%
2229   \let\bbl@tempa\@secondoftwo
2230   \bbl@loopx\bbl@tempb{#2}{%
2231     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2232     \ifin@
2233       \let\bbl@tempa\@firstoftwo
2234     \else
2235       \fi}%
2236   \bbl@tempa
2237 }
```

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

```
2238 \def\bbl@clear@ttribs{%
2239   \ifx\bbl@attributes\undefined\else
```

```

2240 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2241 \expandafter\bbl@clear@ttrib\bbl@tempa.
2242 }%
2243 \let\bbl@attributes\@undefined
2244 \fi}
2245 \def\bbl@clear@ttrib#1-#2.{%
2246 \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2247 \AtBeginDocument{\bbl@clear@ttribs}

```

## 9.7 Support for saving macro definitions

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

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

```

\babel@beginsave 2248 \bbl@trace{Macros for saving definitions}
2249 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2250 \newcount\babel@savecnt
2251 \babel@beginsave

```

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

```

2252 \def\babel@save#1{%
2253 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2254 \toks@\expandafter{\originalTeX\let#1=}
2255 \bbl@exp{%
2256 \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}
2257 \advance\babel@savecnt@ne}
2258 \def\babel@savevariable#1{%
2259 \toks@\expandafter{\originalTeX #1=}
2260 \bbl@exp{\def\originalTeX{\the\toks@\the#1\relax}}}

```

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

```

2261 \def\bbl@frenchspacing{%
2262 \ifnum\the\sfcode`.=\@m
2263 \let\bbl@nonfrenchspacing\relax
2264 \else
2265 \frenchspacing
2266 \let\bbl@nonfrenchspacing\nonfrenchspacing
2267 \fi}
2268 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

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

## 9.8 Short tags

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

```

2269 \bbl@trace{Short tags}
2270 \def\babeltags#1{%
2271   \edef\bbl@tempa{\zap@space#1 \@empty}%
2272   \def\bbl@tempb##1=##2\@{#1}%
2273   \edef\bbl@tempc{%
2274     \noexpand\newcommand
2275     \expandafter\noexpand\csname ##1\endcsname{%
2276       \noexpand\protect
2277       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2278     \noexpand\newcommand
2279     \expandafter\noexpand\csname text##1\endcsname{%
2280       \noexpand\foreignlanguage{##2}}
2281   \bbl@tempc}%
2282   \bbl@for\bbl@tempa\bbl@tempa{%
2283     \expandafter\bbl@tempb\bbl@tempa\@{}}

```

## 9.9 Hyphens

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

```

2284 \bbl@trace{Hyphens}
2285 \@onlypreamble\babelhyphenation
2286 \AtEndOfPackage{%
2287   \newcommand\babelhyphenation[2][\@empty]{%
2288     \ifx\bbl@hyphenation@ \relax
2289       \let\bbl@hyphenation@ \@empty
2290     \fi
2291     \ifx\bbl@hyphlist \@empty \else
2292       \bbl@warning{%
2293         You must not intermingle \string\selectlanguage\space and\%
2294         \string\babelhyphenation\space or some exceptions will not\%
2295         be taken into account. Reported}%
2296     \fi
2297     \ifx\@empty#1%
2298       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2299     \else
2300       \bbl@vforeach{#1}{%
2301         \def\bbl@tempa{##1}%
2302         \bbl@fixname\bbl@tempa
2303         \bbl@iflanguage\bbl@tempa{%
2304           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2305             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2306             \@empty
2307             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2308             #2}}}%
2309       \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt32`.

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

```

2310 \def\bbbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2311 \def\bbbl@t@one{T1}
2312 \def\allowhyphens{\ifx\cf@encoding\bbbl@t@one\else\bbbl@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.

```

2313 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2314 \def\babelhyphen{\active@prefix\babelhyphen\bbbl@hyphen}
2315 \def\bbbl@hyphen{%
2316   \@ifstar{\bbbl@hyphen@i @}{\bbbl@hyphen@i \@empty}}
2317 \def\bbbl@hyphen@i#1#2{%
2318   \bbbl@ifunset{\bbbl@hy@#1#2\@empty}%
2319   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{\#2}}}%
2320   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

```

2321 \def\bbbl@usehyphen#1{%
2322   \leavevmode
2323   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2324   \nobreak\hskip\z@skip}
2325 \def\bbbl@usehyphen#1{%
2326   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2327 \def\bbbl@hyphenchar{%
2328   \ifnum\hyphenchar\font=\m@ne
2329     \babelnullhyphen
2330   \else
2331     \char\hyphenchar\font
2332   \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 \bbbl@hy@nobreak is redundant.

```

2333 \def\bbbl@hy@soft{\bbbl@usehyphen{\discretionary{\bbbl@hyphenchar}{\#2}}{}}
2334 \def\bbbl@hy@@soft{\bbbl@usehyphen{\discretionary{\bbbl@hyphenchar}{\#2}}{}}
2335 \def\bbbl@hy@hard{\bbbl@usehyphen\bbbl@hyphenchar}
2336 \def\bbbl@hy@@hard{\bbbl@usehyphen\bbbl@hyphenchar}
2337 \def\bbbl@hy@nobreak{\bbbl@usehyphen{\mbox{\bbbl@hyphenchar}}}
2338 \def\bbbl@hy@@nobreak{\mbox{\bbbl@hyphenchar}}
2339 \def\bbbl@hy@repeat{%
2340   \bbbl@usehyphen{%
2341     \discretionary{\bbbl@hyphenchar}{\bbbl@hyphenchar}{\bbbl@hyphenchar}}}
2342 \def\bbbl@hy@@repeat{%
2343   \bbbl@usehyphen{%
2344     \discretionary{\bbbl@hyphenchar}{\bbbl@hyphenchar}{\bbbl@hyphenchar}}}
2345 \def\bbbl@hy@empty{\hskip\z@skip}
2346 \def\bbbl@hy@@empty{\discretionary{\#2}{\#1}\bbbl@allowhyphens}

```

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

```

2347 \def\bbbl@disc#1#2{\nobreak\discretionary{#2-}{\#1}\bbbl@allowhyphens}

```

## 9.10 Multiencoding strings

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

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

```
2348 \bbl@trace{Multiencoding strings}
2349 \def\bbl@tglobal#1{\global\let#1#1}
2350 \def\bbl@recatcode#1{% TODO. Used only once?
2351   \@tempcnta="7F
2352   \def\bbl@tempa{%
2353     \ifnum\@tempcnta>"FF\else
2354       \catcode\@tempcnta=#1\relax
2355       \advance\@tempcnta\@ne
2356       \expandafter\bbl@tempa
2357     \fi}%
2358   \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).

```
2359 \@ifpackagewith{babel}{nocase}%
2360   {\let\bbl@patchuclc\relax}%
2361   {\def\bbl@patchuclc{%
2362     \global\let\bbl@patchuclc\relax
2363     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2364     \gdef\bbl@uclc##1{%
2365       \let\bbl@encoded\bbl@encoded@uclc
2366       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2367       {##1}%
2368       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2369         \csname\language @bbl@uclc\endcsname}%
2370       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2371     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2372     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2373 <<(*More package options)>> ≡
2374 \DeclareOption{nocase}{}
2375 <</More package options>>
```

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

```
2376 <<(*More package options)>> ≡
2377 \let\bbl@opt@strings\@nnil % accept strings=value
2378 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2379 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2380 \def\BabelStringsDefault{generic}
2381 <</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.

```

2382 \@onlypreamble\StartBabelCommands
2383 \def\StartBabelCommands{%
2384   \begingroup
2385   \bbl@recatcode{11}%
2386   <\Macros local to BabelCommands>
2387   \def\bbl@provstring##1##2{%
2388     \providecommand##1{##2}%
2389     \bbl@tglobal##1}%
2390   \global\let\bbl@scafter\@empty
2391   \let\StartBabelCommands\bbl@startcmds
2392   \ifx\BabelLanguages\relax
2393     \let\BabelLanguages\CurrentOption
2394   \fi
2395   \begingroup
2396   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2397   \StartBabelCommands}
2398 \def\bbl@startcmds{%
2399   \ifx\bbl@screset\@nnil\else
2400     \bbl@usehooks{stopcommands}{}%
2401   \fi
2402   \endgroup
2403   \begingroup
2404   \@ifstar
2405     {\ifx\bbl@opt@strings\@nnil
2406       \let\bbl@opt@strings\BabelStringsDefault
2407     \fi
2408     \bbl@startcmds@i}%
2409   \bbl@startcmds@i}
2410 \def\bbl@startcmds@i#1#2{%
2411   \edef\bbl@L{\zap@space#1 \@empty}%
2412   \edef\bbl@G{\zap@space#2 \@empty}%
2413   \bbl@startcmds@ii}
2414 \let\bbl@startcmds\StartBabelCommands

```

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

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

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

```

2415 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2416   \let\SetString@gobbletwo
2417   \let\bbl@stringdef@gobbletwo
2418   \let\AfterBabelCommands@gobble
2419   \ifx\@empty#1%
2420     \def\bbl@sc@label{generic}%
2421     \def\bbl@encstring##1##2{%
2422       \ProvideTextCommandDefault##1{##2}%
2423       \bbl@tglobal##1%
2424       \expandafter\bbl@tglobal\csname\string?string##1\endcsname}%
2425     \let\bbl@sctest\in@true

```

```

2426 \else
2427 \let\bbl@sc@charset\space % <- zapped below
2428 \let\bbl@sc@fontenc\space % <- " "
2429 \def\bbl@tempa##1=##2\@nil{%
2430 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2431 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2432 \def\bbl@tempa##1 ##2{% space -> comma
2433 ##1%
2434 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2435 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2436 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2437 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2438 \def\bbl@encstring##1##2{%
2439 \bbl@foreach\bbl@sc@fontenc{%
2440 \bbl@ifunset{T@###1}%
2441 {}%
2442 {\ProvideTextCommand##1{###1}{##2}%
2443 \bbl@tglobal##1%
2444 \expandafter
2445 \bbl@tglobal\csname###1\string##1\endcsname}}}%
2446 \def\bbl@sctest{%
2447 \bbl@xin@{\,\bbl@opt@strings,}{\,\bbl@sc@label,\bbl@sc@fontenc,}}%
2448 \fi
2449 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2450 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2451 \let\AfterBabelCommands\bbl@aftercmds
2452 \let\SetString\bbl@setstring
2453 \let\bbl@stringdef\bbl@encstring
2454 \else % ie, strings=value
2455 \bbl@sctest
2456 \ifin@
2457 \let\AfterBabelCommands\bbl@aftercmds
2458 \let\SetString\bbl@setstring
2459 \let\bbl@stringdef\bbl@provstring
2460 \fi\fi\fi
2461 \bbl@scswitch
2462 \ifx\bbl@G\@empty
2463 \def\SetString##1##2{%
2464 \bbl@error{Missing group for string \string##1}%
2465 {You must assign strings to some category, typically\\
2466 captions or extras, but you set none}}%
2467 \fi
2468 \ifx\@empty#1%
2469 \bbl@usehooks{defaultcommands}{}%
2470 \else
2471 \@expandtwoargs
2472 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2473 \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).

```

2474 \def\bbl@forlang#1#2{%
2475   \bbl@for#1\bbl@L{%
2476     \bbl@xin@{,#1,},{, \BabelLanguages,}%
2477     \ifin@#2\relax\fi}}
2478 \def\bbl@scswitch{%
2479   \bbl@forlang\bbl@tempa{%
2480     \ifx\bbl@G\@empty\else
2481       \ifx\SetString\@gobbleset\else
2482         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2483         \bbl@xin@{,\bbl@GL,},{,\bbl@screset,}%
2484         \ifin@else
2485           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2486           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2487         \fi
2488       \fi
2489     \fi}}
2490 \AtEndOfPackage{%
2491   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
2492   \let\bbl@scswitch\relax}
2493 \@onlypreamble\EndBabelCommands
2494 \def\EndBabelCommands{%
2495   \bbl@usehooks{stopcommands}{}%
2496   \endgroup
2497   \endgroup
2498   \bbl@scafter}
2499 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

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

```

2500 \def\bbl@setstring#1#2{%
2501   \bbl@forlang\bbl@tempa{%
2502     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2503     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2504     {\global\expandafter % TODO - con \bbl@exp ?
2505       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2506       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
2507     {}}%
2508   \def\BabelString{#2}%
2509   \bbl@usehooks{stringprocess}{}%
2510   \expandafter\bbl@stringdef
2511   \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include \bbl@encoded for string to be expanded in case transformations. It is \relax by default, but in \MakeUppercase and \MakeLowercase its value is a modified expandable \@changed@cmd.

```

2512 \ifx\bbl@opt@strings\relax
2513   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2514   \bbl@patchuclc
2515   \let\bbl@encoded\relax
2516   \def\bbl@encoded@uclc#1{%
2517     \@inmathwarn#1%
2518     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax

```

```

2519     \expandafter\ifx\csname ?\string#1\endcsname\relax
2520     \TextSymbolUnavailable#1%
2521     \else
2522     \csname ?\string#1\endcsname
2523     \fi
2524     \else
2525     \csname\cf@encoding\string#1\endcsname
2526     \fi}
2527 \else
2528 \def\bbl@scset#1#2{\def#1{#2}}
2529 \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.

```

2530 <<*Macros local to BabelCommands>> ≡
2531 \def\SetStringLoop##1##2{%
2532     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2533     \count@\z@
2534     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2535         \advance\count@\@ne
2536         \toks@\expandafter{\bbl@tempa}%
2537         \bbl@exp{%
2538             \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2539             \count@=\the\count@\relax}}}%
2540 <</Macros local to BabelCommands>>

```

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

```

2541 \def\bbl@aftercmds#1{%
2542     \toks@\expandafter{\bbl@scafter#1}%
2543     \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.

```

2544 <<*Macros local to BabelCommands>> ≡
2545 \newcommand\SetCase[3][{%
2546     \bbl@patchuclc
2547     \bbl@forlang\bbl@tempa{%
2548         \expandafter\bbl@encstring
2549         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2550         \expandafter\bbl@encstring
2551         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2552         \expandafter\bbl@encstring
2553         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2554 <</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.

```

2555 <<*Macros local to BabelCommands>> ≡
2556 \newcommand\SetHyphenMap[1]{%
2557     \bbl@forlang\bbl@tempa{%
2558         \expandafter\bbl@stringdef
2559         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2560 <</Macros local to BabelCommands>>

```

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

```

2561 \newcommand\BabelLower[2]{% one to one.
2562   \ifnum\lccode#1=#2\else
2563     \babel@savevariable{\lccode#1}%
2564     \lccode#1=#2\relax
2565   \fi}
2566 \newcommand\BabelLowerMM[4]{% many-to-many
2567   \@tempcnta=#1\relax
2568   \@tempcntb=#4\relax
2569   \def\bbl@tempa{%
2570     \ifnum\@tempcnta>#2\else
2571       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2572       \advance\@tempcnta#3\relax
2573       \advance\@tempcntb#3\relax
2574       \expandafter\bbl@tempa
2575     \fi}%
2576   \bbl@tempa}
2577 \newcommand\BabelLowerMO[4]{% many-to-one
2578   \@tempcnta=#1\relax
2579   \def\bbl@tempa{%
2580     \ifnum\@tempcnta>#2\else
2581       \@expandtwoargs\BabelLower{\the\@tempcnta}{\#4}%
2582       \advance\@tempcnta#3
2583       \expandafter\bbl@tempa
2584     \fi}%
2585   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2586 <<(*More package options)>> ≡
2587 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2588 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2589 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2590 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2591 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2592 <</More package options>>

```

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

```

2593 \AtEndOfPackage{%
2594   \ifx\bbl@opt@hyphenmap\undefined
2595     \bbl@xin@{,}{\bbl@language@opts}%
2596     \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
2597   \fi}

```

## 9.11 Macros common to a number of languages

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

```

2598 \bbl@trace{Macros related to glyphs}
2599 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z\hbox{#1}%
2600   \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2601   \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.

```

2602 \def\save@sf@q#1{\leavevmode
2603   \begingroup
2604     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2605   \endgroup}

```

## 9.12 Making glyphs available

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

### 9.12.1 Quotation marks

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

```
2606 \ProvideTextCommand{\quotedblbase}{OT1}{%
2607   \save@sf@q{\set@low@box{\textquotedblright\}%
2608     \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2609 \ProvideTextCommandDefault{\quotedblbase}{%
2610   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2611 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2612   \save@sf@q{\set@low@box{\textquoteright\}%
2613     \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2614 \ProvideTextCommandDefault{\quotesinglbase}{%
2615   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2616 \ProvideTextCommand{\guillemetleft}{OT1}{%
2617   \ifmmode
2618     \ll
2619   \else
2620     \save@sf@q{\nobreak
2621       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2622   \fi}
2623 \ProvideTextCommand{\guillemetright}{OT1}{%
2624   \ifmmode
2625     \gg
2626   \else
2627     \save@sf@q{\nobreak
2628       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2629   \fi}
2630 \ProvideTextCommand{\guillemotleft}{OT1}{%
2631   \ifmmode
2632     \ll
2633   \else
2634     \save@sf@q{\nobreak
2635       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2636   \fi}
2637 \ProvideTextCommand{\guillemotright}{OT1}{%
2638   \ifmmode
2639     \gg
2640   \else
2641     \save@sf@q{\nobreak
```

```

2642      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2643    \fi}

```

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

```

2644 \ProvideTextCommandDefault{\guillemetleft}{%
2645   \UseTextSymbol{OT1}{\guillemetleft}}
2646 \ProvideTextCommandDefault{\guillemetright}{%
2647   \UseTextSymbol{OT1}{\guillemetright}}
2648 \ProvideTextCommandDefault{\guillemotleft}{%
2649   \UseTextSymbol{OT1}{\guillemotleft}}
2650 \ProvideTextCommandDefault{\guillemotright}{%
2651   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2652 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2653   \ifmmode
2654     <%
2655   \else
2656     \save@sf@q{\nobreak
2657       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2658     \fi}
2659 \ProvideTextCommand{\guilsinglright}{OT1}{%
2660   \ifmmode
2661     >%
2662   \else
2663     \save@sf@q{\nobreak
2664       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2665     \fi}

```

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

```

2666 \ProvideTextCommandDefault{\guilsinglleft}{%
2667   \UseTextSymbol{OT1}{\guilsinglleft}}
2668 \ProvideTextCommandDefault{\guilsinglright}{%
2669   \UseTextSymbol{OT1}{\guilsinglright}}

```

### 9.12.2 Letters

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

```

2670 \DeclareTextCommand{\ij}{OT1}{%
2671   i\kern-0.02em\bbl@allowhyphens j}
2672 \DeclareTextCommand{\IJ}{OT1}{%
2673   I\kern-0.02em\bbl@allowhyphens J}
2674 \DeclareTextCommand{\ij}{T1}{\char188}
2675 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2676 \ProvideTextCommandDefault{\ij}{%
2677   \UseTextSymbol{OT1}{\ij}}
2678 \ProvideTextCommandDefault{\IJ}{%
2679   \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).

```

2680 \def\crrtic@{\hrule height0.1ex width0.3em}
2681 \def\crttic@{\hrule height0.1ex width0.33em}
2682 \def\ddj@{%
2683   \setbox0\hbox{d}\dimen@=\ht0
2684   \advance\dimen@1ex
2685   \dimen@.45\dimen@
2686   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2687   \advance\dimen@ii.5ex
2688   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2689 \def\DDJ@{%
2690   \setbox0\hbox{D}\dimen@=.55\ht0
2691   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2692   \advance\dimen@ii.15ex % correction for the dash position
2693   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2694   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2695   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2696 %
2697 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2698 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2699 \ProvideTextCommandDefault{\dj}{%
2700   \UseTextSymbol{OT1}{\dj}}
2701 \ProvideTextCommandDefault{\DJ}{%
2702   \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.

