Babel

Version 3.15
2017/11/03

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

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

This manual describes babel, a package that makes use of the capabilities of TeX version 3 and, to some extent, xetex and luatex, to provide an environment in which documents can be typeset in a language other than US English, or in more than one language or script.

Current development is focused on Unicode engines (XeTeX and LuaTeX) and the so-called *complex scripts*. New features related to font selection, bidi writing and the like will be added incrementally.

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

Contents

I User guide 4

1 The user interface 4
- 1.1 Monolingual documents 4
- 1.2 Multilingual documents 5
- 1.3 Modifiers 6
- 1.4 xelatex and lualatex 6
- 1.5 Troubleshooting 7
- 1.6 Plain 7
- 1.7 Basic language selectors 8
- 1.8 Auxiliary language selectors 8
- 1.9 More on selection 9
- 1.10 Shorthands 10
- 1.11 Package options 14
- 1.12 The base option 16
- 1.13 ini files 16
- 1.14 Selecting fonts 22
- 1.15 Modifying a language 24
- 1.16 Creating a language 24
- 1.17 Getting the current language name 26
- 1.18 Hooks 26
- 1.19 Hyphenation tools 28
- 1.20 Selecting scripts 29
- 1.21 Selecting directions 30
- 1.22 Language attributes 30
- 1.23 Languages supported by babel 30
- 1.24 Tips, workarounds, know issues and notes 32
- 1.25 Current and future work 33
- 1.26 Tentative and experimental code 34

2 Loading languages with language.dat 34
- 2.1 Format 35

3 The interface between the core of babel and the language definition files 36
- 3.1 Basic macros 37
- 3.2 Skeleton 38
- 3.3 Support for active characters 39
- 3.4 Support for saving macro definitions 39
- 3.5 Support for extending macros 40
- 3.6 Macros common to a number of languages 40
- 3.7 Encoding-dependent strings 40

4 Changes 44
- 4.1 Changes in babel version 3.9 44
- 4.2 Changes in babel version 3.7 44

II The code 45

5 Identification and loading of required files 45

6 Tools 45
- 6.1 Multiple languages 49

7 The Package File (LaTeX, `babel.sty`) 50

7.1 `base` . . . 50

7.2 `key=value` options and other general option . . . 51

7.3 Conditional loading of shorthands . . . 53

7.4 Language options . . . 54

8 The kernel of Babel (`babel.def`, common) 57

8.1 Tools . . . 57

8.2 Hooks . . . 59

8.3 Setting up language files . . . 61

8.4 Shorthands . . . 62

8.5 Language attributes . . . 72

8.6 Support for saving macro definitions . . . 74

8.7 Short tags . . . 75

8.8 Hyphens . . . 75

8.9 Multiencoding strings . . . 77

8.10 Macros common to a number of languages . . . 82

8.11 Making glyphs available . . . 83

8.11.1 Quotation marks . . . 83

8.11.2 Letters . . . 84

8.11.3 Shorthands for quotation marks . . . 85

8.11.4 Umlauts and tremas . . . 86

9 The kernel of Babel (`babel.def`, only LaTeX) 87

9.1 The redefinition of the style commands . . . 87

9.2 Creating languages . . . 88

9.3 Cross referencing macros . . . 93

9.4 Marks . . . 97

9.5 Preventing clashes with other packages . . . 97

9.5.1 `ifthen` . . . 97

9.5.2 `varioref` . . . 98

9.5.3 `hhline` . . . 99

9.5.4 `hyperref` . . . 99

9.5.5 `fancyhdr` . . . 99

9.6 Encoding and fonts . . . 100

9.7 Basic bidi support . . . 101

9.8 Layout . . . 104

9.9 Local Language Configuration . . . 104

10 Multiple languages (`switch.def`) 105

10.1 Selecting the language . . . 106

10.2 Errors . . . 113

11 Loading hyphenation patterns 115

12 Font handling with fontspec 120

13 Hooks for XeTeX and LuaTeX 122

13.1 XeTeX . . . 122

13.2 Layout . . . 123

13.3 LuaTeX . . . 126

14 Bidi support in luatex 131

14.1 Layout . . . 131

14.2 Auto bidi with `basic-r` . . . 132

15 The ‘nil’ language 136

16 Support for Plain TEX (`plain.def`) 137

16.1 Not renaming `hyphen.tex` 137

16.2 Emulating some LATEX features 138

16.3 General tools 139

16.4 Encoding related macros 142

16.5 Babel options 145

17 Acknowledgements 146

Part I

User guide

This user guide focuses on LaTeX. There are also some notes on its use with Plain TeX. If you are interested in the TeX multilingual support, please join the kadingira list on `http://tug.org/mailman/listinfo/kadingira`.

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 standand mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings.

EXAMPLE Here is a simple full example for “traditional” TeX engines (see below for xetex and luatex). The packages `fontenc` and `inputenc` do not belong to babel, but they are included in the example because typically you will need them:

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

WARNING A common source of trouble is a wrong setting of the input encoding. 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.

1.2 Multilingual documents

In multilingual documents, just use several options. The last one is considered the main language, 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:

```
\documentclass{article}
\usepackage[main=english,dutch]{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 `\languagename` (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, decribed 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.

```
\documentclass{article}

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

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

1.3 Modifiers

New 3.9c The basic behaviour 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 accept them). An example is (spaces are not significant and they can be added or removed):[1]

```
\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 is a more general mechanism.

1.4 xelatex and lualatex

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

The Latin script is covered by default in current 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.

EXAMPLE The following bilingual, single script document in UTF-8 encoding just prints a couple of 'captions' and `\today` in Danish and Vietnamese. No additional packages are required.

```
\documentclass{article}
\usepackage[vietnamese,danish]{babel}
\begin{document}
\prefacename{} -- \alsoname{} -- \today
\selectlanguage{vietnamese}
\prefacename{} -- \alsoname{} -- \today
\end{document}
```

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

```
\documentclass{article}
\usepackage[russian]{babel}
\babelfont{rm}{DejaVu Serif}
\begin{document}
Россия, находящаяся на пересечении множества культур, а также
```

[1] No predefined "axis" for modifiers are provided because languages and their scripts have quite different needs.

```
   с учётом многонационального характера её населения, — отличается
   высокой степенью этнокультурного многообразия и способностью к
   межкультурному диалогу.