```

2703 \DeclareTextCommand{\SS}{OT1}{\SS}
2704 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

### 9.12.3 Shorthands for quotation marks

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

`\glq` The ‘german’ single quotes.

```

\grq 2705 \ProvideTextCommandDefault{\glq}{%
2706   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2707 \ProvideTextCommand{\grq}{T1}{%
2708   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2709 \ProvideTextCommand{\grq}{TU}{%
2710   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2711 \ProvideTextCommand{\grq}{OT1}{%
2712   \save@sf@q{\kern-.0125em
2713     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2714     \kern.07em\relax}}
2715 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```



`\glqq` The ‘german’ double quotes.

```
\grqq 2716 \ProvideTextCommandDefault{\glqq}{%
2717 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2718 \ProvideTextCommand{\grqq}{T1}{%
2719 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2720 \ProvideTextCommand{\grqq}{TU}{%
2721 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2722 \ProvideTextCommand{\grqq}{OT1}{%
2723 \save@sf@q{\kern-.07em
2724 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2725 \kern.07em\relax}}
2726 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2727 \ProvideTextCommandDefault{\flq}{%
2728 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2729 \ProvideTextCommandDefault{\frq}{%
2730 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2731 \ProvideTextCommandDefault{\flqq}{%
2732 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2733 \ProvideTextCommandDefault{\frqq}{%
2734 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

#### 9.12.4 Umlauts and tremas

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

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

```
2735 \def\umlauthigh{%
2736 \def\bbl@umlauta##1{\leavevmode\bggroup%
2737 \expandafter\accent\csname\fontencoding dqpos\endcsname
2738 ##1\bbl@allowhyphens\egroup}%
2739 \let\bbl@umlaute\bbl@umlauta}
2740 \def\umlautlow{%
2741 \def\bbl@umlauta{\protect\lower@umlaut}}
2742 \def\umlautelow{%
2743 \def\bbl@umlaute{\protect\lower@umlaut}}
2744 \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.

```
2745 \expandafter\ifx\csname U@D\endcsname\relax
2746 \csname newdimen\endcsname\U@D
2747 \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.

```

2748 \def\lower@umlaut#1{%
2749   \leavevmode\bggroup
2750     \U@D 1ex%
2751     {\setbox\z@\hbox{%
2752       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2753       \dimen@ -.45ex\advance\dimen@\ht\z@
2754       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2755     \expandafter\accent\csname\fontencoding dqpos\endcsname
2756     \fontdimen5\font\U@D #1%
2757   \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).

```

2758 \AtBeginDocument{%
2759   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2760   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2761   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2762   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2763   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2764   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2765   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2766   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2767   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2768   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2769   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%

```

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

```

2770 \ifx\l@english\@undefined
2771   \chardef\l@english\z@
2772 \fi
2773 \ifx\l@babelnohyphens\@undefined
2774   \newlanguage\l@babelnohyphens
2775 \fi

```

## 9.13 Layout

### Work in progress.

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

```

2776 \bbl@trace{Bidi layout}
2777 \providecommand\IfBabelLayout[3]{#3}%

```

```

2778 \newcommand\BabelPatchSection[1]{%
2779   \@ifundefined{#1}{}{%
2780     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2781     \@namedef{#1}{%
2782       \ifstar{\bbl@presec@s{#1}}%
2783       {\@dblarg{\bbl@presec@x{#1}}}}}%
2784 \def\bbl@presec@x#1[#2]#3{%
2785   \bbl@exp{%
2786     \\select@language@x{\bbl@main@language}%
2787     \\bbl@cs{sspre@#1}%
2788     \\bbl@cs{ss@#1}%
2789     [\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2790     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2791     \\select@language@x{\languagename}}}%
2792 \def\bbl@presec@s#1#2{%
2793   \bbl@exp{%
2794     \\select@language@x{\bbl@main@language}%
2795     \\bbl@cs{sspre@#1}%
2796     \\bbl@cs{ss@#1}*%
2797     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2798     \\select@language@x{\languagename}}}%
2799 \IfBabelLayout{sectioning}%
2800   {\BabelPatchSection{part}%
2801    \BabelPatchSection{chapter}%
2802    \BabelPatchSection{section}%
2803    \BabelPatchSection{subsection}%
2804    \BabelPatchSection{subsubsection}%
2805    \BabelPatchSection{paragraph}%
2806    \BabelPatchSection{subparagraph}%
2807    \def\babel@toc#1{%
2808      \select@language@x{\bbl@main@language}}}%
2809 \IfBabelLayout{captions}%
2810   {\BabelPatchSection{caption}}}%

```

## 9.14 Load engine specific macros

```

2811 \bbl@trace{Input engine specific macros}
2812 \ifcase\bbl@engine
2813   \input txtbabel.def
2814 \or
2815   \input luababel.def
2816 \or
2817   \input xebabel.def
2818 \fi

```

## 9.15 Creating and modifying languages

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

```

2819 \bbl@trace{Creating languages and reading ini files}
2820 \newcommand\babelprovide[2][]{%
2821   \let\bbl@savelangname\languagename
2822   \edef\bbl@savelocaleid{\the\localeid}%
2823   % Set name and locale id
2824   \edef\languagename{#2}%
2825   % \global\@namedef{\bbl@lcname@#2}{#2}%
2826   \bbl@id@assign

```

```

2827 \let\bb1@KVP@captions\@nil
2828 \let\bb1@KVP@import\@nil
2829 \let\bb1@KVP@main\@nil
2830 \let\bb1@KVP@script\@nil
2831 \let\bb1@KVP@language\@nil
2832 \let\bb1@KVP@hyphenrules\@nil % only for provide@new
2833 \let\bb1@KVP@mapfont\@nil
2834 \let\bb1@KVP@maparabic\@nil
2835 \let\bb1@KVP@mapdigits\@nil
2836 \let\bb1@KVP@intraspace\@nil
2837 \let\bb1@KVP@intrapenalty\@nil
2838 \let\bb1@KVP@onchar\@nil
2839 \let\bb1@KVP@alph\@nil
2840 \let\bb1@KVP@Alph\@nil
2841 \let\bb1@KVP@info\@nil % Ignored with import? Or error/warning?
2842 \bb1@forkv{#1}{% TODO - error handling
2843   \in@{/}{##1}%
2844   \ifin@
2845     \bb1@renewinikey##1\@{##2}%
2846   \else
2847     \bb1@csarg\def{KVP@##1}{##2}%
2848   \fi}%
2849 % == import, captions ==
2850 \ifx\bb1@KVP@import\@nil\else
2851   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
2852   {\ifx\bb1@initload\relax
2853     \begingroup
2854       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
2855       \InputIfFileExists{babel-#2.tex}{}{}%
2856     \endgroup
2857   \else
2858     \xdef\bb1@KVP@import{\bb1@initload}%
2859   \fi}%
2860 {}}%
2861 \fi
2862 \ifx\bb1@KVP@captions\@nil
2863   \let\bb1@KVP@captions\bb1@KVP@import
2864 \fi
2865 % Load ini
2866 \bb1@ifunset{date#2}%
2867   {\bb1@provide@new{#2}}%
2868   {\bb1@ifblank{#1}%
2869     {\bb1@error
2870       {If you want to modify `#2' you must tell how in\\%
2871         the optional argument. See the manual for the\\%
2872         available options.}%
2873       {Use this macro as documented}}%
2874     {\bb1@provide@renew{#2}}}%
2875 % Post tasks
2876 \bb1@exp{\bb1@babelensure[exclude=\\today]{#2}}%
2877 \bb1@ifunset{\bb1@ensure@language}%
2878   {\bb1@exp{%
2879     \\DeclareRobustCommand\<\bb1@ensure@language>[1]{%
2880       \\foreignlanguage{\language}%
2881       {###1}}}%
2882   {}%
2883 \bb1@exp{%
2884   \\bb1@toglobal\<\bb1@ensure@language>%
2885   \\bb1@toglobal\<\bb1@ensure@language\space>%

```

```

2886 % At this point all parameters are defined if 'import'. Now we
2887 % execute some code depending on them. But what about if nothing was
2888 % imported? We just load the very basic parameters: ids and a few
2889 % more.
2890 \bbl@ifunset{bbl@lname@#2}% TODO. Duplicated
2891 {\def\BabelBeforeIni##1##2{%
2892     \begingroup
2893     \catcode\ [=12 \catcode\]=12 \catcode\==12
2894     \catcode\;=12 \catcode\|=12 %
2895     \let\bbl@ini@captions@aux\@gobbletwo
2896     \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2897     \bbl@read@ini{##1}{basic data}%
2898     \bbl@exportkey{chrng}{characters.ranges}{}%
2899     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2900     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2901     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2902     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2903     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2904     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2905     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2906     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2907     \bbl@exportkey{intsp}{typography.intraspace}{}%
2908     \ifx\bbl@initoload\relax\endinput\fi
2909     \endgroup}%
2910     \begingroup % boxed, to avoid extra spaces:
2911     \ifx\bbl@initoload\relax
2912         \setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
2913     \else
2914         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}}%
2915     \fi
2916     \endgroup}%
2917 }}%
2918 % == script, language ==
2919 % Override the values from ini or defines them
2920 \ifx\bbl@KVP@script\@nil\else
2921     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2922 \fi
2923 \ifx\bbl@KVP@language\@nil\else
2924     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2925 \fi
2926 % == onchar ==
2927 \ifx\bbl@KVP@onchar\@nil\else
2928     \bbl@luahyphenate
2929     \directlua{
2930         if Babel.locale_mapped == nil then
2931             Babel.locale_mapped = true
2932             Babel.linebreaking.add_before(Babel.locale_map)
2933             Babel.loc_to_scr = {}
2934             Babel.chr_to_loc = Babel.chr_to_loc or {}
2935         end}%
2936     \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2937     \ifin@
2938         \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2939             \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2940         \fi
2941         \bbl@exp{\\ \bbl@add\\ \bbl@starthyphens
2942             {\\ \bbl@patterns@lua{\languagename}}}%
2943         % TODO - error/warning if no script
2944         \directlua{

```

```

2945     if Babel.script_blocks['\bbl@cl{sbc}'] then
2946       Babel.loc_to_scr[\the\localeid] =
2947         Babel.script_blocks['\bbl@cl{sbc}']
2948       Babel.locale_props[\the\localeid].lc = \the\localeid\space
2949       Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
2950     end
2951   }%
2952 \fi
2953 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2954 \ifin@
2955   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{}%
2956   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{}%
2957   \directlua{
2958     if Babel.script_blocks['\bbl@cl{sbc}'] then
2959       Babel.loc_to_scr[\the\localeid] =
2960         Babel.script_blocks['\bbl@cl{sbc}']
2961     end}%
2962 \ifx\bbl@mapselect\undefined
2963   \AtBeginDocument{%
2964     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2965     {\selectfont}}%
2966   \def\bbl@mapselect{%
2967     \let\bbl@mapselect\relax
2968     \edef\bbl@prefontid{\fontid\font}}%
2969   \def\bbl@mapdir##1{%
2970     {\def\language{##1}%
2971      \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2972      \bbl@switchfont
2973      \directlua{
2974        Babel.locale_props[\the\csname bbl@id@##1\endcsname]
2975          [\bbl@prefontid] = \fontid\font\space}}}%
2976   \fi
2977   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2978 \fi
2979 % TODO - catch non-valid values
2980 \fi
2981 % == mapfont ==
2982 % For bidi texts, to switch the font based on direction
2983 \ifx\bbl@KVP@mapfont\@nil\else
2984   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2985   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
2986     mapfont. Use 'direction'.%
2987     {See the manual for details.}}}%
2988   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{}%
2989   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{}%
2990 \ifx\bbl@mapselect\undefined
2991   \AtBeginDocument{%
2992     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2993     {\selectfont}}%
2994   \def\bbl@mapselect{%
2995     \let\bbl@mapselect\relax
2996     \edef\bbl@prefontid{\fontid\font}}%
2997   \def\bbl@mapdir##1{%
2998     {\def\language{##1}%
2999      \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3000      \bbl@switchfont
3001      \directlua{Babel.fontmap
3002        [\the\csname bbl@wdir@##1\endcsname]
3003        [\bbl@prefontid]=\fontid\font}}}%

```

```

3004 \fi
3005 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3006 \fi
3007 % == intraspace, intrapenalty ==
3008 % For CJK, East Asian, Southeast Asian, if interspace in ini
3009 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3010 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3011 \fi
3012 \bbl@provide@intraspace
3013 % == hyphenate.other.locale ==
3014 \bbl@ifunset{bbl@hyotl@language}{}%
3015 {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
3016 \bbl@startcommands*{\language}{}%
3017 \bbl@csarg\bbl@foreach{hyotl@language}{%
3018 \ifcase\bbl@engine
3019 \ifnum##1<257
3020 \SetHyphenMap{\BabelLower{##1}{##1}}%
3021 \fi
3022 \else
3023 \SetHyphenMap{\BabelLower{##1}{##1}}%
3024 \fi}%
3025 \bbl@endcommands}%
3026 % == hyphenate.other.script ==
3027 \bbl@ifunset{bbl@hyots@language}{}%
3028 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
3029 \bbl@csarg\bbl@foreach{hyots@language}{%
3030 \ifcase\bbl@engine
3031 \ifnum##1<257
3032 \global\lccode##1=##1\relax
3033 \fi
3034 \else
3035 \global\lccode##1=##1\relax
3036 \fi}}%
3037 % == maparabic ==
3038 % Native digits, if provided in ini (TeX level, xe and lua)
3039 \ifcase\bbl@engine\else
3040 \bbl@ifunset{bbl@dgnat@language}{}%
3041 {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
3042 \expandafter\expandafter\expandafter
3043 \bbl@setdigits\csname bbl@dgnat@language\endcsname
3044 \ifx\bbl@KVP@maparabic\@nil\else
3045 \ifx\bbl@latinarabic\@undefined
3046 \expandafter\let\expandafter\@arabic
3047 \csname bbl@counter@language\endcsname
3048 \else % ie, if layout=counters, which redefines \@arabic
3049 \expandafter\let\expandafter\bbl@latinarabic
3050 \csname bbl@counter@language\endcsname
3051 \fi
3052 \fi
3053 \fi}%
3054 \fi
3055 % == mapdigits ==
3056 % Native digits (lua level).
3057 \ifodd\bbl@engine
3058 \ifx\bbl@KVP@mapdigits\@nil\else
3059 \bbl@ifunset{bbl@dgnat@language}{}%
3060 {\RequirePackage{luatexbase}%
3061 \bbl@activate@preotf
3062 \directlua{

```

```

3063     Babel = Babel or {} %% -> presets in luababel
3064     Babel.digits_mapped = true
3065     Babel.digits = Babel.digits or {}
3066     Babel.digits[\the\localeid] =
3067         table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3068     if not Babel.numbers then
3069         function Babel.numbers(head)
3070             local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3071             local GLYPH = node.id'glyph'
3072             local inmath = false
3073             for item in node.traverse(head) do
3074                 if not inmath and item.id == GLYPH then
3075                     local temp = node.get_attribute(item, LOCALE)
3076                     if Babel.digits[temp] then
3077                         local chr = item.char
3078                         if chr > 47 and chr < 58 then
3079                             item.char = Babel.digits[temp][chr-47]
3080                         end
3081                     end
3082                     elseif item.id == node.id'math' then
3083                         inmath = (item.subtype == 0)
3084                     end
3085                 end
3086             end
3087             return head
3088         end
3089     }}%
3090 \fi
3091 \fi
3092 % == alph, Alph ==
3093 % What if extras<lang> contains a \babel@save\@alph? It won't be
3094 % restored correctly when exiting the language, so we ignore
3095 % this change with the \bbl@alph@saved trick.
3096 \ifx\bbl@KVP@alph\@nil\else
3097     \toks@\expandafter\expandafter\expandafter{%
3098         \csname extras\language\endcsname}%
3099     \bbl@exp{%
3100         \def\<extras\language>{%
3101             \let\bbl@alph@saved\@alph
3102             \the\toks@
3103             \let\@alph\bbl@alph@saved
3104             \\\babel@save\@alph
3105             \let\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
3106 \fi
3107 \ifx\bbl@KVP@Alph\@nil\else
3108     \toks@\expandafter\expandafter\expandafter{%
3109         \csname extras\language\endcsname}%
3110     \bbl@exp{%
3111         \def\<extras\language>{%
3112             \let\bbl@Alph@saved\@Alph
3113             \the\toks@
3114             \let\@Alph\bbl@Alph@saved
3115             \\\babel@save\@Alph
3116             \let\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3117 \fi
3118 % == require.babel in ini ==
3119 % To load or reload the babel-*.tex, if require.babel in ini
3120 \bbl@ifunset\bbl@rqtex\language\endcsname\@empty\else
3121     {\expandafter\ifx\csname bbl@rqtex\language\endcsname\@empty\else

```



```

3122 \let\BabelBeforeIni\@gobbletwo
3123 \chardef\atcatcode=\catcode`\@
3124 \catcode`\@=11\relax
3125 \InputIfFileExists{babel-\bbl@cs{rqtex@\language}\language}.tex}{\@}%
3126 \catcode`\@=\atcatcode
3127 \let\atcatcode\relax
3128 \fi}%
3129 % == main ==
3130 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3131 \let\language\bbl@savelangname
3132 \chardef\localeid\bbl@savelocaleid\relax
3133 \fi}

```

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

```

3134 \newcommand\localedigits{\@nameuse{\language digits}}
3135 \def\bbl@setdigits#1#2#3#4#5{%
3136 \bbl@exp{%
3137 \def\<\language digits>####1{% ie, \langdigits
3138 \<bbl@digits@\language>####1\\\@nil}%
3139 \def\<\language counter>####1{% ie, \langcounter
3140 \\\expandafter\<bbl@counter@\language>%
3141 \\\csname c@####1\endcsname}%
3142 \def\<bbl@counter@\language>####1{% ie, \bbl@counter@lang
3143 \\\expandafter\<bbl@digits@\language>%
3144 \\\number####1\\\@nil}}}%
3145 \def\bbl@tempa##1##2##3##4##5{%
3146 \bbl@exp{% Wow, quite a lot of hashes! :-(
3147 \def\<bbl@digits@\language>#####1{%
3148 \\\ifx#####1\\\@nil % ie, \bbl@digits@lang
3149 \\\else
3150 \\\ifx0#####1#1%
3151 \\\else\\\ifx1#####1#2%
3152 \\\else\\\ifx2#####1#3%
3153 \\\else\\\ifx3#####1#4%
3154 \\\else\\\ifx4#####1#5%
3155 \\\else\\\ifx5#####1#1%
3156 \\\else\\\ifx6#####1#2%
3157 \\\else\\\ifx7#####1#3%
3158 \\\else\\\ifx8#####1#4%
3159 \\\else\\\ifx9#####1#5%
3160 \\\else#####1%
3161 \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3162 \\\expandafter\<bbl@digits@\language>%
3163 \\\fi}}}%
3164 \bbl@tempa}

```

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

```

3165 \def\bbl@provide@new#1{%
3166 \@namedef{date#1}{\@nameuse{\language date}} % marks lang exists - required by \StartBabelCommands
3167 \@namedef{extras#1}{\@nameuse{\language extras}}
3168 \@namedef{noextras#1}{\@nameuse{\language noextras}}
3169 \bbl@startcommands*{#1}{captions}%
3170 \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3171 \def\bbl@tempb##1{% elt for \bbl@captionslist
3172 \ifx##1\@empty\else
3173 \bbl@exp{%
3174 \\\SetString\\##1{%

```