   \end{document}
```

1.5 Troubleshooting

- Loading directly `sty` files in LaTeX (ie, `\usepackage{`⟨*language*⟩`}`) is deprecated and you will get the error:[2]

```
   ! 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:[3]

```
   ! Package babel Error: Unknown language `LANG'. Either you have misspelled
   (babel)                its name, it has not been installed, or you requested
   (babel)                it in a previous run. Fix its name, install it or just
   (babel)                rerun the file, respectively
```

  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.

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

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

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

1.6 Plain

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

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

[2] In old versions the error read “You have used an old interface to call babel”, not very helpful.

[3] In old versions the error read “You haven’t loaded the language LANG yet”.

WARNING Not all languages provide a `sty` file and some of them are not compatible with Plain.[4]

1.7 Basic language selectors

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

`\selectlanguage` {⟨*language*⟩}

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

`\foreignlanguage` {⟨*language*⟩}{⟨*text*⟩}

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

1.8 Auxiliary language selectors

`\begin{otherlanguage}` {⟨*language*⟩} … `\end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\selectlanguage`, except the language change is (mostly) local to the environment.
Actually, there might be some non-local changes, as this environment is roughly equivalent to:

[4] Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues will be fixed soon.

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

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

`\begin{otherlanguage*}` {⟨*language*⟩} ... `\end{otherlanguage*}`

Same as `\foreignlanguage` but as environment. Spaces after the environment are *not* ignored.
This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented behaviour and it is just a version as environment of `\foreignlanguage`.

`\begin{hyphenrules}` {⟨*language*⟩} ... `\end{hyphenrules}`

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

1.9 More on selection

`\babeltags` {⟨*tag1*⟩ = ⟨*language1*⟩, ⟨*tag2*⟩ = ⟨*language2*⟩, ...}

New 3.9i In multilingual documents with many language switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.
It defines `\text`⟨*tag1*⟩`{`⟨*text*⟩`}` to be `\foreignlanguage{`⟨*language1*⟩`}{`⟨*text*⟩`}`, and `\begin{`⟨*tag1*⟩`}` to be `\begin{otherlanguage*}{`⟨*language1*⟩`}`, and so on. Note `\`⟨*tag1*⟩ is also allowed, but remember to set it locally inside a group.

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary TEX code.
Shorthands can be used for different kinds of things, as for example: (1) in some languages shorthands such as `"a` are defined to be able to hyphenate the word if the encoding is `OT1`; (2) in some languages shorthands such as `!` are used to insert the right amount of white space; (3) several kinds of discretionaries and breaks can be inserted easily with `"-`, `"=`, etc.
The package inputenc as well as xetex an luatex have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now pdfTeX provides `\knbccode`, and luatex can manipulate the glyph list. Tools for point 3 can be still very useful in general.

[5] With it encoded string may not work as expected.

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`).

A typical error when using shorthands is the following:

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

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

`\shorthandon` {⟨*shorthands-list*⟩}
`\shorthandoff` * {⟨*shorthands-list*⟩}

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands `\shorthandoff` and `\shorthandon` are provided. They each take a list of characters as their arguments. The command `\shorthandoff` sets the `\catcode` for each of the characters in its argument to other (12); the command `\shorthandon` sets the `\catcode` to active (13). Both commands only work on 'known' shorthand characters. If a character is not known to be a shorthand character its category code will be left unchanged.
New 3.9a However, `\shorthandoff` does not behave as you would expect with characters like ~ or ^, because they usually are not "other". For them `\shorthandoff*` is provided, so that with

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

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

`\useshorthands` * {⟨*char*⟩}

The command `\useshorthands` initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.
New 3.9a User shorthands are not always alive, as they may be deactivated by languages (for example, if you use `"` for your user shorthands and switch from german to french, they stop working). Therefore, a starred version `\useshorthands*{`⟨*char*⟩`}` is provided, which makes sure shorthands are always activated.
Currently, if the package option `shorthands` is used, you must include any character to be activated with `\useshorthands`. This restriction will be lifted in a future release.

\defineshorthand [⟨*language*⟩,⟨*language*⟩,...]{⟨*shorthand*⟩}{⟨*code*⟩}

The command \defineshorthand takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to.
New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add \languageshorthands{⟨*lang*⟩} to the corresponding \extras⟨*lang*⟩, as explained below). By default, user shorthands are (re)defined.
User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over "normal" user shorthands.

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

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

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

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

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

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

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

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

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

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

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

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

\languageshorthands {⟨*language*⟩}

The command \languageshorthands can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).[6] 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 with, for example, \useshorthands.)
Very often, this is a more convenient way to deactivate shorthands than \shorthandoff, as for example if you want to define a macro to easy typing phonetic characters with tipa:

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

\babelshorthand {⟨*shorthand*⟩}

With this command you can use a shorthand even if (1) not activated in shorthands (in this case only shorthands for the current language are taken into account, ie, not user shorthands), (2) turned off with \shorthandoff or (3) deactivated with the internal \bbl@deactivate; for example, \babelshorthand{"u} or \babelshorthand{:}. (You can conveniently define your own macros, or even you own user shorthands provided they do not ovelap.)
For your records, here is a list of shorthands, but you must double check them, as they may change:[7]

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

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

Basque " ' ~

Breton : ; ? !

Catalan " ' `

Czech " -

Esperanto ^

Estonian " ~

French (all varieties) : ; ? !

Galician " . ' ~ < >

Greek ~

Hungarian `

Kurmanji ^

Latin " ^ =

Slovak " ^ ' -

Spanish " . < > '

Turkish : ! =

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

[6] 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.

[7] Thanks to Enrico Gregorio

[8] This declaration serves to nothing, but it is preserved for backward compatibility.

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 availabe in the preamble.

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

`activegrave` Same for `` ` ``.

`shorthands=` ⟨*char*⟩⟨*char*⟩... | `off`

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

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

If `'` is included, `activeacute` is set; if `` ` `` is included, `activegrave` is set. Active characters (like ~) should be preceded by `\string` (otherwise they will be expanded by 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 – of course, in such a case you cannot use shorthands in these macros, but this is not a real problem (just use “allowed” characters).

`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 any more.

`config=` ⟨*file*⟩

Load ⟨*file*⟩`.cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).

`main=` ⟨*language*⟩

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=` ⟨*language*⟩

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

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

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

`nocase` New 3.9l Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.

`silent` New 3.9l No warnings and no *infos* are written to the log file.[9]

`strings=` `generic` | `unicode` | `encoded` | ⟨*label*⟩ | ⟨*font encoding*⟩

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

`hyphenmap=` `off` | `main` | `select` | `other` | `other*`

New 3.9g Sets the behaviour of case mapping for hyphenation, provided the language defines it.[10] 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;[11]

`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.[12]

`bidi=` `default` | `basic-r`

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

[9] You can use alternatively the package silence.