```

3175      \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3176      \expandafter\bbl@tempb
3177      \fi}%
3178      \expandafter\bbl@tempb\bbl@captionslist\@empty
3179      \else
3180      \ifx\bbl@initoload\relax
3181      \bbl@read@ini{\bbl@KVP@captions}{data}% Here letters cat = 11
3182      \else
3183      \bbl@read@ini{\bbl@initoload}{data}% Here all letters cat = 11
3184      \fi
3185      \bbl@after@ini
3186      \bbl@savestrings
3187      \fi
3188      \StartBabelCommands*{#1}{date}%
3189      \ifx\bbl@KVP@import\@nil
3190      \bbl@exp{%
3191      \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
3192      \else
3193      \bbl@savetoday
3194      \bbl@savedate
3195      \fi
3196      \bbl@endcommands
3197      \bbl@ifunset{\bbl@lname@#1}%      TODO. Duplicated
3198      {\def\BabelBeforeIni##1##2{%
3199      \begingroup
3200      \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3201      \catcode`\;=12 \catcode`\|=12 %
3202      \let\bbl@ini@captions@aux\@gobbletwo
3203      \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
3204      \bbl@read@ini{##1}{basic data}%
3205      \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3206      \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3207      \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3208      \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3209      \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3210      \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3211      \bbl@exportkey{intsp}{typography.intraspace}{}%
3212      \bbl@exportkey{chrng}{characters.ranges}{}%
3213      \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3214      \ifx\bbl@initoload\relax\endinput\fi
3215      \endgroup}%
3216      \begingroup      % boxed, to avoid extra spaces:
3217      \ifx\bbl@initoload\relax
3218      \setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%
3219      \else
3220      \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}}%
3221      \fi
3222      \endgroup}%
3223      }%
3224      \bbl@exp{%
3225      \gdef\<#1hyphenmins>{%
3226      {\bbl@ifunset{\bbl@lftm@#1}{2}{\bbl@cs{lftm@#1}}}%
3227      {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3228      \bbl@provide@hyphens{#1}%
3229      \ifx\bbl@KVP@main\@nil\else
3230      \expandafter\main@language\expandafter{#1}%
3231      \fi}
3232      \def\bbl@provide@renew#1{%
3233      \ifx\bbl@KVP@captions\@nil\else

```

```

3234 \StartBabelCommands*{#1}{captions}%
3235 \bbl@read@ini{\bbl@KVP@captions}{data}% Here all letters cat = 11
3236 \bbl@after@ini
3237 \bbl@savestrings
3238 \EndBabelCommands
3239 \fi
3240 \ifx\bbl@KVP@import\@nil\else
3241 \StartBabelCommands*{#1}{date}%
3242 \bbl@savetoday
3243 \bbl@savedate
3244 \EndBabelCommands
3245 \fi
3246 % == hyphenrules ==
3247 \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3248 \def\bbl@provide@hyphens#1{%
3249 \let\bbl@tempa\relax
3250 \ifx\bbl@KVP@hyphenrules\@nil\else
3251 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3252 \bbl@foreach\bbl@KVP@hyphenrules{%
3253 \ifx\bbl@tempa\relax % if not yet found
3254 \bbl@ifsamestring{##1}{+}%
3255 {\bbl@exp{\addlanguage<1@##1>}}}%
3256 }%
3257 \bbl@ifunset{1@##1}%
3258 }%
3259 {\bbl@exp{\let\bbl@tempa<1@##1>}}}%
3260 \fi}%
3261 \fi
3262 \ifx\bbl@tempa\relax % if no opt or no language in opt found
3263 \ifx\bbl@KVP@import\@nil
3264 \ifx\bbl@initoload\relax\else
3265 \bbl@exp{% and hyphenrules is not empty
3266 \bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3267 }%
3268 {\let\bbl@tempa<1@\bbl@cl{hyphr}>}}}%
3269 \fi
3270 \else % if importing
3271 \bbl@exp{% and hyphenrules is not empty
3272 \bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3273 }%
3274 {\let\bbl@tempa<1@\bbl@cl{hyphr}>}}}%
3275 \fi
3276 \fi
3277 \bbl@ifunset{\bbl@tempa}% ie, relax or undefined
3278 {\bbl@ifunset{1@#1}% no hyphenrules found - fallback
3279 {\bbl@exp{\adddialect<1@#1>\language}}}%
3280 }% so, 1@<lang> is ok - nothing to do
3281 {\bbl@exp{\adddialect<1@#1>\bbl@tempa}}}% found in opt list or ini
3282

```

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.

```

3283 \ifx\bbl@readstream\@undefined
3284 \csname newread\endcsname\bbl@readstream
3285 \fi
3286 \def\bbl@inipreread#1=#2\@{%
3287 \bbl@trim\def\bbl@tempa{#1}% Redundant below !!

```

```

3288 \bbl@trim\toks@{#2}%
3289 % Move trims here ??
3290 \bbl@ifunset{\bbl@KVP@\bbl@section/\bbl@tempa}%
3291 {\bbl@exp{%
3292     \\g@addto@macro\\bbl@inidata{%
3293         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3294     \expandafter\bbl@inireader\bbl@tempa=#2\@@}%
3295 }%
3296 \def\bbl@read@ini#1#2{%
3297     \bbl@csarg\edef{\lini@\languagename}{#1}%
3298     \openin\bbl@readstream=babel-#1.ini
3299     \ifeof\bbl@readstream
3300         \bbl@error
3301         {There is no ini file for the requested language\\%
3302         (#1). Perhaps you misspelled it or your installation\\%
3303         is not complete.}%
3304         {Fix the name or reinstall babel.}%
3305     \else
3306         \bbl@exp{\def\\bbl@inidata{\\bbl@elt{identificacion}{tag.ini}{#1}}}%
3307         \let\bbl@section\@empty
3308         \let\bbl@savestrings\@empty
3309         \let\bbl@savetoday\@empty
3310         \let\bbl@savestate\@empty
3311         \let\bbl@inireader\bbl@iniskip
3312         \bbl@info{Importing #2 for \languagename\\%
3313             from babel-#1.ini. Reported}%
3314         \loop
3315         \if \ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3316             \endlinechar\m@ne
3317             \read\bbl@readstream to \bbl@line
3318             \endlinechar\^^M
3319             \ifx\bbl@line\@empty\else
3320                 \expandafter\bbl@inline\bbl@line\bbl@inline
3321             \fi
3322         \repeat
3323         \bbl@foreach\bbl@renewlist{%
3324             \bbl@ifunset{\bbl@renew@##1}{\bbl@inisec[##1]\@@}%
3325             \global\let\bbl@renewlist\@empty
3326             % Ends last section. See \bbl@inisec
3327             \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
3328             \bbl@cs{renew@\bbl@section}%
3329             \global\bbl@csarg\let{renew@\bbl@section}\relax
3330             \bbl@cs{secpost@\bbl@section}%
3331             \bbl@csarg{\global\expandafter\let}{inidata@\languagename}\bbl@inidata
3332             \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\languagename}}%
3333             \bbl@tglobal\bbl@ini@loaded
3334         \fi}
3335 \def\bbl@inline#1\bbl@inline{%
3336     \@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.

3337 \def\bbl@iniskip#1\@@{%          if starts with ;
3338 \def\bbl@inisec[##1]##2\@@{%    if starts with opening bracket
3339     \def\bbl@elt##1##2{%
3340         \expandafter\toks@\expandafter{%

```

```

3341 \expandafter{\bbl@section}{##1}{##2}}%
3342 \bbl@exp{%
3343 \\\g@addto@macro\\bbl@inidata{\\bbl@elt\the\toks@}}%
3344 \bbl@inireader##1=##2\@}%
3345 \bbl@cs{renew@bbl@section}%
3346 \global\bbl@csarg\let{renew@bbl@section}\relax
3347 \bbl@cs{secpost@bbl@section}%
3348 % The previous code belongs to the previous section.
3349 % Now start the current one.
3350 \def\bbl@section{#1}%
3351 \def\bbl@elt##1##2{%
3352 \namedef{bbl@KVP@#1/##1}{}}%
3353 \bbl@cs{renew@#1}%
3354 \bbl@cs{secpre@#1}% pre-section 'hook'
3355 \bbl@ifunset{bbl@inikv@#1}%
3356 {\let\bbl@inireader\bbl@iniskip}%
3357 {\bbl@exp{\let\\bbl@inireader<bbl@inikv@#1>}}%
3358 \let\bbl@renewlist\@empty
3359 \def\bbl@renewinikv#1/#2\@#3{%
3360 \bbl@ifunset{bbl@renew@#1}%
3361 {\bbl@add@list\bbl@renewlist{#1}}%
3362 {}}%
3363 \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>.

```

3364 \def\bbl@inikv#1=#2\@{%      key=value
3365 \bbl@trim\def\bbl@tempa{#1}%
3366 \bbl@trim\toks@{#2}%
3367 \bbl@csarg\edef{@kv@bbl@section.\bbl@tempa}{\the\toks@}}

```

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

```

3368 \def\bbl@exportkey#1#2#3{%
3369 \bbl@ifunset{bbl@@kv@#2}%
3370 {\bbl@csarg\gdef{#1@language}{#3}}%
3371 {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3372 \bbl@csarg\gdef{#1@language}{#3}}%
3373 \else
3374 \bbl@exp{\global\let<bbl@#1@language>\<bbl@@kv@#2>}}%
3375 \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.

```

3376 \def\bbl@iniwarning#1{%
3377 \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
3378 {\bbl@warning{%
3379 From babel-\bbl@cs{lini@language}.ini:\\%
3380 \bbl@cs{@kv@identification.warning#1}\\%
3381 Reported }}}
3382 \let\bbl@inikv@identification\bbl@inikv
3383 \def\bbl@secpost@identification{%
3384 \bbl@iniwarning}%
3385 \ifcase\bbl@engine
3386 \bbl@iniwarning{.pdflatex}%
3387 \or
3388 \bbl@iniwarning{.lualatex}%
3389 \or

```

```

3390 \bbl@iniwarning{.xelatex}%
3391 \fi%
3392 \bbl@exportkey{elname}{identification.name.english}{}%
3393 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3394 {\csname bbl@elname\language\endcsname}}%
3395 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
3396 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3397 \bbl@exportkey{esname}{identification.script.name}{}%
3398 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3399 {\csname bbl@esname\language\endcsname}}%
3400 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3401 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
3402 \let\bbl@inikv@typography\bbl@inikv
3403 \let\bbl@inikv@characters\bbl@inikv
3404 \let\bbl@inikv@numbers\bbl@inikv
3405 \def\bbl@inikv@counters#1=#2\@@{%
3406 \def\bbl@tempc{#1}%
3407 \bbl@trim@def{\bbl@tempb*}{#2}%
3408 \in@{.1$}{#1$}%
3409 \ifin@
3410 \bbl@replace\bbl@tempc{.1}{}%
3411 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
3412 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3413 \fi
3414 \in@{.F.}{#1}%
3415 \ifin@else\in@{.S.}{#1}\fi
3416 \ifin@
3417 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3418 \else
3419 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3420 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3421 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3422 \fi}
3423 \def\bbl@after@ini{%
3424 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3425 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3426 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3427 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3428 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3429 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3430 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3431 \bbl@exportkey{intsp}{typography.intraspace}{}%
3432 \bbl@exportkey{jstfy}{typography.justify}{w}%
3433 \bbl@exportkey{chrng}{characters.ranges}{}%
3434 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3435 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3436 \bbl@toglobal\bbl@savetoday
3437 \bbl@toglobal\bbl@savedate}

```

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.

```

3438 \ifcase\bbl@engine
3439 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3440 \bbl@ini@captions@aux{#1}{#2}}
3441 \else
3442 \def\bbl@inikv@captions#1=#2\@@{%
3443 \bbl@ini@captions@aux{#1}{#2}}
3444 \fi

```

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

```
3445 \def\bbl@ini@captions@aux#1#2{%
3446   \bbl@trim@def\bbl@tempa{#1}%
3447   \bbl@ifblank{#2}%
3448   {\bbl@exp{%
3449     \toks@{\bbl@nocaption{\bbl@tempa}{\language\language\bbl@tempa name}}}%
3450   {\bbl@trim\toks@{#2}}}%
3451   \bbl@exp{%
3452     \bbl@add\bbl@savestrings{%
3453       \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.

```
3454 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%           for defaults
3455   \bbl@inidate#1...\relax{#2}{}}
3456 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
3457   \bbl@inidate#1...\relax{#2}{islamic}}
3458 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
3459   \bbl@inidate#1...\relax{#2}{hebrew}}
3460 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
3461   \bbl@inidate#1...\relax{#2}{persian}}
3462 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
3463   \bbl@inidate#1...\relax{#2}{indian}}
3464 \ifcase\bbl@engine
3465   \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%      override
3466     \bbl@inidate#1...\relax{#2}{}}
3467   \bbl@csarg\def{secpre@date.gregorian.licr}{%              discard uni
3468     \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
3469 \fi
3470 % TODO. With the following there is no need to ensure if \select...
3471 \newcommand\localedate{\@nameuse{\bbl@date@\language}}
3472 % eg: 1=months, 2=wide, 3=1, 4=dummy
3473 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3474   \bbl@trim@def\bbl@tempa{#1.#2}%
3475   \bbl@ifsamestring{\bbl@tempa}{months.wide}%              to savedate
3476   {\bbl@trim@def\bbl@tempa{#3}%
3477     \bbl@trim\toks@{#5}%
3478     \bbl@exp{%
3479       \bbl@add\bbl@savestate{%
3480         \SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}%
3481     {\bbl@ifsamestring{\bbl@tempa}{date.long}%              defined now
3482       {\bbl@trim@def\bbl@toreplace{#5}%
3483         \bbl@TG@date
3484         \global\bbl@csarg\let{date@\language}\bbl@toreplace
3485         \bbl@exp{%
3486           \gdef\<\language date>{\protect\<\language date >}}%
3487           \gdef\<\language date >####1####2####3{%
3488             \bbl@usedategroupttrue
3489             \<\language date>{\bbl@date@\language}{####1}{####2}{####3}}}%
3490             \bbl@add\bbl@savetoday{%
3491               \SetString\<\today>{\<\language date>{\the\year}{\the\month}{\the\day}}}%
3492             \<\language date>{\the\year}{\the\month}{\the\day}}}%
3493     }}
3494   }}
```

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.

```

3495 \let\bbl@calendar\@empty
3496 \newcommand\BabelDateSpace{\nobreakspace}
3497 \newcommand\BabelDateDot{.\@}
3498 \newcommand\BabelDated[1]{\number#1}}
3499 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}}
3500 \newcommand\BabelDateM[1]{\number#1}}
3501 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}}
3502 \newcommand\BabelDateMMM[1]{\%
3503 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3504 \newcommand\BabelDatey[1]{\number#1}}%
3505 \newcommand\BabelDateyy[1]{\%
3506 \ifnum#1<10 0\number#1 %
3507 \else\ifnum#1<100 \number#1 %
3508 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3509 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3510 \else
3511 \bbl@error
3512 {Currently two-digit years are restricted to the\
3513 range 0-9999.}%
3514 {There is little you can do. Sorry.}%
3515 \fi\fi\fi\fi}}
3516 \newcommand\BabelDateyyyy[1]{\number#1}} % FIXME - add leading 0
3517 \def\bbl@replace@finish@iii#1{%
3518 \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
3519 \def\bbl@TG@date{%
3520 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}}%
3521 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}}%
3522 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}}%
3523 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}}%
3524 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}}%
3525 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}}%
3526 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}}%
3527 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}}%
3528 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}}%
3529 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}}%
3530 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[####1|]}%
3531 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[####2|]}%
3532 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[####3|]}%
3533 % Note after \bbl@replace \toks@ contains the resulting string.
3534 % TODO - Using this implicit behavior doesn't seem a good idea.
3535 \bbl@replace@finish@iii\bbl@toreplace}
3536 \def\bbl@datecctr[#1|#2]{\localenumeral{#2}{#1}}

```

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

```

3537 \def\bbl@provide@lsys#1{%
3538 \bbl@ifunset{\bbl@lname@#1}%
3539 {\bbl@ini@basic{#1}}%
3540 {}%
3541 \bbl@csarg\let{lsys@#1}\@empty
3542 \bbl@ifunset{\bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
3543 \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
3544 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}}%
3545 \bbl@ifunset{\bbl@lname@#1}{%
3546 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3547 \ifcase\bbl@engine\or\or
3548 \bbl@ifunset{\bbl@prehc@#1}{}%

```



```

3549      {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3550      }%
3551      {\bbl@csarg\bbl@add@list{lsys@#1}{HyphenChar="200B}}}%
3552 \fi
3553 \bbl@csarg\bbl@toglobal{lsys@#1}}

```

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

```

3554 \def\bbl@ini@basic#1{%
3555   \def\BabelBeforeIni##1##2{%
3556     \begingroup
3557     \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
3558     \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3559     \catcode`\;=12 \catcode`\|=12 %
3560     \bbl@read@ini{##1}{font and identification data}%
3561     \endinput          % babel- .tex may contain onlypreamble's
3562     \endgroup}%        boxed, to avoid extra spaces:
3563     {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}{}}}%

```

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

```

3564 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3565   \ifx\\#1%           % \ before, in case #1 is multiletter
3566     \bbl@exp{%
3567       \def\\bbl@tempa####1{%
3568         \ifcase>####1\space\the\toks@<\else>\\@ctrerr<\fi>}}%
3569     \else
3570       \toks@>\expandafter{\the\toks@>\or #1}%
3571       \expandafter\bbl@buildifcase
3572     \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as a special case, for a fixed form (see babel-he.ini, for example).

```

3573 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\language}{#2}}
3574 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3575 \newcommand\localecounter[2]{%
3576   \expandafter\bbl@localecntr\csname c@#2\endcsname{#1}}
3577 \def\bbl@alphnumeral#1#2{%
3578   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3579 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3580   \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
3581     \bbl@alphnumeral@ii{#9}000000#1\or
3582     \bbl@alphnumeral@ii{#9}00000#1#2\or
3583     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3584     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3585     \bbl@alphnum@invalid{>9999}%
3586   \fi}
3587 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3588   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3589   {\bbl@cs{cntr@#1.4@\language}{#5}%
3590    \bbl@cs{cntr@#1.3@\language}{#6}%
3591    \bbl@cs{cntr@#1.2@\language}{#7}%

```

```

3592 \bbl@cs{cntr@#1.1@\language#8%
3593 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3594 \bbl@ifunset{bbl@cntr@#1.S.321@\language#}%
3595 {\bbl@cs{cntr@#1.S.321@\language#}%
3596 \fi}%
3597 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language#}}
3598 \def\bbl@alphnum@invalid#1{%
3599 \bbl@error{Alphabetic numeral too large (#1)}%
3600 {Currently this is the limit.}}

```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

3601 \newcommand\localeinfo[1]{%
3602 \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\language#}%
3603 {\bbl@error{I've found no info for the current locale.\%
3604 The corresponding ini file has not been loaded\%
3605 Perhaps it doesn't exist}%
3606 {See the manual for details.}}%
3607 {\bbl@cs{\csname bbl@info@#1\endcsname @\language#}}
3608 % \@namedef{bbl@info@name.locale}{lcname}
3609 \@namedef{bbl@info@tag.ini}{lini}
3610 \@namedef{bbl@info@name.english}{elname}
3611 \@namedef{bbl@info@name.opentype}{lname}
3612 \@namedef{bbl@info@tag.bcp47}{lbcpr}
3613 \@namedef{bbl@info@tag.opentype}{lotf}
3614 \@namedef{bbl@info@script.name}{esname}
3615 \@namedef{bbl@info@script.name.opentype}{sname}
3616 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3617 \@namedef{bbl@info@script.tag.opentype}{sotf}
3618 \let\bbl@ensureinfo@gobble
3619 \newcommand\BabelEnsureInfo{%
3620 \def\bbl@ensureinfo##1{%
3621 \ifx\InputIfFileExists\@undefined\else % not in plain
3622 \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}}%
3623 \fi}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3624 \newcommand\getlocaleproperty[3]{%
3625 \let#1\relax
3626 \def\bbl@elt##1##2##3{%
3627 \bbl@ifsamestring{##1/##2}{##3}%
3628 {\providecommand#1{##3}%
3629 \def\bbl@elt####1####2####3{}}%
3630 {}}%
3631 \bbl@cs{inidata@#2}%
3632 \ifx#1\relax
3633 \bbl@error
3634 {Unknown key for locale '#2':\%
3635 #3\%
3636 \string#1 will be set to \relax}%
3637 {Perhaps you misspelled it.}%
3638 \fi}
3639 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

## 10 Adjusting the Babel bahavior

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

```
3640 \newcommand\babeladjust[1]{% TODO. Error handling.
3641   \bbl@forkv{#1}{%
3642     \bbl@ifunset{bbl@ADJ@##1@##2}%
3643     {\bbl@cs{ADJ@##1}{##2}}%
3644     {\bbl@cs{ADJ@##1@##2}}}
3645 %
3646 \def\bbl@adjust@lua#1#2{%
3647   \ifvmode
3648     \ifnum\currentgrouplevel=\z@
3649       \directlua{ Babel.#2 }%
3650       \expandafter\expandafter\expandafter\@gobble
3651     \fi
3652   \fi
3653   {\bbl@error   % The error is gobbled if everything went ok.
3654     {Currently, #1 related features can be adjusted only\\%
3655       in the main vertical list.}%
3656     {Maybe things change in the future, but this is what it is.}}}
3657 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3658   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3659 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3660   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3661 \@namedef{bbl@ADJ@bidi.text@on}{%
3662   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3663 \@namedef{bbl@ADJ@bidi.text@off}{%
3664   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3665 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3666   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3667 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3668   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3669 %
3670 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3671   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3672 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3673   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3674 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3675   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3676 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3677   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3678 %
3679 \def\bbl@adjust@layout#1{%
3680   \ifvmode
3681     #1%
3682     \expandafter\@gobble
3683   \fi
3684   {\bbl@error   % The error is gobbled if everything went ok.
3685     {Currently, layout related features can be adjusted only\\%
3686       in vertical mode.}%
3687     {Maybe things change in the future, but this is what it is.}}}
3688 \@namedef{bbl@ADJ@layout.tabular@on}{%
3689   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3690 \@namedef{bbl@ADJ@layout.tabular@off}{%
3691   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3692 \@namedef{bbl@ADJ@layout.lists@on}{%
3693   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3694 \@namedef{bbl@ADJ@layout.lists@on}{%
```

```

3695 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3696 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3697 \bbl@activateposthyphen}
3698 %
3699 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3700 \bbl@bcpallowedtrue}
3701 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3702 \bbl@bcpallowedfalse}
3703 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3704 \def\bbl@bcp@prefix{#1}}
3705 \def\bbl@bcp@prefix{bcp47-}
3706 \@namedef{bbl@ADJ@autoload.options}#1{%
3707 \def\bbl@autoload@options{#1}}
3708 \let\bbl@autoload@bcptoptions\@empty
3709 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3710 \def\bbl@autoload@bcptoptions{#1}}
3711 % TODO: use babel name, override
3712 %
3713 % As the final task, load the code for lua.
3714 %
3715 \ifx\directlua\@undefined\else
3716 \ifx\bbl@luapatterns\@undefined
3717 \input luababel.def
3718 \fi
3719 \fi
3720 </core>

```

A proxy file for switch.def

```

3721 <*kernel>
3722 \let\bbl@onlyswitch\@empty
3723 \input babel.def
3724 \let\bbl@onlyswitch\@undefined
3725 </kernel>
3726 <*patterns>

```

## 11 Loading hyphenation patterns

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

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

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

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

```

3727 <<Make sure ProvidesFile is defined>>
3728 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
3729 \xdef\bbl@format{\jobname}
3730 \def\bbl@version{\<version>}
3731 \def\bbl@date{\<date>}
3732 \ifx\AtBeginDocument\@undefined
3733 \def\@empty{}
3734 \let\orig@dump\dump
3735 \def\dump{%

```

```

3736 \ifx\@ztryfc\@undefined
3737 \else
3738 \toks0=\expandafter{\@preamblecmds}%
3739 \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3740 \def\@begindocumenthook{}%
3741 \fi
3742 \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3743 \fi
3744 <<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.

```

3745 \def\process@line#1#2 #3 #4 {%
3746 \ifx=#1%
3747 \process@synonym{#2}%
3748 \else
3749 \process@language{#1#2}{#3}{#4}%
3750 \fi
3751 \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.

```

3752 \toks@{}
3753 \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.

```

3754 \def\process@synonym#1{%
3755 \ifnum\last@language=\m@ne
3756 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3757 \else
3758 \expandafter\chardef\csname l@#1\endcsname\last@language
3759 \wlog{\string\l@#1=\string\language\the\last@language}%
3760 \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
3761 \csname\language\hyphenmins\endcsname
3762 \let\bbl@elt\relax
3763 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
3764 \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read. For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

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

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

3765 \def\process@language#1#2#3{%
3766   \expandafter\addlanguage\csname l@#1\endcsname
3767   \expandafter\language\csname l@#1\endcsname
3768   \edef\languagename{#1}%
3769   \bbl@hook@everylanguage{#1}%
3770   % > luatex
3771   \bbl@get@enc#1::\@@@
3772   \begingroup
3773     \lefthyphenmin\m@ne
3774     \bbl@hook@loadpatterns{#2}%
3775     % > luatex
3776     \ifnum\lefthyphenmin=\m@ne
3777     \else
3778       \expandafter\xdef\csname #1hyphenmins\endcsname{%
3779         \the\lefthyphenmin\the\righthyphenmin}%
3780     \fi
3781   \endgroup
3782   \def\bbl@tempa{#3}%
3783   \ifx\bbl@tempa\@empty\else
3784     \bbl@hook@loadexceptions{#3}%
3785     % > luatex
3786   \fi
3787   \let\bbl@elt\relax
3788   \edef\bbl@languages{%
3789     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3790   \ifnum\the\language=\z@
3791     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3792       \set@hyphenmins\tw@\thr@@\relax
3793     \else
3794       \expandafter\expandafter\expandafter\set@hyphenmins
3795       \csname #1hyphenmins\endcsname
3796     \fi
3797     \the\toks@
3798     \toks@{}%
3799   \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.

```

3800 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. loadkernel currently loads nothing, but define some basic macros instead.

```

3801 \def\bbbl@hook@everylanguage#1{}
3802 \def\bbbl@hook@loadpatterns#1{\input #1\relax}
3803 \let\bbbl@hook@loadexceptions\bbbl@hook@loadpatterns
3804 \def\bbbl@hook@loadkernel#1{%
3805   \def\addlanguage{\alloc@9\language\chardef\ccclvi}%
3806   \def\adddialect##1##2{%
3807     \global\chardef##1##2\relax
3808     \wlog{\string##1 = a dialect from \string\language##2}}%
3809   \def\iflanguage##1{%
3810     \expandafter\ifx\csname l@##1\endcsname\relax
3811       \nolater{##1}%
3812     \else
3813       \ifnum\csname l@##1\endcsname=\language
3814         \expandafter\expandafter\expandafter\@firstoftwo
3815       \else
3816         \expandafter\expandafter\expandafter\@secondoftwo
3817       \fi
3818     \fi}%
3819   \def\providehyphenmins##1##2{%
3820     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
3821       \@namedef{##1hyphenmins}{##2}%
3822     \fi}%
3823   \def\set@hyphenmins##1##2{%
3824     \lefthyphenmin##1\relax
3825     \righthyphenmin##2\relax}%
3826   \def\selectlanguage{%
3827     \errhelp{Selecting a language requires a package supporting it}%
3828     \errmessage{Not loaded}}%
3829   \let\foreignlanguage\selectlanguage
3830   \let\otherlanguage\selectlanguage
3831   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
3832   \def\bbbl@usehooks##1##2{% TODO. Temporary!!
3833     \def\setlocale{%
3834       \errhelp{Find an armchair, sit down and wait}%
3835       \errmessage{Not yet available}}%
3836     \let\uselocale\setlocale
3837     \let\locale\setlocale
3838     \let\selectlocale\setlocale
3839     \let\localename\setlocale
3840     \let\textlocale\setlocale
3841     \let\textlanguage\setlocale
3842     \let\languagetext\setlocale}
3843   \begingroup
3844     \def\AddBabelHook#1#2{%
3845       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3846         \def\next{\toks1}%
3847       \else
3848         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
3849       \fi
3850     \next}
3851   \ifx\directlua@undefined
3852     \ifx\XeTeXinputencoding@undefined\else
3853       \input xebabel.def
3854     \fi
3855   \else

```

```

3856 \input luababel.def
3857 \fi
3858 \openin1 = babel-\bbl@format.cfg
3859 \ifeof1
3860 \else
3861 \input babel-\bbl@format.cfg\relax
3862 \fi
3863 \closein1
3864 \endgroup
3865 \bbl@hook@loadkernel{switch.def}

```

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

```

3866 \openin1 = language.dat

```

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

```

3867 \def\languagename{english}%
3868 \ifeof1
3869 \message{I couldn't find the file language.dat,\space
3870         I will try the file hyphen.tex}
3871 \input hyphen.tex\relax
3872 \chardef\l@english\z@
3873 \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`.

```

3874 \last@language\m@ne

```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```

3875 \loop
3876 \endlinechar\m@ne
3877 \read1 to \bbl@line
3878 \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.

```

3879 \if T\ifeof1F\fi T\relax
3880 \ifx\bbl@line\empty\else
3881 \edef\bbl@line{\bbl@line\space\space\space}%
3882 \expandafter\process@line\bbl@line\relax
3883 \fi
3884 \repeat

```

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

```

3885 \begingroup
3886 \def\bbl@elt#1#2#3#4{%
3887 \global\language=#2\relax
3888 \gdef\languagename{#1}%
3889 \def\bbl@elt##1##2##3##4{}}%
3890 \bbl@languages
3891 \endgroup
3892 \fi
3893 \closein1

```



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

```
3894 \if/\the\toks@/\else
3895 \errhelp{language.dat loads no language, only synonyms}
3896 \errmessage{Orphan language synonym}
3897 \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.

```
3898 \let\bbl@line\@undefined
3899 \let\process@line\@undefined
3900 \let\process@synonym\@undefined
3901 \let\process@language\@undefined
3902 \let\bbl@get@enc\@undefined
3903 \let\bbl@hyph@enc\@undefined
3904 \let\bbl@tempa\@undefined
3905 \let\bbl@hook@loadkernel\@undefined
3906 \let\bbl@hook@everylanguage\@undefined
3907 \let\bbl@hook@loadpatterns\@undefined
3908 \let\bbl@hook@loadexceptions\@undefined
3909 </patterns>
```

Here the code for `iniTEX` ends.

## 12 Font handling with fontspec

Add the bidi handler just before luaoftload, 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].

```
3910 <<{*More package options}>> ≡
3911 \chardef\bbl@bidimode\z@
3912 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
3913 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
3914 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
3915 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
3916 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
3917 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
3918 <</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`.

```
3919 <<{*Font selection}>> ≡
3920 \bbl@trace{Font handling with fontspec}
3921 \@onlypreamble\babelfont
3922 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
3923 \bbl@foreach{#1}{\%
3924 \expandafter\ifx\csname date##1\endcsname\relax
3925 \IfFileExists{babel-##1.tex}%
3926 {\babelfontprovide{##1}}%
3927 {}%
3928 \fi}%
3929 \edef\bbl@tempa{#1}%
3930 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3931 \ifx\fontspec\@undefined
3932 \usepackage{fontspec}%
```

```

3933 \fi
3934 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3935 \bbl@bblfont}
3936 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
3937 \bbl@ifunset{\bbl@tempb family}%
3938 {\bbl@providfam{\bbl@tempb}}%
3939 {\bbl@exp{%
3940 \\\bbl@sreplace\<\bbl@tempb family >%
3941 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3942 % For the default font, just in case:
3943 \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}}%
3944 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3945 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
3946 \bbl@exp{%
3947 \let\<bbl@\bbl@tempb dflt@\language>\<bbl@\bbl@tempb dflt@>%
3948 \\\bbl@font@set\<bbl@\bbl@tempb dflt@\language>%
3949 \<\bbl@tempb default>\<\bbl@tempb family>}}%
3950 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
3951 \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:

```

3952 \def\bbl@providfam#1{%
3953 \bbl@exp{%
3954 \\\newcommand\<#1default>{}% Just define it
3955 \\\bbl@add@list\\\bbl@font@fams{#1}%
3956 \\\DeclareRobustCommand\<#1family>%
3957 \\\not@math@alphabet\<#1family>\relax
3958 \\\fontfamily\<#1default>\selectfont}%
3959 \\\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.

```

3960 \def\bbl@nostdfont#1{%
3961 \bbl@ifunset{\bbl@WFF@\f@family}%
3962 {\bbl@csarg\gdef{\bbl@WFF@\f@family}}}% Flag, to avoid dupl warns
3963 \bbl@infowarn{The current font is not a babel standard family:\%
3964 #1%
3965 \fontname\font\%
3966 There is nothing intrinsically wrong with this warning, and\%
3967 you can ignore it altogether if you do not need these\%
3968 families. But if they are used in the document, you should be\%
3969 aware 'babel' will no set Script and Language for them, so\%
3970 you may consider defining a new family with \string\babelfont.\%
3971 See the manual for further details about \string\babelfont.\%
3972 Reported}}
3973 {}}%
3974 \gdef\bbl@switchfont{%
3975 \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}}%
3976 \bbl@exp{% eg Arabic -> arabic
3977 \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}}%
3978 \bbl@foreach\bbl@font@fams{%
3979 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
3980 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
3981 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
3982 {}% 123=F - nothing!
3983 {\bbl@exp{% 3=T - from generic
3984 \global\let\<bbl@##1dflt@\language>%
3985 \<bbl@##1dflt@>}}}%
3986 {\bbl@exp{% 2=T - from script

```

```

3987      \global\let\<bbl@##1dflt@\languagename>%
3988      \<bbl@##1dflt@*\bbl@tempa>}}}%
3989      {}}}%
3990      \def\bbl@tempa{\bbl@nostdfont{}}}%
3991      \bbl@foreach\bbl@font@fams{%      don't gather with prev for
3992      \bbl@ifunset{bbl@##1dflt@\languagename}%
3993      {\bbl@cs{famrst@##1}%
3994      \global\bbl@csarg\let{famrst@##1}\relax}%
3995      {\bbl@exp{% order is relevant
3996      \\bbl@add\\originalTeX{%
3997      \\bbl@font@rst{bbl@cl{##1dflt}}}%
3998      \<##1default>\<##1family>{##1}}}%
3999      \\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4000      \<##1default>\<##1family>}}}%
4001      \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.

```

4002 \ifx\f@family\undefined\else      % if latex
4003 \ifcase\bbl@engine      % if pdftex
4004 \let\bbl@cckststdfonts\relax
4005 \else
4006 \def\bbl@cckststdfonts{%
4007 \begingroup
4008 \global\let\bbl@cckststdfonts\relax
4009 \let\bbl@tempa\@empty
4010 \bbl@foreach\bbl@font@fams{%
4011 \bbl@ifunset{bbl@##1dflt@}%
4012 {\@nameuse{##1family}%
4013 \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4014 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4015 \space\space\fontname\font\\}%
4016 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4017 \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4018 {}}}%
4019 \ifx\bbl@tempa\@empty\else
4020 \bbl@infowarn{The following font families will use the default\\%
4021 settings for all or some languages:\\%
4022 \bbl@tempa
4023 There is nothing intrinsically wrong with it, but\\%
4024 'babel' will no set Script and Language, which could\\%
4025 be relevant in some languages. If your document uses\\%
4026 these families, consider redefining them with \string\babelfont.\\%
4027 Reported}%
4028 \fi
4029 \endgroup}
4030 \fi
4031 \fi

```

Now the macros defining the font with fontspec.