[10] Turned off in plain.

[11] Duplicated options count as several ones.

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

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, which will be improved in the future. Remember basic-r is available in luatex only.

```
\documentclass{article}

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

\babelprovide[import=ar, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الاغريقي) 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}
```

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 hyphenations 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 hyphenations patterns of a single language, too.

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

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

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

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

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

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

1.13 ini files

An alternative approach to define a language is by means of an ini file. Currently babel provides about 200 of these files containing the basic data required for a language.

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

EXAMPLE Although Georgian has its own `ldf` file, here is how to declare this language with an `ini` file in Unicode engines. The `nil` language is required, because currently babel raises an error if there is no language.

```
\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

af	Afrikaans^{ul}	ce	Chechen
agq	Aghem	cgg	Chiga
ak	Akan	chr	Cherokee
am	Amharic^{ul}	ckb	Central Kurdish
ar	Arabic^{ul}	cs	Czech^{ul}
as	Assamese	cy	Welsh^{ul}
asa	Asu	da	Danish^{ul}
ast	Asturian^{ul}	dav	Taita
az-Cyrl	Azerbaijani	de-AT	German^{ul}
az-Latn	Azerbaijani	de-CH	German^{ul}
az	Azerbaijani^{ul}	de	German^{ul}
bas	Basaa	dje	Zarma
be	Belarusian^{ul}	dsb	Lower Sorbian^{ul}
bem	Bemba	dua	Duala
bez	Bena	dyo	Jola-Fonyi
bg	Bulgarian^{ul}	dz	Dzongkha
bm	Bambara	ebu	Embu
bn	Bangla^{ul}	ee	Ewe
bo	Tibetan^{u}	el	Greek^{ul}
brx	Bodo	en-AU	English^{ul}
bs-Cyrl	Bosnian	en-CA	English^{ul}
bs-Latn	Bosnian^{ul}	en-GB	English^{ul}
bs	Bosnian^{ul}	en-NZ	English^{ul}
ca	Catalan^{ul}	en-US	English^{ul}

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

pa-Arab	Punjabi
pa-Guru	Punjabi
pa	Punjabi
pl	Polish^{ul}
pms	Piedmontese^{ul}
ps	Pashto
pt-BR	Portuguese^{ul}
pt-PT	Portuguese^{ul}
pt	Portuguese^{ul}
qu	Quechua
rm	Romansh^{ul}
rn	Rundi
ro	Romanian^{ul}
rof	Rombo
ru	Russian^{ul}
rw	Kinyarwanda
rwk	Rwa
sah	Sakha
saq	Samburu
sbp	Sangu
se	Northern Sami^{ul}
seh	Sena
ses	Koyraboro Senni
sg	Sango
shi-Latn	Tachelhit
shi-Tfng	Tachelhit
shi	Tachelhit
si	Sinhala
sk	Slovak^{ul}
sl	Slovenian^{ul}
smn	Inari Sami
sn	Shona
so	Somali
sq	Albanian^{ul}
sr-Cyrl-BA	Serbian^{ul}
sr-Cyrl-ME	Serbian^{ul}
sr-Cyrl-XK	Serbian^{ul}
sr-Cyrl	Serbian^{ul}
sr-Latn-BA	Serbian^{ul}
sr-Latn-ME	Serbian^{ul}
sr-Latn-XK	Serbian^{ul}
sr-Latn	Serbian^{ul}

sr	Serbian^{ul}
sv	Swedish^{ul}
sw	Swahili
ta	Tamil^{u}
te	Telugu^{ul}
teo	Teso
th	Thai^{ul}
ti	Tigrinya
tk	Turkmen^{ul}
to	Tongan
tr	Turkish^{ul}
twq	Tasawaq
tzm	Central Atlas Tamazight
ug	Uyghur
uk	Ukrainian^{ul}
ur	Urdu^{ul}
uz-Arab	Uzbek
uz-Cyrl	Uzbek
uz-Latn	Uzbek
uz	Uzbek
vai-Latn	Vai
vai-Vaii	Vai
vai	Vai
vi	Vietnamese^{ul}
vun	Vunjo
wae	Walser
xog	Soga
yav	Yangben
yi	Yiddish
yo	Yoruba
yue	Cantonese
zgh	Standard Moroccan Tamazight
zh-Hans-HK	Chinese
zh-Hans-MO	Chinese
zh-Hans-SG	Chinese
zh-Hans	Chinese
zh-Hant-HK	Chinese
zh-Hant-MO	Chinese
zh-Hant	Chinese
zh	Chinese
zu	Zulu

In some context (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).

aghem
akan
albanian
american
amharic
arabic
armenian
assamese

asturian
asu
australian
austrian
azerbaijani-cyrillic
azerbaijani-cyrl
azerbaijani-latin
azerbaijani-latn
azerbaijani
bafia
bambara
basaa
basque
belarusian
bemba
bena
bengali
bodo
bosnian-cyrillic
bosnian-cyrl
bosnian-latin
bosnian-latn
bosnian
brazilian
breton
british
bulgarian
burmese
canadian
cantonese
catalan
centralatlastamazight
centralkurdish
chechen
cherokee
chiga
chinese-hans-hk
chinese-hans-mo
chinese-hans-sg
chinese-hans
chinese-hant-hk
chinese-hant-mo
chinese-hant
chinese-simplified-hongkongsarchina
chinese-simplified-macausarchina
chinese-simplified-singapore
chinese-simplified
chinese-traditional-hongkongsarchina
chinese-traditional-macausarchina
chinese-traditional
chinese
colognian
cornish
croatian
czech
danish
duala
dutch
dzongkha
embu
english-au
english-australia
english-ca
english-canada
english-gb
english-newzealand
english-nz
english-unitedkingdom
english-unitedstates
english-us
english
esperanto
estonian
ewe
ewondo
faroese
filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi

hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
scottishgaelic

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

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`.

`\babelfont` `[`⟨*language-list*⟩`]{`⟨*font-family*⟩`}[`⟨*font-options*⟩`]{`⟨*font-name*⟩`}`

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

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

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

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

```
\documentclass{article}

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

\babelprovide[import=he]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

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

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

EXAMPLE Here is how to do it:

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

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

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

NOTE `\fontspec` is not touched at all, only the preset font families (`rm`, `sf`, `tt`, and the like). If a language is switched when an *ad hoc* font is active, or you select the font it with this command, neither the script nor the language are 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 behaviour.

WARNING Do not use `\setxxxxfont` and `\babelfont` at the same time. `\babelfont` follows the standard LaTeX conventions to set the basic families – define `\xxdefault`, and activate it with `\xxfamily`. On the other hand, `\setxxxxfont` in fontspec takes a different approach, because `\xxfamily` is redefined with the family name hardcoded (so that `\xxdefault` becomes no-op). Of course, both methods are incompatible, and if you use `\setxxxxfont`, font switching with `\babelfont` just does *not* work (nor the standard `\xxdefault`, for that matter).

1.15 Modifying a language

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

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

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

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

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

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

- Macros to be run when a language is selected can be add to `\extras`⟨*lang*⟩:

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

  There is a counterpart for code to be run when a language is unselected: `\noextras`⟨*lang*⟩.

NOTE 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.

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.

`\babelprovide` `[`⟨*options*⟩`]{`⟨*language-name*⟩`}`

Defines the internal structure of the language with some defaults: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3, but captions and date are not defined. Conveniently, babel warns you about what to do. Very likely you will find alerts like that in the `log` file:

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

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

EXAMPLE If you need a language named `arhinish`:

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

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.
There are about 200 `ini` files, with data taken from the `ldf` files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the `ini` files. A few languages will show a warning about the current lack of suitability of the date format (hindi, french, breton, and occitan).
Besides `\today`, there is a `\<language>date` macro with three arguments: year, month and day numbers. In fact, `\today` calls `\<language>today` which in turn calls `\<language>date{\year}{\month}{\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 behaviour applies. Note in this example we set `chavacano` as first option – without it, it would select `spanish` even if `chavacano` exists.
A special value is `+`, which allocates a new language (in the TEX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with `\babelpatterns`, as for example:

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

In other engines it just supresses 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. This value is particularly important because it sets the writing direction.

`language=` ⟨*language-name*⟩

New 3.15 Sets the language name to be used by fontspec (eg, `Hindi`). Overrides the value in the `ini` file. Not so important, but sometimes still relevant.

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

1.17 Getting the current language name

`\languagename` The control sequence `\languagename` 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.

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

1.18 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 {⟨*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).
Current events are the following; in some of them you can use one to three TEX parameters (#1, #2, #3), with the meaning given:

adddialect (language name, dialect name) Used by luababel.def to load the patterns if not preloaded.
patterns (language name, language with encoding) Executed just after the \language has been set. The second argument has the patterns name actually selected (in the form of either lang:ENC or lang).
hyphenation (language name, language with encoding) Executed locally just before exceptions given in \babelhyphenation are actually set.
defaultcommands Used (locally) in \StartBabelCommands.
encodedcommands (input, font encodings) Used (locally) in \StartBabelCommands. Both xetex and luatex make sure the encoded text is read correctly.
stopcommands Used to reset the the above, if necessary.
write This event comes just after the switching commands are written to the aux file.
beforeextras Just before executing \extras⟨*language*⟩. This event and the next one should not contain language-dependent code (for that, add it to \extras⟨*language*⟩).
afterextras Just after executing \extras⟨*language*⟩. For example, the following deactivates shorthands in all languages:

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

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

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

initiateactive (char as active, char as other, original char) New 3.9i Executed just after a shorthand has been ‘initiated’. The three parameters are the same character with different catcodes: active, other (\string’ed) and the original one.
afterreset New 3.9i Executed when selecting a language just after \originalTeX is run and reset to its base value, before executing \captions⟨*language*⟩ and \date⟨*language*⟩.

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

everylanguage (language) Executed before every language patterns are loaded.
loadkernel (file) By default loads switch.def. It can be used to load a different version of this files or to load nothing.
loadpatterns (patterns file) Loads the patterns file. Used by luababel.def.
loadexceptions (exceptions file) Loads the exceptions file. Used by luababel.def.

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

1.19 Hyphenation tools

\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 oportunity or, in TEX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking oportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking oportunity.

In TEX, - and \- forbid further breaking oportunities in the word. This is the desired behaviour 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, Portugese, Catalan or Danish, "- is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian, it is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking oportunities in the rest of the word.

Therefore, some macros are provide 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, Portugese and Spanish.
- \babelhyphen{nobreak} inserts a hard hyphen without a break after it (even if a space follows).
- \babelhyphen{empty} inserts a break oportunity 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 enabling it, there is a starred counterpart: \babelhyphen*{soft} (which in most cases is equivalent to the original \-), \babelhyphen*{hard}, etc.

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

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

\babelhyphenation [⟨*language*⟩,⟨*language*⟩,...]{⟨*exceptions*⟩}

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones.

It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as the language specific encoding (not set in the preamble by default). Multiple \babelhyphenation's are allowed. For example:

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

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

`\babelpatterns` `[`⟨*language*⟩`,`⟨*language*⟩`,`...`]{`⟨*patterns*⟩`}`

New 3.9m *In luatex only*,[13] 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.

1.20 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low level) or a language name (high level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.[14]

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.[15]

`\ensureascii` `{`⟨*text*⟩`}`

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

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

The foregoing rules (which are applied "at begin document") cover most of cases. No asumption 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.

[13] 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.

[14] 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. As to directionality, it poses special challenges because it also affects individual characters and layout elements.

[15] But still defined for backwards compatibility.

1.21 Selecting directions

No macros to select the writing direction are provided, either – writing direction is intrinsic to each script and therefore it is best set by the language (which could be a dummy one). Furthermore, there are in fact two right-to-left modes, depending on the language, which differ in the way 'weak' numeric characters are ordered (eg, Arabic %123 *vs* Hebrew 123%).
However, digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic-r` and, usually, xetex). For it (although available in all engines), the following command is provided:

`\babelsublr` {⟨*lr-text*⟩}

Set {⟨*lr-text*⟩} in L mode. It's mainly intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is not a `rl` counterpart.

WARNING Setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). This means the babel bidi code may take some time before it is truly stable. 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).

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

Working – 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, while the section text is still the current language. The latter is passed to tocs and marks, too. It can be set individually or to all standard sectioning commands. TODO: In the latter case, `\markboth` and `markright` are not redefined, because the text passed includes already the language selector. TODO: layout=nomarks ?

1.22 Language attributes

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

1.23 Languages supported by babel

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

Afrikaans afrikaans
Azerbaijani azerbaijani

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

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchle, ethiop and friulan.
Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK). For example, if you have got the velthuis/devnag package, you can create a file with extension `.dn`:

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

Then you preprocess it with `devnag` ⟨*file*⟩, which creates ⟨*file*⟩`.tex`; you can then typeset the latter with LaTeX.