When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.

```

4032 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4033 \bbl@xin@{<>}{#1}%
4034 \ifin@
4035 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4036 \fi

```

```

4037 \bbl@exp{%
4038   \def\#2{#1}%           eg, \rmdefault{\bbl@rmdflt@lang}
4039   \\\bbl@ifsamestring{#2}{\f@family}{\#3\let\#1\bbl@tempa\relax}{}}
4040 %       TODO - next should be global?, but even local does its job. I'm
4041 %       still not sure -- must investigate:
4042 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4043   \let\bbl@tempe\bbl@mapselect
4044   \let\bbl@mapselect\relax
4045   \let\bbl@temp@fam#4%     eg, '\rmfamily', to be restored below
4046   \let#4\empty           %       Make sure \renewfontfamily is valid
4047   \bbl@exp{%
4048     \let\#1\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4049     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4050     {\#1\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4051     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4052     {\#1\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4053     \renewfontfamily\#4%
4054     [\bbl@cs{lsys@\language}\#2]{\#3}% ie \bbl@exp{.}{\#3}
4055   \begingroup
4056     #4%
4057     \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4058   \endgroup
4059   \let#4\bbl@temp@fam
4060   \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4061   \let\bbl@mapselect\bbl@tempe}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```

4062 \def\bbl@font@rst#1#2#3#4{%
4063   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```

4064 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4065 \newcommand\babelFSstore[2][{}]{%
4066   \bbl@ifblank{#1}%
4067   {\bbl@csarg\def{sname@#2}{Latin}}%
4068   {\bbl@csarg\def{sname@#2}{#1}}%
4069   \bbl@provide@dirs{#2}%
4070   \bbl@csarg\ifnum{wdir@#2}>\z@
4071     \let\bbl@beforeforeign\leavevmode
4072     \EnableBabelHook{babel-bidi}%
4073   \fi
4074   \bbl@foreach{#2}{%
4075     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4076     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4077     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4078 \def\bbl@FSstore#1#2#3#4{%
4079   \bbl@csarg\edef{#2default#1}{\#3}%
4080   \expandafter\addto\csname extras#1\endcsname{%
4081     \let#4#3%
4082     \ifx#3\f@family
4083       \edef#3{\csname bbl@#2default#1\endcsname}%
4084       \fontfamily{#3}\selectfont
4085     \else

```

```

4086     \edef#3{\csname bbl@#2default#1\endcsname}%
4087     \fi}%
4088 \expandafter\addto\csname noextras#1\endcsname{%
4089     \ifx#3\f@family
4090         \fontfamily{#4}\selectfont
4091     \fi
4092     \let#3#4}}
4093 \let\bbl@langfeatures\@empty
4094 \def\babelFSfeatures{% make sure \fontspec is redefined once
4095     \let\bbl@ori@fontspec\fontspec
4096     \renewcommand\fontspec[1][{}]{%
4097         \bbl@ori@fontspec[\bbl@langfeatures##1]}
4098     \let\babelFSfeatures\bbl@FSfeatures
4099     \babelFSfeatures}
4100 \def\bbl@FSfeatures#1#2{%
4101     \expandafter\addto\csname extras#1\endcsname{%
4102         \babel@save\bbl@langfeatures
4103         \edef\bbl@langfeatures{#2,}}
4104 <</Font selection>>

```

## 13 Hooks for XeTeX and LuaTeX

### 13.1 XeTeX

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

```

4105 <<{*Footnote changes}>> ≡
4106 \bbl@trace{Bidi footnotes}
4107 \ifnum\bbl@bidimode>\z@
4108     \def\bbl@footnote#1#2#3{%
4109         \@ifnextchar[%
4110             {\bbl@footnote@o{#1}{#2}{#3}}%
4111             {\bbl@footnote@x{#1}{#2}{#3}}}
4112     \def\bbl@footnote@x#1#2#3#4{%
4113         \bgroup
4114         \select@language@x{\bbl@main@language}%
4115         \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4116         \egroup}
4117     \def\bbl@footnote@o#1#2#3[#4]#5{%
4118         \bgroup
4119         \select@language@x{\bbl@main@language}%
4120         \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4121         \egroup}
4122     \def\bbl@footnotetext#1#2#3{%
4123         \@ifnextchar[%
4124             {\bbl@footnotetext@o{#1}{#2}{#3}}%
4125             {\bbl@footnotetext@x{#1}{#2}{#3}}}
4126     \def\bbl@footnotetext@x#1#2#3#4{%
4127         \bgroup
4128         \select@language@x{\bbl@main@language}%
4129         \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4130         \egroup}
4131     \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4132         \bgroup
4133         \select@language@x{\bbl@main@language}%
4134         \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4135         \egroup}

```

```

4136 \def\BabelFootnote#1#2#3#4{%
4137   \ifx\bbbl@fn@footnote\undefined
4138     \let\bbbl@fn@footnote\footnote
4139   \fi
4140   \ifx\bbbl@fn@footnotetext\undefined
4141     \let\bbbl@fn@footnotetext\footnotetext
4142   \fi
4143   \bbbl@ifblank{#2}%
4144     {\def#1{\bbbl@footnote{\@firstofone}{#3}{#4}}
4145      \@namedef{\bbbl@stripslash#1text}%
4146        {\bbbl@footnotetext{\@firstofone}{#3}{#4}}}%
4147     {\def#1{\bbbl@exp{\bbbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4148      \@namedef{\bbbl@stripslash#1text}%
4149        {\bbbl@exp{\bbbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4150 \fi
4151 <</Footnote changes>>

```

Now, the code.

```

4152 <*xetex>
4153 \def\BabelStringsDefault{unicode}
4154 \let\xebbl@stop\relax
4155 \AddBabelHook{xetex}{encodedcommands}{%
4156   \def\bbbl@tempa{#1}%
4157   \ifx\bbbl@tempa\empty
4158     \XeTeXinputencoding"bytes"%
4159   \else
4160     \XeTeXinputencoding"#1"%
4161   \fi
4162   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4163 \AddBabelHook{xetex}{stopcommands}{%
4164   \xebbl@stop
4165   \let\xebbl@stop\relax}
4166 \def\bbbl@intraspace#1 #2 #3@@{%
4167   \bbbl@csarg\gdef{\xeisp@{language}}%
4168     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4169 \def\bbbl@intrapenalty#1@@{%
4170   \bbbl@csarg\gdef{\xeipn@{language}}%
4171     {\XeTeXlinebreakpenalty #1\relax}}
4172 \def\bbbl@provide@intraspace{%
4173   \bbbl@xin@{\bbbl@cl{lnbrk}}{s}%
4174   \ifin@else\bbbl@xin@{\bbbl@cl{lnbrk}}{c}\fi
4175   \ifin@
4176     \bbbl@ifunset{\bbbl@intsp@{language}}{%
4177       {\expandafter\ifx\csname \bbbl@intsp@{language}\endcsname\empty\else
4178         \ifx\bbbl@KVP@intraspace\@nil
4179           \bbbl@exp{%
4180             \bbbl@intraspace\bbbl@cl{intsp}\@{}%
4181           \fi
4182           \ifx\bbbl@KVP@intrapenalty\@nil
4183             \bbbl@intrapenalty0\@@
4184           \fi
4185         \fi
4186         \ifx\bbbl@KVP@intraspace\@nil\else % We may override the ini
4187           \expandafter\bbbl@intraspace\bbbl@KVP@intraspace\@@
4188         \fi
4189         \ifx\bbbl@KVP@intrapenalty\@nil\else
4190           \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
4191         \fi
4192         \bbbl@exp{%

```

```

4193      \\bbl@add<extras\language>{%
4194      \XeTeXlinebreaklocale "bbl@cl{lbcpr}"%
4195      \<bbl@xeisp@language>%
4196      \<bbl@xeipn@language>%
4197      \\bbl@tglobal\<extras\language>%
4198      \\bbl@add<noextras\language>{%
4199      \XeTeXlinebreaklocale "en"%
4200      \\bbl@tglobal\<noextras\language>}%
4201      \ifx\bbl@ispace\undefined
4202      \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4203      \ifx\AtBeginDocument\@notprerr
4204      \expandafter\@secondoftwo % to execute right now
4205      \fi
4206      \AtBeginDocument{%
4207      \expandafter\bbl@add
4208      \csname selectfont \endcsname\bbl@ispace}%
4209      \expandafter\bbl@tglobal\csname selectfont \endcsname}%
4210      \fi}%
4211      \fi}
4212      \ifx\DisableBabelHook\undefined\endinput\fi
4213      \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4214      \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
4215      \DisableBabelHook{babel-fontspec}
4216      <<Font selection>>
4217      \input txtbabel.def
4218      </xetex>

```

## 13.2 Layout

*In progress.*

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

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

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdfTeX and xetex.

```

4219      (*texxet)
4220      \providecommand\bbl@provide@intraspace{}
4221      \bbl@trace{Redefinitions for bidi layout}
4222      \def\bbl@sspre@caption{%
4223      \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4224      \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4225      \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4226      \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4227      \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4228      \def\@hangfrom#1{%
4229      \setbox\@tempboxa\hbox{#1}%
4230      \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4231      \noindent\box\@tempboxa}
4232      \def\raggedright{%
4233      \let\@centercr
4234      \bbl@startskip\z@skip
4235      \@rightskip\@flushglue
4236      \bbl@endskip\@rightskip
4237      \parindent\z@
4238      \parfillskip\bbl@startskip}

```

```

4239 \def\raggedleft{%
4240   \let\\@centercr
4241   \bbl@startskip\@flushglue
4242   \bbl@endskip\z@skip
4243   \parindent\z@
4244   \parfillskip\bbl@endskip}
4245 \fi
4246 \IfBabelLayout{lists}
4247   {\bbl@sreplace\list
4248     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4249     \def\bbl@listleftmargin{%
4250       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4251     \ifcase\bbl@engine
4252       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4253       \def\p@enumii{\p@enumii}\theenumii{}
4254     \fi
4255     \bbl@sreplace\@verbatim
4256       {\leftskip\@totalleftmargin}%
4257       {\bbl@startskip\textwidth
4258         \advance\bbl@startskip-\linewidth}%
4259     \bbl@sreplace\@verbatim
4260       {\rightskip\z@skip}%
4261       {\bbl@endskip\z@skip}}%
4262   {}
4263 \IfBabelLayout{contents}
4264   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4265     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4266   {}
4267 \IfBabelLayout{columns}
4268   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4269     \def\bbl@outputbox#1{%
4270       \hb@xt@\textwidth{%
4271         \hskip\columnwidth
4272         \hfil
4273         {\normalcolor\vrule \@width\columnseprule}%
4274         \hfil
4275         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4276         \hskip-\textwidth
4277         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4278         \hskip\columnsep
4279         \hskip\columnwidth}}}%
4280   {}
4281 \langle\langle Footnote changes \rangle\rangle
4282 \IfBabelLayout{footnotes}%
4283   {\BabelFootnote\footnote\languagename{}\{}}%
4284   \BabelFootnote\localfootnote\languagename{}\{}}%
4285   \BabelFootnote\mainfootnote{}\{}}%
4286   {}

```

Implicitly reverses sectioning labels in bidi=basic, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

4287 \IfBabelLayout{counters}%
4288   {\let\bbl@latinarabic=@arabic
4289     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4290     \let\bbl@asciroman=@roman
4291     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4292     \let\bbl@asciiRoman=@Roman
4293     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
4294 \end{texet}

```



### 13.3 LuaTeX

The loader for luatex is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, `lua(e)tex` is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the base option); (2) at `hyphen.cfg`, to modify some macros; (3) in the middle of `plain.def` and `babel.sty`, by `babel.def`, with the commands and other definitions for luatex (eg, `\babelpatterns`).

```
4295 (*luatex)
4296 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4297 \bbl@trace{Read language.dat}
4298 \ifx\bbl@readstream\undefined
4299   \csname newread\endcsname\bbl@readstream
4300 \fi
4301 \begingroup
4302   \toks@{}
4303   \count@ \z@ % 0=start, 1=0th, 2=normal
4304   \def\bbl@process@line#1#2 #3 #4 {%
4305     \ifx=#1%
4306       \bbl@process@synonym{#2}%
4307     \else
4308       \bbl@process@language{#1#2}{#3}{#4}%
4309     \fi
4310     \ignorespaces}
4311   \def\bbl@manylang{%
4312     \ifnum\bbl@last>\@ne
4313       \bbl@info{Non-standard hyphenation setup}%
4314     \fi
```

```

4315 \let\bbl@manylang\relax}
4316 \def\bbl@process@language#1#2#3{%
4317 \ifcase\count@
4318 \@ifundefined{zth#1}{\count@tw@}{\count@ne}%
4319 \or
4320 \count@tw@
4321 \fi
4322 \ifnum\count@=\tw@
4323 \expandafter\addlanguage\csname l@#1\endcsname
4324 \language\allocationnumber
4325 \chardef\bbl@last\allocationnumber
4326 \bbl@manylang
4327 \let\bbl@elt\relax
4328 \xdef\bbl@languages{%
4329 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4330 \fi
4331 \the\toks@
4332 \toks@{}}
4333 \def\bbl@process@synonym@aux#1#2{%
4334 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4335 \let\bbl@elt\relax
4336 \xdef\bbl@languages{%
4337 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4338 \def\bbl@process@synonym#1{%
4339 \ifcase\count@
4340 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4341 \or
4342 \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{%
4343 \else
4344 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4345 \fi}
4346 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4347 \chardef\l@english\z@
4348 \chardef\l@USenglish\z@
4349 \chardef\bbl@last\z@
4350 \global\@namedef{\bbl@hyphendata@0}{\hyphen.tex}{}
4351 \gdef\bbl@languages{%
4352 \bbl@elt{english}{0}{\hyphen.tex}{}%
4353 \bbl@elt{USenglish}{0}{}}
4354 \else
4355 \global\let\bbl@languages@format\bbl@languages
4356 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4357 \ifnum#2>\z@\else
4358 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4359 \fi}%
4360 \xdef\bbl@languages{\bbl@languages}%
4361 \fi
4362 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4363 \bbl@languages
4364 \openin\bbl@readstream=language.dat
4365 \ifeof\bbl@readstream
4366 \bbl@warning{I couldn't find language.dat. No additional\\%
4367 patterns loaded. Reported}%
4368 \else
4369 \loop
4370 \endlinechar\m@ne
4371 \read\bbl@readstream to \bbl@line
4372 \endlinechar\^^M
4373 \if T\ifeof\bbl@readstream F\fi T\relax

```

```

4374         \ifx\bbl@line\@empty\else
4375             \edef\bbl@line{\bbl@line\space\space\space}%
4376             \expandafter\bbl@process@line\bbl@line\relax
4377         \fi
4378     \repeat
4379 \fi
4380 \endgroup
4381 \bbl@trace{Macros for reading patterns files}
4382 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4383 \ifx\babelcatcodetablenum\undefined
4384     \ifx\newcatcodetable\undefined
4385         \def\babelcatcodetablenum{5211}
4386         \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4387     \else
4388         \newcatcodetable\babelcatcodetablenum
4389         \newcatcodetable\bbl@pattcodes
4390     \fi
4391 \else
4392     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4393 \fi
4394 \def\bbl@luapatterns#1#2{%
4395     \bbl@get@enc#1::\@@
4396     \setbox\z@\hbox\bgroup
4397         \begingroup
4398             \savecatcodetable\babelcatcodetablenum\relax
4399             \initcatcodetable\bbl@pattcodes\relax
4400             \catcodetable\bbl@pattcodes\relax
4401             \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4402             \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4403             \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4404             \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4405             \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4406             \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4407             \input #1\relax
4408             \catcodetable\babelcatcodetablenum\relax
4409         \endgroup
4410     \def\bbl@tempa{#2}%
4411     \ifx\bbl@tempa\@empty\else
4412         \input #2\relax
4413     \fi
4414 \egroup}%
4415 \def\bbl@patterns@lua#1{%
4416     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4417         \csname l@#1\endcsname
4418         \edef\bbl@tempa{#1}%
4419     \else
4420         \csname l@#1:\f@encoding\endcsname
4421         \edef\bbl@tempa{#1:\f@encoding}%
4422     \fi\relax
4423     \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4424     \@ifundefined{bbl@hyphendata@the\language}%
4425     {\def\bbl@elt##1##2##3##4{%
4426         \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4427         \def\bbl@tempb{##3}%
4428         \ifx\bbl@tempb\@empty\else % if not a synonymous
4429             \def\bbl@tempc{##3}{##4}%
4430         \fi
4431         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4432     \fi}%

```

```

4433 \bbl@languages
4434 \ifundefined{bbl@hyphendata@the\language}%
4435 {\bbl@info{No hyphenation patterns were set for\%
4436 language '\bbl@tempa'. Reported}}%
4437 {\expandafter\expandafter\expandafter\bbl@luapatterns
4438 \csname bbl@hyphendata@the\language\endcsname}}{}
4439 \endinput\fi
4440 % Here ends \ifx\AddBabelHook\@undefined
4441 % A few lines are only read by hyphen.cfg
4442 \ifx\DisableBabelHook\@undefined
4443 \AddBabelHook{luatex}{everylanguage}{%
4444 \def\process@language##1##2##3{%
4445 \def\process@line####1####2 ####3 ####4 {}}}
4446 \AddBabelHook{luatex}{loadpatterns}{%
4447 \input #1\relax
4448 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4449 {\#1}}{}
4450 \AddBabelHook{luatex}{loadexceptions}{%
4451 \input #1\relax
4452 \def\bbl@tempb##1##2{{\#1}{\#1}}%
4453 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4454 {\expandafter\expandafter\expandafter\bbl@tempb
4455 \csname bbl@hyphendata@the\language\endcsname}}
4456 \endinput\fi
4457 % Here stops reading code for hyphen.cfg
4458 % The following is read the 2nd time it's loaded
4459 \begingroup
4460 \catcode`\%=12
4461 \catcode`\'=12
4462 \catcode`\%=12
4463 \catcode`\:=12
4464 \directlua{
4465 Babel = Babel or {}
4466 function Babel.bytes(line)
4467 return line:gsub(".",
4468 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4469 end
4470 function Babel.begin_process_input()
4471 if luatexbase and luatexbase.add_to_callback then
4472 luatexbase.add_to_callback('process_input_buffer',
4473 Babel.bytes,'Babel.bytes')
4474 else
4475 Babel.callback = callback.find('process_input_buffer')
4476 callback.register('process_input_buffer',Babel.bytes)
4477 end
4478 end
4479 function Babel.end_process_input ()
4480 if luatexbase and luatexbase.remove_from_callback then
4481 luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4482 else
4483 callback.unregister('process_input_buffer',Babel.callback)
4484 end
4485 end
4486 function Babel.addpatterns(pp, lg)
4487 local lg = lang.new(lg)
4488 local pats = lang.patterns(lg) or ''
4489 lang.clear_patterns(lg)
4490 for p in pp:gmatch('[^%s]+') do
4491 ss = ''

```

```

4492     for i in string.utfcharacters(p:gsub('%d', '')) do
4493         ss = ss .. '%d?' .. i
4494     end
4495     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4496     ss = ss:gsub('%.%%d%?$', '%%.')
4497     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4498     if n == 0 then
4499         tex.sprint(
4500             [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4501             .. p .. [[]])
4502         pats = pats .. ' ' .. p
4503     else
4504         tex.sprint(
4505             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4506             .. p .. [[]])
4507     end
4508 end
4509 lang.patterns(lg, pats)
4510 end
4511 }
4512 \endgroup
4513 \ifx\newattribute\@undefined\else
4514     \newattribute\bbl@attr@locale
4515     \AddBabelHook{luatex}{beforeextras}{%
4516         \setattribute\bbl@attr@locale\localeid}
4517 \fi
4518 \def\BabelStringsDefault{unicode}
4519 \let\luabbl@stop\relax
4520 \AddBabelHook{luatex}{encodedcommands}{%
4521     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4522     \ifx\bbl@tempa\bbl@tempb\else
4523         \directlua{Babel.begin_process_input()}%
4524         \def\luabbl@stop{%
4525             \directlua{Babel.end_process_input()}}%
4526     \fi}%
4527 \AddBabelHook{luatex}{stopcommands}{%
4528     \luabbl@stop
4529     \let\luabbl@stop\relax}
4530 \AddBabelHook{luatex}{patterns}{%
4531     \@ifundefined{bbl@hyphendata@the\language}%
4532     {\def\bbl@elt##1##2##3##4{%
4533         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4534         \def\bbl@tempb{##3}%
4535         \ifx\bbl@tempb\@empty\else % if not a synonymous
4536             \def\bbl@tempc{##3}{##4}%
4537         \fi
4538         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4539     \fi}%
4540     \bbl@languages
4541     \@ifundefined{bbl@hyphendata@the\language}%
4542     {\bbl@info{No hyphenation patterns were set for\%
4543         language '#2'. Reported}}%
4544     {\expandafter\expandafter\expandafter\bbl@luapatterns
4545         \csname bbl@hyphendata@the\language\endcsname}}}%
4546 \@ifundefined{bbl@patterns@}{%
4547     \begingroup
4548     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
4549     \ifin\else
4550         \ifx\bbl@patterns@\@empty\else

```

```

4551         \directlua{ Babel.addpatterns(
4552             [[\bbl@patterns@]], \number\language) }%
4553     \fi
4554     \@ifundefined{bbl@patterns@#1}%
4555         \@empty
4556         {\directlua{ Babel.addpatterns(
4557             [[\space\csname bbl@patterns@#1\endcsname]],
4558             \number\language) }}%
4559     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4560 \fi
4561 \endgroup}%
4562 \bbl@exp{%
4563     \bbl@ifunset{bbl@prehc@\language}\space}%
4564     {\bbl@ifblank{\bbl@cs{prehc@\language}}}%
4565     {\prehyphenchar=\bbl@c1{prehc}\relax}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4566 \@onlypreamble\babelpatterns
4567 \AtEndOfPackage{%
4568     \newcommand\babelpatterns[2][\@empty]{%
4569         \ifx\bbl@patterns@\relax
4570             \let\bbl@patterns@\@empty
4571         \fi
4572         \ifx\bbl@pttnlist@\@empty\else
4573             \bbl@warning{%
4574                 You must not intermingle \string\selectlanguage\space and\%
4575                 \string\babelpatterns\space or some patterns will not\%
4576                 be taken into account. Reported}%
4577             \fi
4578             \ifx\@empty#1%
4579                 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4580             \else
4581                 \edef\bbl@tempb{\zap@space#1 \@empty}%
4582                 \bbl@for\bbl@tempa\bbl@tempb{%
4583                     \bbl@fixname\bbl@tempa
4584                     \bbl@iflanguage\bbl@tempa{%
4585                         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4586                             \@ifundefined{bbl@patterns@\bbl@tempa}%
4587                                 \@empty
4588                                 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4589                             #2}}}%
4590             \fi}}

```

### 13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

*In progress.* Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4591 \directlua{
4592     Babel = Babel or {}
4593     Babel.linebreaking = Babel.linebreaking or {}
4594     Babel.linebreaking.before = {}
4595     Babel.linebreaking.after = {}