1.24 Tips, workarounds, know 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 hhline 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 hhline (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 TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.[16] So, if you write a chunk of French text with `\foreinglanguage`, the apostrophes might not be taken into account. This is a limitation of TeX, not of babel. Alternatively, you may use `\useshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code `"8000`) as a shorthand can make TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.
iflang Tests correctly the current language.

[16] This explains why LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savinghyphcodes` is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.

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.

1.25 Current and future work

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).

It is possible now to typeset Arabic or Hebrew with numbers and L text. Next on the roadmap are line breaking in Thai and the like, as well as "non-European" digits. Also on the roadmap are R layouts (lists, footnotes, tables, column order), page and section numbering, and maybe kashida justification.

As to Thai line breaking, here is the basic idea of what luatex can do for us, with the Thai patterns and a little script (the final version will not be so little, of course). It replaces each discretionary by the equivalent to ZWJ.

```
\documentclass{article}

\usepackage[nil]{babel}

\babelprovide[import=th, main]{thai}

\babelfont{rm}{FreeSerif}

\directlua{
local GLYF = node.id'glyph'
function insertsp (head)
  local size = 0
  for item in node.traverse(head) do
    local i = item.id
    if i == GLYF then
      f = font.getfont(item.font)
      size = f.size
    elseif i == 7 then
      local n = node.new(12, 0)
      node.setglue(n, 0, size * 1) % 1 is a factor
      node.insert_before(head, item, n)
      node.remove(head, item)
    end
  end
end

luatexbase.add_to_callback('hyphenate',
  function (head, tail)
    lang.hyphenate(head)
    insertsp(head)
  end, 'insertsp')
}

\begin{document}

(Thai text.)

\end{document}
```

Useful additions would be, for example, time, currency, addresses and personal names.[17]. But that is the easy part, because they don't require modifying the LaTeX internals.
Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ból", in Spanish an item labelled "3.^{o}" may be referred to as either "ítem 3.^{o}" or "3.^{er} ítem", and so on.

1.26 Tentative and experimental code

Handling of **"Unicode" fonts** is problematic. There is fontspec, but special macros are required (not only the NFSS ones) and it doesn't provide "orthogonal axis" for features, including those related to the language (mainly language and script). A couple of tentative macros, were provided by babel ($\geq$3.9g) with a partial solution. These macros are now deprecated – use `\babelfont`.

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

So, for example:

```
\setmainfont[Language=Turkish]{Minion Pro}
\babelFSstore{turkish}
\setmainfont{Minion Pro}
\babelFSfeatures{turkish}{Language=Turkish}
```

Bidi writing is taking its *first steps*. *First steps* means exactly that. For example, in luatex any Arabic text must be marked up explicitly in L mode. On the other hand, xetex poses quite different challenges. Document layout (lists, footnotes, etc.) is not touched at all.
See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). xetex relies on the font to properly handle these unmarked changes, so it is not under the control of TeX.

2 Loading languages with `language.dat`

TeX and most engines based on it (pdfTeX, xetex, ϵ-TeX, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, LaTeX, XeLaTeX, pdfLaTeX). babel provides a tool which has become standand in many distributions and

[17] See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those system, however, have limited application to TeX because their aim is just to display information and not fine typesetting.

based on a “configuration file” named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

New 3.9q With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).[18] Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).[19]

2.1 Format

In that file the person who maintains a TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[20]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.[21] For example:

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

With the previous settings, if the enconding when the language is selected is `T1` then the patterns in hyphen`T1.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.

[18] This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

[19] 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`.

[20] This is because different operating systems sometimes use *very* different file-naming conventions.

[21] This in not a new feature, but in former versions it didn’t work correctly.

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

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

- Some of the language-specific definitions might be used by plain 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 `\`⟨*lang*⟩`hyphenmins`, `\captions`⟨*lang*⟩, `\date`⟨*lang*⟩, `\extras`⟨*lang*⟩ and `\noextras`⟨*lang*⟩(the last two may be left empty); where ⟨*lang*⟩ is either the name of the language definition file or the name of the LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date`⟨*lang*⟩ but not `\captions`⟨*lang*⟩ does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\l@`⟨*lang*⟩ to be a dialect of `\language0` when `\l@`⟨*lang*⟩ is undefined.

- Language names must be all lowercase. If an unknow language is selected, babel will attempt setting it after lowercasing its name.

- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in LaTeX (quotes are entered as ``` `` ``` and `''`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).

- Captions should not contain shorthands or encoding dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.

- Avoid adding things to `\noextras`⟨*lang*⟩ except for umlauthigh and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras`⟨*lang*⟩.

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low level) or the language (high level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.[22]

- Please, for "private" internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

3.1 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 behaviour 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 (currenty, default pattern files do *not* set them).

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

`\date`⟨*lang*⟩ The macro `\date`⟨*lang*⟩ defines `\today`.

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

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

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

`\main@language` To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of

[22] But not removed, for backward compatibility.

\selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.

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

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

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

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

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

\substitutefontfamily (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.2 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 in sec. 3.7 (babel 3.9 and later).

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

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

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

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

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

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

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

\StartBabelCommands*{<dialect>}{captions}
\SetString\chaptername{<chapter name>}
```

```
% More strings

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

\EndBabelCommands

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

\ldf@finish{<language>}
```

3.3 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 LaTeX to give a character the category code 'active'. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

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

The command `\bbl@activate` is used to change the way an active character expands. `\bbl@activate` 'switches on' the active behaviour 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 TeXbook states: "Plain TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code." [2, p. 380] It is used to set text 'verbatim'. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. LaTeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special`⟨*char*⟩ and `\bbl@remove@special`⟨*char*⟩ add and remove the character ⟨*char*⟩ to these two sets.

3.4 Support for saving macro definitions

Language definition files may want to *re*define 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[23].

`\babel@save`

To save the current meaning of any control sequence, the macro `\babel@save` is provided. It takes one argument, ⟨*csname*⟩, 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.

[23] This mechanism was introduced by Bernd Raichle.

3.5 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 behaviour is preserved for backward compatibility. If you are using etoolbox, by Philipp Lehman, consider using the tools provided by this package instead of \addto.

3.6 Macros common to a number of languages

\bbl@allowhyphens In several languages compound words are used. This means that when TEX has to hyphenate such a compound word, it only does so at the '-' that is used in such words. To allow hyphenation in the rest of such a compound word, the macro \bbl@allowhyphens can be used.

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

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

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

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

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

3.7 Encoding-dependent strings

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

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

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

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

\StartBabelCommands {⟨*language-list*⟩}{⟨*category*⟩}[⟨*selector*⟩]

The ⟨*language-list*⟩ 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 take precedence (ie, it works much like `\providecommand`).
Encoding info is `charset=` followed by a charset, which if given sets how the strings should be traslated to the internal representation used by the engine, typically `utf8`, which is the only value supported currently (default is no traslations). 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 `\SetCase`s are still honoured (in a `encoded` way).
The ⟨*category*⟩ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.[24] It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
  \SetString\monthiiname{Februar}
```

[24] In future releases further categories may be added.

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

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

\EndBabelCommands
```

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

`\StartBabelCommands` `*{`⟨*language-list*⟩`}{`⟨*category*⟩`}[`⟨*selector*⟩`]`

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 appropiate.[25]

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

`\AfterBabelCommands` `{`⟨*code*⟩`}`

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

`\SetString` `{`⟨*macro-name*⟩`}{`⟨*string*⟩`}`

Adds ⟨*macro-name*⟩ to the current category, and defines globally ⟨*lang-macro-name*⟩ to ⟨*code*⟩ (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` `{`⟨*macro-name*⟩`}{`⟨*string-list*⟩`}`

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

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

[25] This replaces in 3.9g a short-lived `\UseStrings` which has been removed because it did not work.

#1 is replaced by the roman numeral.

\SetCase [⟨*map-list*⟩]{⟨*toupper-code*⟩}{⟨*tolower-code*⟩}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap {⟨*to-lower-macros*⟩}

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

Most of changes in version 3.9 are 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 behaviour for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

- `\select@language` did not set `\languagename`. 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 where not reset at the end of language options, and that lead to incompatibilities between languages.
- `\textormath` raised and 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.

4.2 Changes in babel version 3.7

In babel version 3.7 a number of bugs that were found in version 3.6 are fixed. Also a number of changes and additions have occurred:

- Shorthands are expandable again. The disadvantage is that one has to type `'{}a` when the acute accent is used as a shorthand character. The advantage is that a number of other problems (such as the breaking of ligatures, etc.) have vanished.
- Two new commands, `\shorthandon` and `\shorthandoff` have been introduced to enable to temporarily switch off one or more shorthands.
- Support for typesetting Hebrew (and potential support for typesetting other right-to-left written languages) is now available thanks to Rama Porrat and Boris Lavva.
- A language attribute has been added to the `\mark...` commands in order to make sure that a Greek header line comes out right on the last page before a language switch.

- Hyphenation pattern files are now read *inside a group*; therefore any changes a pattern file needs to make to lowercase codes, uppercase codes, and category codes are kept local to that group. If they are needed for the language, these changes will need to be repeated and stored in `\extras...`
- The concept of language attributes is introduced. It is intended to give the user some control over the features a language-definition file provides. Its first use is for the Greek language, where the user can choose the πολυτονικό ("polytonikó" or multi-accented) Greek way of typesetting texts.
- The environment `hyphenrules` is introduced.
- The syntax of the file `language.dat` has been extended to allow (optionally) specifying the font encoding to be used while processing the patterns file.
- The command `\providehyphenmins` should now be used in language definition files in order to be able to keep any settings provided by the pattern file.

Part II
The code

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

5 Identification and loading of required files

Code documentation is still under revision.
The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.
babel.def defines the rest of macros. It has tow parts: a generic one and a second one only for LaTeX.
babel.sty is the LaTeX package, which set options and load language styles.
plain.def defines some LaTeX macros required by `babel.def` and provides a few tools for Plain.
hyphen.cfg is the file to be used when generating the formats to load hyphenation patterns. By default it also loads `switch.def`.

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

```
⟨⟨version=3.15⟩⟩
⟨⟨date=2017/11/03⟩⟩
```

6 Tools

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 behaviour of the latter. Used in babel.def and in babel.sty, which means in LaTeX is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```
⟨⟨*Basic macros⟩⟩ ≡
\def\bbl@stripslash{\expandafter\@gobble\string}
\def\bbl@add#1#2{%
  \bbl@ifunset{\bbl@stripslash#1}%
    {\def#1{#2}}%
    {\expandafter\def\expandafter#1\expandafter{#1#2}}}
\def\bbl@xin@{\@expandtwoargs\in@}
\def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
\def\bbl@cs#1{\csname bbl@#1\endcsname}
\def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
\def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
\def\bbl@@loop#1#2#3,{%
  \ifx\@nnil#3\relax\else
    \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
  \fi}
\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.

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

\bbl@afterelse
\bbl@afterfi

Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the \else and \fi parts of an \if-statement[26]. These macros will break if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

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

\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.

```
\def\bbl@tempa#1{%
  \long\def\bbl@trim##1##2{%
    \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
  \def\bbl@trim@c{%
    \ifx\bbl@trim@a\@sptoken
      \expandafter\bbl@trim@b
    \else
      \expandafter\bbl@trim@b\expandafter#1%
    \fi}%
  \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
```

[26] This code is based on code presented in TUGboat vol. 12, no2, June 1991 in "An expansion Power Lemma" by Sonja Maus.

```
\bbl@tempa{ }
\long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
\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 ϵ-tex engine, it is based on \ifcsname, which is more efficient, and do not waste memory.

```
\def\bbl@ifunset#1{%
  \expandafter\ifx\csname#1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\bbl@ifunset{ifcsname}%
  {}%
  {\def\bbl@ifunset#1{%
     \ifcsname#1\endcsname
       \expandafter\ifx\csname#1\endcsname\relax
         \bbl@afterelse\expandafter\@firstoftwo
       \else
         \bbl@afterfi\expandafter\@secondoftwo
       \fi
     \else
       \expandafter\@firstoftwo
     \fi}}
```

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

```
\def\bbl@ifblank#1{%
  \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
\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).

```
\def\bbl@forkv#1#2{%
  \def\bbl@kvcmd##1##2##3{#2}%
  \bbl@kvnext#1,\@nil,}
\def\bbl@kvnext#1,{%
  \ifx\@nil#1\relax\else
    \bbl@ifblank{#1}{}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
    \expandafter\bbl@kvnext
  \fi}
\def\bbl@forkv@eq#1=#2=#3\@nil#4{%
  \bbl@trim@def\bbl@forkv@a{#1}%
  \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).

```
\def\bbl@vforeach#1#2{%
  \def\bbl@forcmd##1{#2}%
  \bbl@fornext#1,\@nil,}
\def\bbl@fornext#1,{%
  \ifx\@nil#1\relax\else
    \bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}%
    \expandafter\bbl@fornext
  \fi}
\def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

\bbl@replace

```
\def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
  \toks@{}%
  \def\bbl@replace@aux##1#2##2#2{%
    \ifx\bbl@nil##2%
      \toks@\expandafter{\the\toks@##1}%
    \else
      \toks@\expandafter{\the\toks@##1#3}%
      \bbl@afterfi
      \bbl@replace@aux##2#2%
    \fi}%
  \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
  \edef#1{\the\toks@}}
```

\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.