```

```

4596 Babel.locale = {} % Free to use, indexed with \localeid
4597 function Babel.linebreaking.add_before(func)
4598     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4599     table.insert(Babel.linebreaking.before, func)
4600 end
4601 function Babel.linebreaking.add_after(func)
4602     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4603     table.insert(Babel.linebreaking.after, func)
4604 end
4605 }
4606 \def\bbl@intraspace#1 #2 #3\@@{%
4607     \directlua{
4608         Babel = Babel or {}
4609         Babel.intraspaces = Babel.intraspaces or {}
4610         Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
4611             {b = #1, p = #2, m = #3}
4612         Babel.locale_props[\the\localeid].intraspace = %
4613             {b = #1, p = #2, m = #3}
4614     }}
4615 \def\bbl@intrapenalty#1\@@{%
4616     \directlua{
4617         Babel = Babel or {}
4618         Babel.intrapenalties = Babel.intrapenalties or {}
4619         Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
4620         Babel.locale_props[\the\localeid].intrapenalty = #1
4621     }}
4622 \begingroup
4623 \catcode`\%=12
4624 \catcode`\^=14
4625 \catcode`\'=12
4626 \catcode`\~=12
4627 \gdef\bbl@seaintraspace{^
4628     \let\bbl@seaintraspace\relax
4629     \directlua{
4630         Babel = Babel or {}
4631         Babel.sea_enabled = true
4632         Babel.sea_ranges = Babel.sea_ranges or {}
4633         function Babel.set_chranges (script, chrng)
4634             local c = 0
4635             for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
4636                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4637                 c = c + 1
4638             end
4639         end
4640         function Babel.sea_disc_to_space (head)
4641             local sea_ranges = Babel.sea_ranges
4642             local last_char = nil
4643             local quad = 655360 ^^ 10 pt = 655360 = 10 * 65536
4644             for item in node.traverse(head) do
4645                 local i = item.id
4646                 if i == node.id'glyph' then
4647                     last_char = item
4648                 elseif i == 7 and item.subtype == 3 and last_char
4649                     and last_char.char > 0x0C99 then
4650                     quad = font.getfont(last_char.font).size
4651                     for lg, rg in pairs(sea_ranges) do
4652                         if last_char.char > rg[1] and last_char.char < rg[2] then
4653                             lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
4654                             local intraspace = Babel.intraspaces[lg]

```

```

4655         local intrapenalty = Babel.intrapenalties[lg]
4656         local n
4657         if intrapenalty ~= 0 then
4658             n = node.new(14, 0)    ^^ penalty
4659             n.penalty = intrapenalty
4660             node.insert_before(head, item, n)
4661         end
4662         n = node.new(12, 13)    ^^ (glue, spaceskip)
4663         node.setglue(n, intraspace.b * quad,
4664                     intraspace.p * quad,
4665                     intraspace.m * quad)
4666         node.insert_before(head, item, n)
4667         node.remove(head, item)
4668     end
4669 end
4670 end
4671 end
4672 end
4673 }^^
4674 \bbl@luahyphenate}
4675 \catcode`\%=14
4676 \gdef\bbl@cjkintraspacespace{%
4677 \let\bbl@cjkintraspacespace\relax
4678 \directlua{
4679     Babel = Babel or {}
4680     require'babel-data-cjk.lua'
4681     Babel.cjk_enabled = true
4682     function Babel.cjk_linebreak(head)
4683         local GLYPH = node.id'glyph'
4684         local last_char = nil
4685         local quad = 655360    % 10 pt = 655360 = 10 * 65536
4686         local last_class = nil
4687         local last_lang = nil
4688
4689         for item in node.traverse(head) do
4690             if item.id == GLYPH then
4691
4692                 local lang = item.lang
4693
4694                 local LOCALE = node.get_attribute(item,
4695             luatexbase.registernumber'bbl@attr@locale')
4696                 local props = Babel.locale_props[LOCALE]
4697
4698                 local class = Babel.cjk_class[item.char].c
4699
4700                 if class == 'cp' then class = 'cl' end % )] as CL
4701                 if class == 'id' then class = 'I' end
4702
4703                 local br = 0
4704                 if class and last_class and Babel.cjk_breaks[last_class][class] then
4705                     br = Babel.cjk_breaks[last_class][class]
4706                 end
4707
4708                 if br == 1 and props.linebreak == 'c' and
4709                     lang ~= \the\l@nohyphenation\space and
4710                     last_lang ~= \the\l@nohyphenation then
4711                     local intrapenalty = props.intrapenalty
4712                     if intrapenalty ~= 0 then
4713                         local n = node.new(14, 0)    % penalty

```



```

4714         n.penalty = intrapenalty
4715         node.insert_before(head, item, n)
4716     end
4717     local intraspace = props.intraspace
4718     local n = node.new(12, 13)      % (glue, spaceskip)
4719     node.setglue(n, intraspace.b * quad,
4720                 intraspace.p * quad,
4721                 intraspace.m * quad)
4722     node.insert_before(head, item, n)
4723 end
4724
4725     quad = font.getfont(item.font).size
4726     last_class = class
4727     last_lang = lang
4728     else % if penalty, glue or anything else
4729         last_class = nil
4730     end
4731 end
4732 lang.hyphenate(head)
4733 end
4734 }%
4735 \bbl@luahyphenate}
4736 \gdef\bbl@luahyphenate{%
4737 \let\bbl@luahyphenate\relax
4738 \directlua{
4739     luatexbase.add_to_callback('hyphenate',
4740     function (head, tail)
4741         if Babel.linebreaking.before then
4742             for k, func in ipairs(Babel.linebreaking.before) do
4743                 func(head)
4744             end
4745         end
4746         if Babel.cjk_enabled then
4747             Babel.cjk_linebreak(head)
4748         end
4749         lang.hyphenate(head)
4750         if Babel.linebreaking.after then
4751             for k, func in ipairs(Babel.linebreaking.after) do
4752                 func(head)
4753             end
4754         end
4755         if Babel.sea_enabled then
4756             Babel.sea_disc_to_space(head)
4757         end
4758     end,
4759     'Babel.hyphenate')
4760 }
4761 }
4762 \endgroup
4763 \def\bbl@provide@intraspace{%
4764     \bbl@ifunset{\bbl@intsp@language}{}%
4765     {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
4766         \bbl@xin@{\bbl@cl{lbrk}}{c}%
4767         \ifin@           % cjk
4768         \bbl@cjk@intraspace
4769         \directlua{
4770             Babel = Babel or {}
4771             Babel.locale_props = Babel.locale_props or {}
4772             Babel.locale_props[\the\localeid].linebreak = 'c'

```

```

4773     }%
4774     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@cl}%
4775     \ifx\bbl@KVP@intrapenalty\@nil
4776     \bbl@intrapenalty0\@@
4777     \fi
4778   \else           % sea
4779     \bbl@seaintraspace
4780     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@cl}%
4781     \directlua{
4782       Babel = Babel or {}
4783       Babel.sea_ranges = Babel.sea_ranges or {}
4784       Babel.set_chranges('\bbl@cl{sbcpr}',
4785                          '\bbl@cl{chrng}')
4786     }%
4787     \ifx\bbl@KVP@intrapenalty\@nil
4788     \bbl@intrapenalty0\@@
4789     \fi
4790   \fi
4791 \fi
4792 \ifx\bbl@KVP@intrapenalty\@nil\else
4793   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4794 \fi}}

```

### 13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

*Work in progress.*

Common stuff.

```

4795 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4796 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4797 \DisableBabelHook{babel-fontspec}
4798 <<Font selection>>

```

### 13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

4799 \directlua{
4800 Babel.script_blocks = {
4801   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4802              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4803   ['Armn'] = {{0x0530, 0x058F}},
4804   ['Beng'] = {{0x0980, 0x09FF}},
4805   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},

```

```

4806 ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4807 ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4808             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4809 ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4810 ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4811             {0xAB00, 0xAB2F}},
4812 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4813 % Don't follow strictly Unicode, which places some Coptic letters in
4814 % the 'Greek and Coptic' block
4815 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4816 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4817             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4818             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4819             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4820             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4821             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4822 ['Hebr'] = {{0x0590, 0x05FF}},
4823 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4824             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4825 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4826 ['Knda'] = {{0x0C80, 0x0CFF}},
4827 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4828             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4829             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4830 ['Laoo'] = {{0x0E80, 0x0EFF}},
4831 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4832             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4833             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4834 ['Mahj'] = {{0x11150, 0x1117F}},
4835 ['Mlym'] = {{0x0D00, 0x0D7F}},
4836 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4837 ['Orya'] = {{0x0B00, 0x0B7F}},
4838 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4839 ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
4840 ['Taml'] = {{0x0B80, 0x0BFF}},
4841 ['Telu'] = {{0x0C00, 0x0C7F}},
4842 ['Tfng'] = {{0x2D30, 0x2D7F}},
4843 ['Thai'] = {{0x0E00, 0x0E7F}},
4844 ['Tibt'] = {{0x0F00, 0x0FFF}},
4845 ['Vaii'] = {{0xA500, 0xA63F}},
4846 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4847 }
4848
4849 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyr1
4850 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4851 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4852
4853 function Babel.locale_map(head)
4854   if not Babel.locale_mapped then return head end
4855
4856   local LOCALE = luatexbase.registernumber'bb1@attr@locale'
4857   local GLYPH = node.id('glyph')
4858   local inmath = false
4859   local toloc_save
4860   for item in node.traverse(head) do
4861     local toloc
4862     if not inmath and item.id == GLYPH then
4863       % Optimization: build a table with the chars found
4864       if Babel.chr_to_loc[item.char] then

```

```

4865         toloc = Babel.chr_to_loc[item.char]
4866     else
4867         for lc, maps in pairs(Babel.loc_to_scr) do
4868             for _, rg in pairs(maps) do
4869                 if item.char >= rg[1] and item.char <= rg[2] then
4870                     Babel.chr_to_loc[item.char] = lc
4871                     toloc = lc
4872                     break
4873                 end
4874             end
4875         end
4876     end
4877     % Now, take action, but treat composite chars in a different
4878     % fashion, because they 'inherit' the previous locale. Not yet
4879     % optimized.
4880     if not toloc and
4881         (item.char >= 0x0300 and item.char <= 0x036F) or
4882         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4883         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
4884         toloc = toloc_save
4885     end
4886     if toloc and toloc > -1 then
4887         if Babel.locale_props[toloc].lg then
4888             item.lang = Babel.locale_props[toloc].lg
4889             node.set_attribute(item, LOCALE, toloc)
4890         end
4891         if Babel.locale_props[toloc]['/'..item.font] then
4892             item.font = Babel.locale_props[toloc]['/'..item.font]
4893         end
4894         toloc_save = toloc
4895     end
4896     elseif not inmath and item.id == 7 then
4897         item.replace = item.replace and Babel.locale_map(item.replace)
4898         item.pre      = item.pre and Babel.locale_map(item.pre)
4899         item.post      = item.post and Babel.locale_map(item.post)
4900     elseif item.id == node.id'math' then
4901         inmath = (item.subtype == 0)
4902     end
4903 end
4904 return head
4905 end
4906 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

4907 \newcommand\babelcharproperty[1]{%
4908   \count@=#1\relax
4909   \ifvmode
4910     \expandafter\bbl@chprop
4911   \else
4912     \bbl@error{\string\babelcharproperty\space can be used only in\\%
4913       vertical mode (preamble or between paragraphs)}%
4914     {See the manual for futher info}%
4915   \fi}
4916 \newcommand\bbl@chprop[3][\the\count@]{%
4917   \@tempcnta=#1\relax
4918   \bbl@ifunset{\bbl@chprop@#2}%
4919   {\bbl@error{No property named '#2'. Allowed values are\\%
4920     direction (bc), mirror (bmg), and linebreak (lb)}%

```

```

4921             {See the manual for futher info}}%
4922     {}%
4923 \loop
4924   \bbl@cs{chprop@#2}{#3}%
4925   \ifnum\count@<\@tempcnta
4926     \advance\count@\@ne
4927   \repeat}
4928 \def\bbl@chprop@direction#1{%
4929   \directlua{
4930     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4931     Babel.characters[\the\count@]['d'] = '#1'
4932   }}
4933 \let\bbl@chprop@bc\bbl@chprop@direction
4934 \def\bbl@chprop@mirror#1{%
4935   \directlua{
4936     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4937     Babel.characters[\the\count@]['m'] = '\number#1'
4938   }}
4939 \let\bbl@chprop@bmg\bbl@chprop@mirror
4940 \def\bbl@chprop@linebreak#1{%
4941   \directlua{
4942     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
4943     Babel.cjk_characters[\the\count@]['c'] = '#1'
4944   }}
4945 \let\bbl@chprop@lb\bbl@chprop@linebreak
4946 \def\bbl@chprop@locale#1{%
4947   \directlua{
4948     Babel.chr_to_loc = Babel.chr_to_loc or {}
4949     Babel.chr_to_loc[\the\count@] =
4950       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
4951   }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

4952 \begingroup
4953 \catcode`\#=12
4954 \catcode`\%=12
4955 \catcode`\&=14
4956 \directlua{
4957   Babel.linebreaking.replacements = {}
4958
4959   function Babel.str_to_nodes(fn, matches, base)
4960     local n, head, last
4961     if fn == nil then return nil end
4962     for s in string.utfvalues(fn(matches)) do
4963       if base.id == 7 then
4964         base = base.replace

```

```

4965     end
4966     n = node.copy(base)
4967     n.char = s
4968     if not head then
4969         head = n
4970     else
4971         last.next = n
4972     end
4973     last = n
4974 end
4975 return head
4976 end
4977
4978 function Babel.fetch_word(head, funct)
4979     local word_string = ''
4980     local word_nodes = {}
4981     local lang
4982     local item = head
4983
4984     while item do
4985
4986         if item.id == 29
4987             and not(item.char == 124) && ie, not |
4988             and not(item.char == 61) && ie, not =
4989             and (item.lang == lang or lang == nil) then
4990             lang = lang or item.lang
4991             word_string = word_string .. unicode.utf8.char(item.char)
4992             word_nodes[#word_nodes+1] = item
4993
4994             elseif item.id == 7 and item.subtype == 2 then
4995             word_string = word_string .. '='
4996             word_nodes[#word_nodes+1] = item
4997
4998             elseif item.id == 7 and item.subtype == 3 then
4999             word_string = word_string .. '|'
5000             word_nodes[#word_nodes+1] = item
5001
5002             elseif word_string == '' then
5003                 && pass
5004
5005             else
5006                 return word_string, word_nodes, item, lang
5007             end
5008
5009             item = item.next
5010         end
5011     end
5012
5013 function Babel.post_hyphenate_replace(head)
5014     local u = unicode.utf8
5015     local lbkr = Babel.linebreaking.replacements
5016     local word_head = head
5017
5018     while true do
5019         local w, wn, nw, lang = Babel.fetch_word(word_head)
5020         if not lang then return head end
5021
5022         if not lbkr[lang] then
5023             break

```

```

5024     end
5025
5026     for k=1, #lbr[lang] do
5027         local p = lbr[lang][k].pattern
5028         local r = lbr[lang][k].replace
5029
5030         while true do
5031             local matches = { u.match(w, p) }
5032             if #matches < 2 then break end
5033
5034             local first = table.remove(matches, 1)
5035             local last = table.remove(matches, #matches)
5036
5037             %% Fix offsets, from bytes to unicode.
5038             first = u.len(w:sub(1, first-1)) + 1
5039             last = u.len(w:sub(1, last-1))
5040
5041             local new %% used when inserting and removing nodes
5042             local changed = 0
5043
5044             %% This loop traverses the replace list and takes the
5045             %% corresponding actions
5046             for q = first, last do
5047                 local crep = r[q-first+1]
5048                 local char_node = wn[q]
5049                 local char_base = char_node
5050
5051                 if crep and crep.data then
5052                     char_base = wn[crep.data+first-1]
5053                 end
5054
5055                 if crep == {} then
5056                     break
5057                 elseif crep == nil then
5058                     changed = changed + 1
5059                     node.remove(head, char_node)
5060                 elseif crep and (crep.pre or crep.no or crep.post) then
5061                     changed = changed + 1
5062                     d = node.new(7, 0) %% (disc, discretionary)
5063                     d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5064                     d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5065                     d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5066                     d.attr = char_base.attr
5067                     if crep.pre == nil then %% TeXbook p96
5068                         d.penalty = crep.penalty or tex.hyphenpenalty
5069                     else
5070                         d.penalty = crep.penalty or tex.exhyphenpenalty
5071                     end
5072                     head, new = node.insert_before(head, char_node, d)
5073                     node.remove(head, char_node)
5074                     if q == 1 then
5075                         word_head = new
5076                     end
5077                 elseif crep and crep.string then
5078                     changed = changed + 1
5079                     local str = crep.string(matches)
5080                     if str == '' then
5081                         if q == 1 then
5082                             word_head = char_node.next

```

```

5083         end
5084         head, new = node.remove(head, char_node)
5085     elseif char_node.id == 29 and u.len(str) == 1 then
5086         char_node.char = string.utfvalue(str)
5087     else
5088         local n
5089         for s in string.utfvalues(str) do
5090             if char_node.id == 7 then
5091                 log('Automatic hyphens cannot be replaced, just removed.')
5092             else
5093                 n = node.copy(char_base)
5094             end
5095             n.char = s
5096             if q == 1 then
5097                 head, new = node.insert_before(head, char_node, n)
5098                 word_head = new
5099             else
5100                 node.insert_before(head, char_node, n)
5101             end
5102         end
5103
5104         node.remove(head, char_node)
5105     end    %% string length
5106 end    %% if char and char.string
5107 end    %% for char in match
5108 if changed > 20 then
5109     texio.write('Too many changes. Ignoring the rest.')
5110 elseif changed > 0 then
5111     w, wn, nw = Babel.fetch_word(word_head)
5112 end
5113
5114     end    %% for match
5115 end    %% for patterns
5116 word_head = nw
5117 end    %% for words
5118 return head
5119 end
5120
5121 %% The following functions belong to the next macro
5122
5123 %% This table stores capture maps, numbered consecutively
5124 Babel.capture_maps = {}
5125
5126 function Babel.capture_func(key, cap)
5127     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
5128     ret = ret:gsub('{{[0-9]}|([^|]+)|(-)}', Babel.capture_func_map)
5129     ret = ret:gsub("%[%[%]%%.%.", '')
5130     ret = ret:gsub("%.%[%[%]%%", '')
5131     return key .. [[=function(m) return ]] .. ret .. [[ end]]
5132 end
5133
5134 function Babel.capt_map(from, mapno)
5135     return Babel.capture_maps[mapno][from] or from
5136 end
5137
5138 %% Handle the {n|abc|ABC} syntax in captures
5139 function Babel.capture_func_map(capno, from, to)
5140     local froms = {}
5141     for s in string.utfcharacters(from) do

```