```
\def\bbl@exp#1{%
  \begingroup
    \let\\\noexpand
    \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
    \edef\bbl@exp@aux{\endgroup#1}%
  \bbl@exp@aux}
```

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.

```
\def\bbl@ifsamestring#1#2{%
  \begingroup
    \protected@edef\bbl@tempb{#1}%
    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
    \protected@edef\bbl@tempc{#2}%
    \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
    \ifx\bbl@tempb\bbl@tempc
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
\chardef\bbl@engine=%
  \ifx\directlua\@undefined
    \ifx\XeTeXinputencoding\@undefined
      \z@
    \else
      \tw@
    \fi
  \else
    \@ne
  \fi
⟨⟨/Basic macros⟩⟩
```

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

```
⟨⟨∗Make sure ProvidesFile is defined⟩⟩ ≡
\ifx\ProvidesFile\@undefined
```

```
  \def\ProvidesFile#1[#2 #3 #4]{%
    \wlog{File: #1 #4 #3 <#2>}%
    \let\ProvidesFile\@undefined}
\fi
⟨⟨/Make sure ProvidesFile is defined⟩⟩
```

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

```
⟨⟨*Load patterns in luatex⟩⟩ ≡
\ifx\directlua\@undefined\else
  \ifx\bbl@luapatterns\@undefined
    \input luababel.def
  \fi
\fi
⟨⟨/Load patterns in luatex⟩⟩
```

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

```
⟨⟨*Load macros for plain if not LaTeX⟩⟩ ≡
\ifx\AtBeginDocument\@undefined
  \input plain.def\relax
\fi
⟨⟨/Load macros for plain if not LaTeX⟩⟩
```

6.1 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 requires loading switch.def in the format.

```
⟨⟨*Define core switching macros⟩⟩ ≡
\ifx\language\@undefined
  \csname newcount\endcsname\language
\fi
⟨⟨/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.

```
⟨⟨*Define core switching macros⟩⟩ ≡
\ifx\newlanguage\@undefined
  \csname newcount\endcsname\last@language
  \def\addlanguage#1{%
    \global\advance\last@language\@ne
    \ifnum\last@language<\@cclvi
    \else
      \errmessage{No room for a new \string\language!}%
    \fi
    \global\chardef#1\last@language
    \wlog{\string#1 = \string\language\the\last@language}}
\else
  \countdef\last@language=19
  \def\addlanguage{\alloc@9\language\chardef\@cclvi}
\fi
⟨⟨/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 LaTeX2.09. 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 The Package File (LaTeX, `babel.sty`)

In order to make use of the features of LaTeX 2$_\varepsilon$, the babel system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options whose name is different from that of the `.ldf` file (like variant spellings). It also takes care of a number of compatibility issues with other packages an defines a few aditional package options.
Apart from all the language options below we also have a few options that influence the behaviour 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.

7.1 `base`

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

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ The Babel package]
\@ifpackagewith{babel}{debug}
  {\let\bbl@debug\@firstofone}
  {\let\bbl@debug\@gobble}
\input switch.def\relax
⟨⟨Load patterns in luatex⟩⟩
⟨⟨Basic macros⟩⟩
\def\AfterBabelLanguage#1{%
  \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.

```
\ifx\bbl@languages\@undefined\else
  \begingroup
    \catcode`\^^I=12
    \@ifpackagewith{babel}{showlanguages}{%
      \begingroup
```

```
        \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
        \wlog{<*languages>}%
        \bbl@languages
        \wlog{</languages>}%
      \endgroup}{}
  \endgroup
  \def\bbl@elt#1#2#3#4{%
    \ifnum#2=\z@
      \gdef\bbl@nulllanguage{#1}%
      \def\bbl@elt##1##2##3##4{}%
    \fi}%
  \bbl@languages
\fi
\@ifpackagewith{babel}{bidi=basic-r}{% must go before any \DeclareOption
  \let\bbl@beforeforeign\leavevmode
  \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
  \RequirePackage{luatexbase}%
  \directlua{
    require('babel-bidi.lua')
    require('babel-bidi-basic-r.lua')
    luatexbase.add_to_callback('pre_linebreak_filter',
      Babel.pre_otfload,
      'Babel.pre_otfload',
      luatexbase.priority_in_callback('pre_linebreak_filter',
        'luaotfload.node_processor') or nil)
    luatexbase.add_to_callback('hpack_filter',
      Babel.pre_otfload,
      'Babel.pre_otfload',
      luatexbase.priority_in_callback('hpack_filter',
        'luaotfload.node_processor') or nil)}}{}
```

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

```
\@ifpackagewith{babel}{base}{%
  \ifx\directlua\@undefined
    \DeclareOption*{\bbl@patterns{\CurrentOption}}%
  \else
    \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
  \fi
  \DeclareOption{base}{}%
  \DeclareOption{showlanguages}{}%
  \ProcessOptions
  \global\expandafter\let\csname opt@babel.sty\endcsname\relax
  \global\expandafter\let\csname ver@babel.sty\endcsname\relax
  \global\let\@ifl@ter@@\@ifl@ter
  \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
  \endinput}{}%
```

7.2 key=value options and other general option

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

```
\bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
\def\bbl@tempb#1.#2{%
   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
```

```
\def\bbl@tempd#1.#2\@nnil{%
  \ifx\@empty#2%
    \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
  \else
    \in@{=}{#1}\ifin@
      \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
    \else
      \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
      \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
    \fi
  \fi}
\let\bbl@tempc\@empty
\bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
\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.

```
\DeclareOption{KeepShorthandsActive}{}
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
\DeclareOption{debug}{}
\DeclareOption{noconfigs}{}
\DeclareOption{showlanguages}{}
\DeclareOption{silent}{}
\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
⟨⟨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.