```

5142     table.insert(froms, s)
5143 end
5144 local cnt = 1
5145 table.insert(Babel.capture_maps, {})
5146 local mlen = table.getn(Babel.capture_maps)
5147 for s in string.utfcharacters(to) do
5148     Babel.capture_maps[mlen][froms[cnt]] = s
5149     cnt = cnt + 1
5150 end
5151 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5152     (mlen) .. ").. " .. "["
5153 end
5154
5155 }

```

Now the T<sub>E</sub>X high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the  $\{n\}$  syntax. For example, `pre={1}{1}`- becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua `load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

5156 \catcode`\#=6
5157 \gdef\babelposthyphenation#1#2#3{&%
5158     \bbl@activateposthyphen
5159     \beginngroup
5160     \def\babeltempa{\bbl@add@list\babeltempb}&%
5161     \let\babeltempb\@empty
5162     \bbl@foreach{#3}{&%
5163         \bbl@ifsamestring{##1}{remove}&%
5164         {\bbl@add@list\babeltempb{nil}}&%
5165         {\directlua{
5166             local rep = [[#1]]
5167             rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5168             rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5169             rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5170             rep = rep:gsub( '(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5171             tex.print([[\\string\babeltempa{}}] .. rep .. [[}}]])
5172         }}}&%
5173     \directlua{
5174         local lbkr = Babel.linebreaking.replacements
5175         local u = unicode.utf8
5176         &% Convert pattern:
5177         local patt = string.gsub([[#2]], '%s', '')
5178         if not u.find(patt, '()', nil, true) then
5179             patt = '()' .. patt .. '()'
5180         end
5181         patt = u.gsub(patt, '{(.)}',
5182             function (n)
5183                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5184             end)
5185         lbkr[\\the\\csname l@#1\\endcsname] = lbkr[\\the\\csname l@#1\\endcsname] or {}
5186         table.insert(lbkr[\\the\\csname l@#1\\endcsname],
5187             { pattern = patt, replace = { \babeltempb } })
5188     }&%

```

```

5189 \endgroup}
5190 \endgroup
5191 \def\bbl@activateposthyphen{%
5192   \let\bbl@activateposthyphen\relax
5193   \directlua{
5194     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5195   }}

```

## 13.7 Layout

### Work in progress.

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5196 \bbl@trace{Redefinitions for bidi layout}
5197 \ifx\@eqnnum\@undefined\else
5198   \ifx\bbl@attr@dir\@undefined\else
5199     \edef\@eqnnum{%
5200       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5201       \unexpanded\expandafter{\@eqnnum}}
5202   \fi
5203 \fi
5204 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5205 \ifnum\bbl@bidimode>\z@
5206   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5207     \bbl@exp{%
5208       \mathdir\the\bodydir
5209       #1%           Once entered in math, set boxes to restore values
5210       \<ifmode>%
5211       \everyvbox{%
5212         \the\everyvbox
5213         \bodydir\the\bodydir
5214         \mathdir\the\mathdir
5215         \everyhbox{\the\everyhbox}%
5216         \everyvbox{\the\everyvbox}}%
5217       \everyhbox{%
5218         \the\everyhbox
5219         \bodydir\the\bodydir
5220         \mathdir\the\mathdir
5221         \everyhbox{\the\everyhbox}%
5222         \everyvbox{\the\everyvbox}}%
5223       \<fi>}}%
5224   \def\@hangfrom#1{%
5225     \setbox\@tempboxa\hbox{{#1}}%
5226     \hangindent\wd\@tempboxa
5227     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5228       \shapemode\@ne
5229     \fi

```

```

5230 \noindent\box\@tempboxa}
5231 \fi
5232 \IfBabelLayout{tabular}
5233 {\let\bbl@OL@tabular\@tabular
5234 \bbl@sreplace\@tabular{$}\bbl@nextfake$}%
5235 \let\bbl@NL@tabular\@tabular
5236 \AtBeginDocument{%
5237 \ifx\bbl@NL@tabular\@tabular\else
5238 \bbl@sreplace\@tabular{$}\bbl@nextfake$}%
5239 \let\bbl@NL@tabular\@tabular
5240 \fi}}
5241 {}
5242 \IfBabelLayout{lists}
5243 {\let\bbl@OL@list\list
5244 \bbl@sreplace\list{\parshape}\bbl@listparshape}%
5245 \let\bbl@NL@list\list
5246 \def\bbl@listparshape#1#2#3{%
5247 \parshape #1 #2 #3 %
5248 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5249 \shapemode\tw@
5250 \fi}}
5251 {}
5252 \IfBabelLayout{graphics}
5253 {\let\bbl@pictresetdir\relax
5254 \def\bbl@pictsetdir{%
5255 \ifcase\bbl@thetextdir
5256 \let\bbl@pictresetdir\relax
5257 \else
5258 \textdir TLT\relax
5259 \def\bbl@pictresetdir{\textdir TRT\relax}%
5260 \fi}%
5261 \let\bbl@OL@picture\@picture
5262 \let\bbl@OL@put\put
5263 \bbl@sreplace\@picture{\hskip-}\bbl@pictsetdir\hskip-}%
5264 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5265 \@killglue
5266 \raise#2\unitlength
5267 \hb@xt@z@{\kern#1\unitlength\bbl@pictresetdir#3}\hss}}%
5268 \AtBeginDocument
5269 {\ifx\tikz@atbegin@node\undefined\else
5270 \let\bbl@OL@pgfpicture\pgfpicture
5271 \bbl@sreplace\pgfpicture{\pgfpicturetrue}\bbl@pictsetdir\pgfpicturetrue}%
5272 \bbl@add\pgfsys@beginpicture\bbl@pictsetdir}%
5273 \bbl@add\tikz@atbegin@node\bbl@pictresetdir}%
5274 \fi}}
5275 {}

```

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.

```

5276 \IfBabelLayout{counters}%
5277 {\let\bbl@OL@textsuperscript\textsuperscript
5278 \bbl@sreplace\textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5279 \let\bbl@latinarabic=\@arabic
5280 \let\bbl@OL@arabic\@arabic
5281 \def\@arabic#1{\babelsublr\bbl@latinarabic#1}}%
5282 \@ifpackagewith{babel}{bidi=default}%
5283 {\let\bbl@asciroman=\@roman
5284 \let\bbl@OL@roman\@roman

```

```

5285 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5286 \let\bbl@asciiRoman=\@Roman
5287 \let\bbl@OL@@roman=\@Roman
5288 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5289 \let\bbl@OL@labelenumii\labelenumii
5290 \def\labelenumii{}\theenumii{}%
5291 \let\bbl@OL@p@enumiii\p@enumiii
5292 \def\p@enumiii{\p@enumii}\theenumii{}\{}\}%
5293 <<Footnote changes>>
5294 \IfBabelLayout{footnotes}%
5295 {\let\bbl@OL@footnote\footnote
5296 \BabelFootnote\footnote\languagename{}\}%
5297 \BabelFootnote\localfootnote\languagename{}\}%
5298 \BabelFootnote\mainfootnote{}\%\}%
5299 {}

```

Some  $\LaTeX$  macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5300 \IfBabelLayout{extras}%
5301 {\let\bbl@OL@underline\underline
5302 \bbl@sreplace\underline{\$}\@underline{}\bbl@nextfake$\@underline}%
5303 \let\bbl@OL@LaTeX2e\LaTeX2e
5304 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5305 \if b\expandafter\@car\f@series\@nil\boldmath\fi
5306 \babelsublr{%
5307 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
5308 {}
5309 </luatex>

```

### 13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```

5310 (*basic-r)
5311 Babel = Babel or {}
5312
5313 Babel.bidi_enabled = true
5314
5315 require('babel-data-bidi.lua')
5316
5317 local characters = Babel.characters
5318 local ranges = Babel.ranges
5319
5320 local DIR = node.id("dir")
5321
5322 local function dir_mark(head, from, to, outer)
5323   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5324   local d = node.new(DIR)
5325   d.dir = '+' .. dir
5326   node.insert_before(head, from, d)
5327   d = node.new(DIR)
5328   d.dir = '-' .. dir
5329   node.insert_after(head, to, d)
5330 end
5331
5332 function Babel.bidi(head, ispar)
5333   local first_n, last_n          -- first and last char with nums
5334   local last_es                 -- an auxiliary 'last' used with nums
5335   local first_d, last_d         -- first and last char in L/R block
5336   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):

```

5337   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5338   local strong_lr = (strong == 'l') and 'l' or 'r'
5339   local outer = strong
5340
5341   local new_dir = false
5342   local first_dir = false
5343   local inmath = false
5344
5345   local last_lr
5346
5347   local type_n = ''
5348
5349   for item in node.traverse(head) do

```

```

5350
5351 -- three cases: glyph, dir, otherwise
5352 if item.id == node.id'glyph'
5353   or (item.id == 7 and item.subtype == 2) then
5354
5355     local itemchar
5356     if item.id == 7 and item.subtype == 2 then
5357       itemchar = item.replace.char
5358     else
5359       itemchar = item.char
5360     end
5361     local chardata = characters[itemchar]
5362     dir = chardata and chardata.d or nil
5363     if not dir then
5364       for nn, et in ipairs(ranges) do
5365         if itemchar < et[1] then
5366           break
5367         elseif itemchar <= et[2] then
5368           dir = et[3]
5369           break
5370         end
5371       end
5372     end
5373     dir = dir or 'l'
5374     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).

```

5375   if new_dir then
5376     attr_dir = 0
5377     for at in node.traverse(item.attr) do
5378       if at.number == luatexbase.registernumber'bbl@attr@dir' then
5379         attr_dir = at.value % 3
5380       end
5381     end
5382     if attr_dir == 1 then
5383       strong = 'r'
5384     elseif attr_dir == 2 then
5385       strong = 'al'
5386     else
5387       strong = 'l'
5388     end
5389     strong_lr = (strong == 'l') and 'l' or 'r'
5390     outer = strong_lr
5391     new_dir = false
5392   end
5393
5394   if dir == 'nsm' then dir = strong end -- W1

```

**Numbers.** The dual <al>/<r> system for R is somewhat cumbersome.

```

5395   dir_real = dir -- We need dir_real to set strong below
5396   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:

```

5397   if strong == 'al' then

```

```

5398         if dir == 'en' then dir = 'an' end           -- W2
5399         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5400         strong_lr = 'r'                                -- W3
5401     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

5402     elseif item.id == node.id'dir' and not inmath then
5403         new_dir = true
5404         dir = nil
5405     elseif item.id == node.id'math' then
5406         inmath = (item.subtype == 0)
5407     else
5408         dir = nil          -- Not a char
5409     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>.

```

5410     if dir == 'en' or dir == 'an' or dir == 'et' then
5411         if dir ~= 'et' then
5412             type_n = dir
5413         end
5414         first_n = first_n or item
5415         last_n = last_es or item
5416         last_es = nil
5417     elseif dir == 'es' and last_n then -- W3+W6
5418         last_es = item
5419     elseif dir == 'cs' then           -- it's right - do nothing
5420     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5421         if strong_lr == 'r' and type_n ~= '' then
5422             dir_mark(head, first_n, last_n, 'r')
5423         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5424             dir_mark(head, first_n, last_n, 'r')
5425             dir_mark(head, first_d, last_d, outer)
5426             first_d, last_d = nil, nil
5427         elseif strong_lr == 'l' and type_n ~= '' then
5428             last_d = last_n
5429         end
5430         type_n = ''
5431         first_n, last_n = nil, nil
5432     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:

```

5433     if dir == 'l' or dir == 'r' then
5434         if dir ~= outer then
5435             first_d = first_d or item
5436             last_d = item
5437         elseif first_d and dir ~= strong_lr then
5438             dir_mark(head, first_d, last_d, outer)
5439             first_d, last_d = nil, nil
5440         end
5441     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.

```

5442   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5443       item.char = characters[item.char] and
5444           characters[item.char].m or item.char
5445   elseif (dir or new_dir) and last_lr ~= item then
5446       local mir = outer .. strong_lr .. (dir or outer)
5447       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5448           for ch in node.traverse(node.next(last_lr)) do
5449               if ch == item then break end
5450               if ch.id == node.id'glyph' and characters[ch.char] then
5451                   ch.char = characters[ch.char].m or ch.char
5452               end
5453           end
5454       end
5455   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).

```

5456   if dir == 'l' or dir == 'r' then
5457       last_lr = item
5458       strong = dir_real          -- Don't search back - best save now
5459       strong_lr = (strong == 'l') and 'l' or 'r'
5460   elseif new_dir then
5461       last_lr = nil
5462   end
5463 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

5464   if last_lr and outer == 'r' then
5465       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5466           if characters[ch.char] then
5467               ch.char = characters[ch.char].m or ch.char
5468           end
5469       end
5470   end
5471   if first_n then
5472       dir_mark(head, first_n, last_n, outer)
5473   end
5474   if first_d then
5475       dir_mark(head, first_d, last_d, outer)
5476   end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

5477   return node.prev(head) or head
5478 end
5479 </basic-r>

```

And here the Lua code for bidi=basic:

```

5480 <(*basic)
5481 Babel = Babel or {}
5482
5483 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5484

```



```

5485 Babel.fontmap = Babel.fontmap or {}
5486 Babel.fontmap[0] = {}      -- l
5487 Babel.fontmap[1] = {}      -- r
5488 Babel.fontmap[2] = {}      -- al/an
5489
5490 Babel.bidi_enabled = true
5491 Babel.mirroring_enabled = true
5492
5493 require('babel-data-bidi.lua')
5494
5495 local characters = Babel.characters
5496 local ranges = Babel.ranges
5497
5498 local DIR = node.id('dir')
5499 local GLYPH = node.id('glyph')
5500
5501 local function insert_implicit(head, state, outer)
5502   local new_state = state
5503   if state.sim and state.eim and state.sim ~= state.eim then
5504     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5505     local d = node.new(DIR)
5506     d.dir = '+' .. dir
5507     node.insert_before(head, state.sim, d)
5508     local d = node.new(DIR)
5509     d.dir = '-' .. dir
5510     node.insert_after(head, state.eim, d)
5511   end
5512   new_state.sim, new_state.eim = nil, nil
5513   return head, new_state
5514 end
5515
5516 local function insert_numeric(head, state)
5517   local new
5518   local new_state = state
5519   if state.san and state.ean and state.san ~= state.ean then
5520     local d = node.new(DIR)
5521     d.dir = '+TLT'
5522     _, new = node.insert_before(head, state.san, d)
5523     if state.san == state.sim then state.sim = new end
5524     local d = node.new(DIR)
5525     d.dir = '-TLT'
5526     _, new = node.insert_after(head, state.ean, d)
5527     if state.ean == state.eim then state.eim = new end
5528   end
5529   new_state.san, new_state.ean = nil, nil
5530   return head, new_state
5531 end
5532
5533 -- TODO - \hbox with an explicit dir can lead to wrong results
5534 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5535 -- was s made to improve the situation, but the problem is the 3-dir
5536 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5537 -- well.
5538
5539 function Babel.bidi(head, ispar, hdir)
5540   local d -- d is used mainly for computations in a loop
5541   local prev_d = ''
5542   local new_d = false
5543

```

```

5544 local nodes = {}
5545 local outer_first = nil
5546 local inmath = false
5547
5548 local glue_d = nil
5549 local glue_i = nil
5550
5551 local has_en = false
5552 local first_et = nil
5553
5554 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5555
5556 local save_outer
5557 local temp = node.get_attribute(head, ATDIR)
5558 if temp then
5559     temp = temp % 3
5560     save_outer = (temp == 0 and 'l') or
5561                  (temp == 1 and 'r') or
5562                  (temp == 2 and 'al')
5563 elseif ispar then -- Or error? Shouldn't happen
5564     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5565 else -- Or error? Shouldn't happen
5566     save_outer = ('TRT' == hdir) and 'r' or 'l'
5567 end
5568 -- when the callback is called, we are just _after_ the box,
5569 -- and the textdir is that of the surrounding text
5570 -- if not ispar and hdir ~= tex.textdir then
5571 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5572 -- end
5573 local outer = save_outer
5574 local last = outer
5575 -- 'al' is only taken into account in the first, current loop
5576 if save_outer == 'al' then save_outer = 'r' end
5577
5578 local fontmap = Babel.fontmap
5579
5580 for item in node.traverse(head) do
5581
5582     -- In what follows, #node is the last (previous) node, because the
5583     -- current one is not added until we start processing the neutrals.
5584
5585     -- three cases: glyph, dir, otherwise
5586     if item.id == GLYPH
5587         or (item.id == 7 and item.subtype == 2) then
5588
5589         local d_font = nil
5590         local item_r
5591         if item.id == 7 and item.subtype == 2 then
5592             item_r = item.replace -- automatic discs have just 1 glyph
5593         else
5594             item_r = item
5595         end
5596         local chardata = characters[item_r.char]
5597         d = chardata and chardata.d or nil
5598         if not d or d == 'nsm' then
5599             for nn, et in ipairs(ranges) do
5600                 if item_r.char < et[1] then
5601                     break
5602                 elseif item_r.char <= et[2] then

```

```

5603         if not d then d = et[3]
5604         elseif d == 'nsm' then d_font = et[3]
5605         end
5606         break
5607     end
5608 end
5609 end
5610 d = d or 'l'
5611
5612 -- A short 'pause' in bidi for mapfont
5613 d_font = d_font or d
5614 d_font = (d_font == 'l' and 0) or
5615           (d_font == 'nsm' and 0) or
5616           (d_font == 'r' and 1) or
5617           (d_font == 'al' and 2) or
5618           (d_font == 'an' and 2) or nil
5619 if d_font and fontmap and fontmap[d_font][item_r.font] then
5620     item_r.font = fontmap[d_font][item_r.font]
5621 end
5622
5623 if new_d then
5624     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5625     if inmath then
5626         attr_d = 0
5627     else
5628         attr_d = node.get_attribute(item, ATDIR)
5629         attr_d = attr_d % 3
5630     end
5631     if attr_d == 1 then
5632         outer_first = 'r'
5633         last = 'r'
5634     elseif attr_d == 2 then
5635         outer_first = 'r'
5636         last = 'al'
5637     else
5638         outer_first = 'l'
5639         last = 'l'
5640     end
5641     outer = last
5642     has_en = false
5643     first_et = nil
5644     new_d = false
5645 end
5646
5647 if glue_d then
5648     if (d == 'l' and 'l' or 'r') ~= glue_d then
5649         table.insert(nodes, {glue_i, 'on', nil})
5650     end
5651     glue_d = nil
5652     glue_i = nil
5653 end
5654
5655 elseif item.id == DIR then
5656     d = nil
5657     new_d = true
5658
5659 elseif item.id == node.id'glue' and item.subtype == 13 then
5660     glue_d = d
5661     glue_i = item

```

```

5662     d = nil
5663
5664     elseif item.id == node.id'math' then
5665         inmath = (item.subtype == 0)
5666
5667     else
5668         d = nil
5669     end
5670
5671     -- AL <= EN/ET/ES      -- W2 + W3 + W6
5672     if last == 'al' and d == 'en' then
5673         d = 'an'          -- W3
5674     elseif last == 'al' and (d == 'et' or d == 'es') then
5675         d = 'on'          -- W6
5676     end
5677
5678     -- EN + CS/ES + EN      -- W4
5679     if d == 'en' and #nodes >= 2 then
5680         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5681             and nodes[#nodes-1][2] == 'en' then
5682             nodes[#nodes][2] = 'en'
5683         end
5684     end
5685
5686     -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
5687     if d == 'an' and #nodes >= 2 then
5688         if (nodes[#nodes][2] == 'cs')
5689             and nodes[#nodes-1][2] == 'an' then
5690             nodes[#nodes][2] = 'an'
5691         end
5692     end
5693
5694     -- ET/EN                -- W5 + W7->1 / W6->on
5695     if d == 'et' then
5696         first_et = first_et or (#nodes + 1)
5697     elseif d == 'en' then
5698         has_en = true
5699         first_et = first_et or (#nodes + 1)
5700     elseif first_et then    -- d may be nil here !
5701         if has_en then
5702             if last == 'l' then
5703                 temp = 'l'    -- W7
5704             else
5705                 temp = 'en'   -- W5
5706             end
5707         else
5708             temp = 'on'       -- W6
5709         end
5710         for e = first_et, #nodes do
5711             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5712         end
5713         first_et = nil
5714         has_en = false
5715     end
5716
5717     if d then
5718         if d == 'al' then
5719             d = 'r'
5720             last = 'al'

```

```

5721     elseif d == 'l' or d == 'r' then
5722         last = d
5723     end
5724     prev_d = d
5725     table.insert(nodes, {item, d, outer_first})
5726 end
5727
5728 outer_first = nil
5729
5730 end
5731
5732 -- TODO -- repeated here in case EN/ET is the last node. Find a
5733 -- better way of doing things:
5734 if first_et then      -- dir may be nil here !
5735     if has_en then
5736         if last == 'l' then
5737             temp = 'l'    -- W7
5738         else
5739             temp = 'en'   -- W5
5740         end
5741     else
5742         temp = 'on'      -- W6
5743     end
5744     for e = first_et, #nodes do
5745         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5746     end
5747 end
5748
5749 -- dummy node, to close things
5750 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5751
5752 ----- NEUTRAL -----
5753
5754 outer = save_outer
5755 last = outer
5756
5757 local first_on = nil
5758
5759 for q = 1, #nodes do
5760     local item
5761
5762     local outer_first = nodes[q][3]
5763     outer = outer_first or outer
5764     last = outer_first or last
5765
5766     local d = nodes[q][2]
5767     if d == 'an' or d == 'en' then d = 'r' end
5768     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5769
5770     if d == 'on' then
5771         first_on = first_on or q
5772     elseif first_on then
5773         if last == d then
5774             temp = d
5775         else
5776             temp = outer
5777         end
5778         for r = first_on, q - 1 do
5779             nodes[r][2] = temp

```

```

5780         item = nodes[r][1]    -- MIRRORING
5781         if Babel.mirroring_enabled and item.id == GLYPH
5782             and temp == 'r' and characters[item.char] then
5783             local font_mode = font.fonts[item.font].properties.mode
5784             if font_mode ~= 'harf' and font_mode ~= 'plug' then
5785                 item.char = characters[item.char].m or item.char
5786             end
5787         end
5788     end
5789     first_on = nil
5790 end
5791
5792     if d == 'r' or d == 'l' then last = d end
5793 end
5794
5795 ----- IMPLICIT, REORDER -----
5796
5797 outer = save_outer
5798 last = outer
5799
5800 local state = {}
5801 state.has_r = false
5802
5803 for q = 1, #nodes do
5804
5805     local item = nodes[q][1]
5806
5807     outer = nodes[q][3] or outer
5808
5809     local d = nodes[q][2]
5810
5811     if d == 'nsm' then d = last end          -- W1
5812     if d == 'en' then d = 'an' end
5813     local isdir = (d == 'r' or d == 'l')
5814
5815     if outer == 'l' and d == 'an' then
5816         state.san = state.san or item
5817         state.ean = item
5818     elseif state.san then
5819         head, state = insert_numeric(head, state)
5820     end
5821
5822     if outer == 'l' then
5823         if d == 'an' or d == 'r' then      -- im -> implicit
5824             if d == 'r' then state.has_r = true end
5825             state.sim = state.sim or item
5826             state.eim = item
5827         elseif d == 'l' and state.sim and state.has_r then
5828             head, state = insert_implicit(head, state, outer)
5829         elseif d == 'l' then
5830             state.sim, state.eim, state.has_r = nil, nil, false
5831         end
5832     else
5833         if d == 'an' or d == 'l' then
5834             if nodes[q][3] then -- nil except after an explicit dir
5835                 state.sim = item -- so we move sim 'inside' the group
5836             else
5837                 state.sim = state.sim or item
5838             end
5839         end
5840     end
5841 end

```

```

5839         state.eim = item
5840     elseif d == 'r' and state.sim then
5841         head, state = insert_implicit(head, state, outer)
5842     elseif d == 'r' then
5843         state.sim, state.eim = nil, nil
5844     end
5845 end
5846
5847 if isdir then
5848     last = d          -- Don't search back - best save now
5849 elseif d == 'on' and state.san then
5850     state.san = state.san or item
5851     state.ean = item
5852 end
5853
5854 end
5855
5856 return node.prev(head) or head
5857 end
5858 </basic>

```

## 14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},

```

For the meaning of these codes, see the Unicode standard.

## 15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```

5859 <nil>
5860 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
5861 \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.

```

5862 \ifx\l@nil\@undefined
5863   \newlanguage\l@nil
5864   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
5865   \let\bbl@elt\relax
5866   \edef\bbl@languages{% Add it to the list of languages
5867     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}
5868 \fi

```

5869 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```
\captionnil
\datenil 5870 \let\captionnil\@empty
5871 \let\datenil\@empty
```

```
5872 \ldf@finish{nil}
5873 \</nil>
```



```

5885 \let\@undefined
5886 }
5887 \fi
5888 </bplain | bplain>

```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```

5889 <bplain>\a plain.tex
5890 <bplain>\a 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.

```

5891 <bplain>\def\fmtname{babel-plain}
5892 <bplain>\def\fmtname{babel-lplain}

```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

## 16.2 Emulating some $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```

5893 <<*Emulate LaTeX>> ≡
5894 % == Code for plain ==
5895 \def\@empty{}
5896 \def\loadlocalcfg#1{%
5897   \openin0#1.cfg
5898   \ifeof0
5899     \closein0
5900   \else
5901     \closein0
5902     {\immediate\write16{*****}%
5903      \immediate\write16{* Local config file #1.cfg used}%
5904      \immediate\write16{*}%
5905     }
5906     \input #1.cfg\relax
5907   \fi
5908 \endofldef}

```

## 16.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```

5909 \long\def\@firstofone#1{#1}
5910 \long\def\@firstoftwo#1#2{#1}
5911 \long\def\@secondoftwo#1#2{#2}
5912 \def\@nnil{\@nil}
5913 \def\@gobbletwo#1#2{}
5914 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
5915 \def\@star@or@long#1{%
5916   \@ifstar
5917   {\let\l@ngrel@x\relax#1}%
5918   {\let\l@ngrel@x\long#1}}
5919 \let\l@ngrel@x\relax
5920 \def\@car#1#2\@nil{#1}
5921 \def\@cdr#1#2\@nil{#2}
5922 \let\@typeset@protect\relax
5923 \let\protected@edef\edef
5924 \long\def\@gobble#1{}

```

```

5925 \edef\@backslashchar{\expandafter\@gobble\string\}
5926 \def\strip@prefix#1>{}
5927 \def\g@addto@macro#1#2{%
5928     \toks@\expandafter{#1#2}%
5929     \xdef#1{\the\toks@}}
5930 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
5931 \def\@nameuse#1{\csname #1\endcsname}
5932 \def\@ifundefined#1{%
5933     \expandafter\ifx\csname#1\endcsname\relax
5934     \expandafter\@firstoftwo
5935     \else
5936     \expandafter\@secondoftwo
5937     \fi}
5938 \def\@expandtwoargs#1#2#3{%
5939     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
5940 \def\zap@space#1 #2{%
5941     #1%
5942     \ifx#2\@empty\else\expandafter\zap@space\fi
5943     #2}
5944 \let\bbl@trace\@gobble

```

$\LaTeX_2\epsilon$  has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

5945 \ifx\@preamblecmds\@undefined
5946     \def\@preamblecmds{}
5947 \fi
5948 \def\@onlypreamble#1{%
5949     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
5950         \@preamblecmds\do#1}}
5951 \@onlypreamble\@onlypreamble

```

Mimick  $\LaTeX$ 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

5952 \def\begindocument{%
5953     \@begindocumenthook
5954     \global\let\@begindocumenthook\@undefined
5955     \def\do##1{\global\let##1\@undefined}%
5956     \@preamblecmds
5957     \global\let\do\noexpand}
5958 \ifx\@begindocumenthook\@undefined
5959     \def\@begindocumenthook{}
5960 \fi
5961 \@onlypreamble\@begindocumenthook
5962 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick  $\LaTeX$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

5963 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
5964 \@onlypreamble\AtEndOfPackage
5965 \def\@endoflfd{}
5966 \@onlypreamble\@endoflfd
5967 \let\bbl@afterlang\@empty
5968 \chardef\bbl@opt@hyphenmap\z@

```

$\LaTeX$  needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

5969 \catcode`\&=\z@

```

```

5970 \ifx&if@filesw\@undefined
5971 \expandafter\let\csname if@filesw\expandafter\endcsname
5972 \csname iffalse\endcsname
5973 \fi
5974 \catcode\&=4

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

5975 \def\newcommand{\@star@or@long\new@command}
5976 \def\new@command#1{%
5977 \@testopt{\@newcommand#1}0}
5978 \def\@newcommand#1[#2]{%
5979 \@ifnextchar [{\@xargdef#1[#2]]%
5980 {\@argdef#1[#2]}}
5981 \long\def\@argdef#1[#2]#3{%
5982 \@yargdef#1\@ne{#2}{#3}}
5983 \long\def\@xargdef#1[#2][#3]#4{%
5984 \expandafter\def\expandafter#1\expandafter{%
5985 \expandafter\@protected@testopt\expandafter #1%
5986 \csname\string#1\expandafter\endcsname{#3}}%
5987 \expandafter\@yargdef \csname\string#1\endcsname
5988 \tw@{#2}{#4}}
5989 \long\def\@yargdef#1#2#3{%
5990 \@tempcnta#3\relax
5991 \advance \@tempcnta \@ne
5992 \let\@hash@\relax
5993 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
5994 \@tempcntb #2%
5995 \@whilenum\@tempcntb <\@tempcnta
5996 \do{%
5997 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
5998 \advance\@tempcntb \@ne}%
5999 \let\@hash@###
6000 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6001 \def\providecommand{\@star@or@long\provide@command}
6002 \def\provide@command#1{%
6003 \begingroup
6004 \escapechar\m@ne\def\@gtempa{{\string#1}}%
6005 \endgroup
6006 \expandafter\@ifundefined\@gtempa
6007 {\def\reserved@a{\new@command#1}}%
6008 {\let\reserved@a\relax
6009 \def\reserved@a{\new@command\reserved@a}}%
6010 \reserved@a}%
6011 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6012 \def\declare@robustcommand#1{%
6013 \edef\reserved@a{\string#1}%
6014 \def\reserved@b{#1}%
6015 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6016 \edef#1{%
6017 \ifx\reserved@a\reserved@b
6018 \noexpand\x@protect
6019 \noexpand#1%
6020 \fi
6021 \noexpand\protect
6022 \expandafter\noexpand\csname
6023 \expandafter\@gobble\string#1 \endcsname
6024 }%
6025 \expandafter\new@command\csname
6026 \expandafter\@gobble\string#1 \endcsname

```

```

6027 }
6028 \def\x@protect#1{%
6029   \ifx\protect\@typeset@protect\else
6030     \x@protect#1%
6031   \fi
6032 }
6033 \catcode`\&=\z@ % Trick to hide conditionals
6034 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

6035 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6036 \catcode`\&=4
6037 \ifx\in@\undefined
6038   \def\in@#1#2{%
6039     \def\in@##1#1##2##3\in@{%
6040       \ifx\in@##2\in@false\else\in@true\fi}%
6041     \in@@#2#1\in@\in@}
6042 \else
6043   \let\bbl@tempa\empty
6044 \fi
6045 \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).

```

6046 \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.

```

6047 \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<sub>ε</sub> versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```

6048 \ifx\@tempcnta\undefined
6049   \csname newcount\endcsname\@tempcnta\relax
6050 \fi
6051 \ifx\@tempcntb\undefined
6052   \csname newcount\endcsname\@tempcntb\relax
6053 \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`).

```

6054 \ifx\bye\undefined
6055   \advance\count10 by -2\relax
6056 \fi
6057 \ifx\@ifnextchar\undefined
6058   \def\@ifnextchar#1#2#3{%
6059     \let\reserved@d=#1%
6060     \def\reserved@a{#2}\def\reserved@b{#3}%
6061     \futurelet\@let@token\@ifnch}
6062   \def\@ifnch{%

```

```

6063 \ifx\@let@token\@sptoken
6064 \let\reserved@c\@xifnch
6065 \else
6066 \ifx\@let@token\reserved@
6067 \let\reserved@c\reserved@a
6068 \else
6069 \let\reserved@c\reserved@b
6070 \fi
6071 \fi
6072 \reserved@c}
6073 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6074 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6075 \fi
6076 \def\@testopt#1#2{%
6077 \@ifnextchar[#{1}{#1[#2]}}
6078 \def\@protected@testopt#1{%
6079 \ifx\protect\@typeset@protect
6080 \expandafter\@testopt
6081 \else
6082 \@x@protect#1%
6083 \fi}
6084 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6085 #2\relax}\fi}
6086 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6087 \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\TeX$  environment.

```

6088 \def\DeclareTextCommand{%
6089 \@dec@text@cmd\providecommand
6090 }
6091 \def\ProvideTextCommand{%
6092 \@dec@text@cmd\providecommand
6093 }
6094 \def\DeclareTextSymbol#1#2#3{%
6095 \@dec@text@cmd\chardef#1{#2}#3\relax
6096 }
6097 \def\@dec@text@cmd#1#2#3{%
6098 \expandafter\def\expandafter#2%
6099 \expandafter{%
6100 \csname#3-cmd\expandafter\endcsname
6101 \expandafter#2%
6102 \csname#3\string#2\endcsname
6103 }%
6104 % \let\@ifdefinable\@rc@ifdefinable
6105 \expandafter#1\csname#3\string#2\endcsname
6106 }
6107 \def\@current@cmd#1{%
6108 \ifx\protect\@typeset@protect\else
6109 \noexpand#1\expandafter\@gobble
6110 \fi
6111 }
6112 \def\@changed@cmd#1#2{%
6113 \ifx\protect\@typeset@protect
6114 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6115 \expandafter\ifx\csname ?\string#1\endcsname\relax
6116 \expandafter\def\csname ?\string#1\endcsname{%

```

```

6117         \@changed@x@err{#1}%
6118     }%
6119 \fi
6120 \global\expandafter\let
6121 \csname\cf@encoding\string#1\expandafter\endcsname
6122 \csname ?\string#1\endcsname
6123 \fi
6124 \csname\cf@encoding\string#1%
6125 \expandafter\endcsname
6126 \else
6127     \noexpand#1%
6128 \fi
6129 }
6130 \def\@changed@x@err#1{%
6131     \errhelp{Your command will be ignored, type <return> to proceed}%
6132     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6133 \def\DeclareTextCommandDefault#1{%
6134     \DeclareTextCommand#1?%
6135 }
6136 \def\ProvideTextCommandDefault#1{%
6137     \ProvideTextCommand#1?%
6138 }
6139 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6140 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6141 \def\DeclareTextAccent#1#2#3{%
6142     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6143 }
6144 \def\DeclareTextCompositeCommand#1#2#3#4{%
6145     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6146     \edef\reserved@b{\string##1}%
6147     \edef\reserved@c{%
6148         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6149     \ifx\reserved@b\reserved@c
6150         \expandafter\expandafter\expandafter\ifx
6151             \expandafter\@car\reserved@a\relax\relax\@nil
6152             \@text@composite
6153     \else
6154         \edef\reserved@b##1{%
6155             \def\expandafter\noexpand
6156             \csname#2\string#1\endcsname####1{%
6157                 \noexpand\@text@composite
6158                 \expandafter\noexpand\csname#2\string#1\endcsname
6159                 ####1\noexpand\@empty\noexpand\@text@composite
6160                 {##1}%
6161             }%
6162         }%
6163         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6164     \fi
6165     \expandafter\def\csname\expandafter\string\csname
6166         #2\endcsname\string#1-\string#3\endcsname{#4}
6167 \else
6168     \errhelp{Your command will be ignored, type <return> to proceed}%
6169     \errmessage{\string\DeclareTextCompositeCommand\space used on
6170         inappropriate command \protect#1}
6171 \fi
6172 }
6173 \def\@text@composite#1#2#3\@text@composite{%
6174     \expandafter\@text@composite@x
6175     \csname\string#1-\string#2\endcsname

```

```

6176 }
6177 \def\@text@composite@x#1#2{%
6178     \ifx#1\relax
6179         #2%
6180     \else
6181         #1%
6182     \fi
6183 }
6184 %
6185 \def\@strip@args#1:#2-#3\@strip@args{#2}
6186 \def\DeclareTextComposite#1#2#3#4{%
6187     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6188     \bgroup
6189         \lccode`\@=#4%
6190         \lowercase{%
6191     \egroup
6192         \reserved@a @%
6193     }%
6194 }
6195 %
6196 \def\UseTextSymbol#1#2{%
6197 %     \let\@curr@enc\cf@encoding
6198 %     \@use@text@encoding{#1}%
6199     #2%
6200 %     \@use@text@encoding\@curr@enc
6201 }
6202 \def\UseTextAccent#1#2#3{%
6203 %     \let\@curr@enc\cf@encoding
6204 %     \@use@text@encoding{#1}%
6205 %     #2{\@use@text@encoding\@curr@enc\selectfont#3}%
6206 %     \@use@text@encoding\@curr@enc
6207 }
6208 \def\@use@text@encoding#1{%
6209 %     \edef\font@name{#1}%
6210 %     \xdef\font@name{%
6211 %         \csname\curr@fontshape/\font@size\endcsname
6212 %     }%
6213 %     \pickup@font
6214 %     \font@name
6215 %     \@@enc@update
6216 }
6217 \def\DeclareTextSymbolDefault#1#2{%
6218     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6219 }
6220 \def\DeclareTextAccentDefault#1#2{%
6221     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6222 }
6223 \def\cf@encoding{OT1}

```

Currently we only use the  $\text{\LaTeX 2}_{\epsilon}$  method for accents for those that are known to be made active in *some* language definition file.

```

6224 \DeclareTextAccent{"}{OT1}{127}
6225 \DeclareTextAccent{'}{OT1}{19}
6226 \DeclareTextAccent{^}{OT1}{94}
6227 \DeclareTextAccent{\`}{OT1}{18}
6228 \DeclareTextAccent{\~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN  $\text{\TeX}$ .

```

6229 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}

```

```

6230 \DeclareTextSymbol{\textquotedblright}{OT1}{}`"}
6231 \DeclareTextSymbol{\textquoteleft}{OT1}{}`'
6232 \DeclareTextSymbol{\textquoteright}{OT1}{}`'}
6233 \DeclareTextSymbol{\i}{OT1}{16}
6234 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\text{\LaTeX}$ -control sequence `\scriptsize` to be available. Because plain  $\text{\TeX}$  doesn't have such a sophisticated font mechanism as  $\text{\LaTeX}$  has, we just `\let` it to `\sevenrm`.

```

6235 \ifx\scriptsize\@undefined
6236   \let\scriptsize\sevenrm
6237 \fi
6238 % End of code for plain
6239 <</Emulate LaTeX>>

```

A proxy file:

```

6240 <*plain>
6241 \input babel.def
6242 </plain>

```

## 17 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitsluijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tzolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).