```
\let\bbl@opt@shorthands\@nnil
\let\bbl@opt@config\@nnil
\let\bbl@opt@main\@nnil
\let\bbl@opt@headfoot\@nnil
\let\bbl@opt@layout\@nnil
```

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

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

```
\let\bbl@language@opts\@empty
\DeclareOption*{%
```

```
  \bbl@xin@{\string=}{\CurrentOption}%
  \ifin@
    \expandafter\bbl@tempa\CurrentOption\bbl@tempa
  \else
    \bbl@add@list\bbl@language@opts{\CurrentOption}%
  \fi}
```

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

```
\ProcessOptions*
```

7.3 Conditional loading of shorthands

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

```
\def\bbl@sh@string#1{%
  \ifx#1\@empty\else
    \ifx#1t\string~%
    \else\ifx#1c\string,%
    \else\string#1%
    \fi\fi
    \expandafter\bbl@sh@string
  \fi}
\ifx\bbl@opt@shorthands\@nnil
  \def\bbl@ifshorthand#1#2#3{#2}%
\else\ifx\bbl@opt@shorthands\@empty
  \def\bbl@ifshorthand#1#2#3{#3}%
\else
```

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

```
  \def\bbl@ifshorthand#1{%
    \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
    \ifin@
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}
```

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

```
  \edef\bbl@opt@shorthands{%
    \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

The following is ignored with shorthands=off, since it is intended to take some aditional actions for certain chars.

```
  \bbl@ifshorthand{'}%
    {\PassOptionsToPackage{activeacute}{babel}}{}
  \bbl@ifshorthand{`}%
    {\PassOptionsToPackage{activegrave}{babel}}{}
\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.

```
\ifx\bbl@opt@headfoot\@nnil\else
```

```
  \g@addto@macro\@resetactivechars{%
    \set@typeset@protect
    \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
    \let\protect\noexpand}
\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.

```
\ifx\bbl@opt@safe\@undefined
  \def\bbl@opt@safe{BR}
\fi
\ifx\bbl@opt@main\@nnil\else
  \edef\bbl@language@opts{%
    \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
      \bbl@opt@main}
\fi
```

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

```
\ifx\bbl@opt@layout\@nnil
  \newcommand\IfBabelLayout[3]{#3}%
\else
  \newcommand\IfBabelLayout[1]{%
    \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
    \ifin@
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}
\fi
```

7.4 Language options

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

```
\let\bbl@afterlang\relax
\let\BabelModifiers\relax
\let\bbl@loaded\@empty
\def\bbl@load@language#1{%
  \InputIfFileExists{#1.ldf}%
    {\edef\bbl@loaded{\CurrentOption
       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
     \expandafter\let\expandafter\bbl@afterlang
        \csname\CurrentOption.ldf-h@@k\endcsname
     \expandafter\let\expandafter\BabelModifiers
        \csname bbl@mod@\CurrentOption\endcsname}%
    {\bbl@error{%
       Unknown option `\CurrentOption'. Either you misspelled it\\%
       or the language definition file \CurrentOption.ldf was not found}{%
       Valid options are: shorthands=, KeepShorthandsActive,\\%
       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
       headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

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

```
\def\bbl@try@load@lang#1#2#3{%
    \IfFileExists{\CurrentOption.ldf}%
      {\bbl@load@language{\CurrentOption}}%
      {#1\bbl@load@language{#2}#3}}
\DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
\DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
\DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
\DeclareOption{hebrew}{%
  \input{rlbabel.def}%
  \bbl@load@language{hebrew}}
\DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
\DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
\DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
\DeclareOption{polutonikogreek}{%
  \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
\DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
\DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
\DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
\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.

```
\ifx\bbl@opt@config\@nnil
  \@ifpackagewith{babel}{noconfigs}{}%
    {\InputIfFileExists{bblopts.cfg}%
      {\typeout{*************************************^^J%
               * Local config file bblopts.cfg used^^J%
               *}}%
      {}}%
\else
  \InputIfFileExists{\bbl@opt@config.cfg}%
    {\typeout{*************************************^^J%
             * Local config file \bbl@opt@config.cfg used^^J%
             *}}%
    {\bbl@error{%
       Local config file `\bbl@opt@config.cfg' not found}{%
       Perhaps you misspelled it.}}%
\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 name of the option and the file are the same.

```
\bbl@for\bbl@tempa\bbl@language@opts{%
  \bbl@ifunset{ds@\bbl@tempa}%
    {\edef\bbl@tempb{%
       \noexpand\DeclareOption
         {\bbl@tempa}%
         {\noexpand\bbl@load@language{\bbl@tempa}}}%
     \bbl@tempb}%
     \@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 accesing the file system just to see if the option could be a language.

```
\bbl@foreach\@classoptionslist{%
  \bbl@ifunset{ds@#1}%
```

```
    {\IfFileExists{#1.ldf}%
      {\DeclareOption{#1}{\bbl@load@language{#1}}}%
      {}}%
    {}}
```

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

```
\ifx\bbl@opt@main\@nnil\else
  \expandafter
  \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
  \DeclareOption{\bbl@opt@main}{}
\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):

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

```
\ifx\bbl@opt@main\@nnil
  \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
  \let\bbl@tempc\@empty
  \bbl@for\bbl@tempb\bbl@tempa{%
    \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
    \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
  \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
  \expandafter\bbl@tempa\bbl@loaded,\@nnil
  \ifx\bbl@tempb\bbl@tempc\else
    \bbl@warning{%
      Last declared language option is `\bbl@tempc',\\%
      but the last processed one was `\bbl@tempb'.\\%
      The main language cannot be set as both a global\\%
      and a package option. Use `main=\bbl@tempc' as\\%
      option. Reported}%
  \fi
\else
  \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
  \ExecuteOptions{\bbl@opt@main}
  \DeclareOption*{}
  \ProcessOptions*
\fi
\def\AfterBabelLanguage{%
  \bbl@error
    {Too late for \string\AfterBabelLanguage}%
    {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.

```
\ifx\bbl@main@language\@undefined
  \bbl@error{%
```

```
    You haven't specified a language option}{%
    You need to specify a language, either as a global option\\%
    or as an optional argument to the \string\usepackage\space
    command;\\%
    You shouldn't try to proceed from here, type x to quit.}
\fi
⟨/package⟩
```

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

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `babel.def` contains most of the code, while `switch.def` defines the language switching commands; both can be read at run time. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns (by default, it also inputs `switch.def`, for "historical reasons", but it is not necessary). When `babel.def` is loaded it checks if the current version of `switch.def` is in the format; if not it is loaded. A further file, `babel.sty`, contains LaTeX-specific stuff. Because plain TeX users might want to use some of the features of the babel system too, care has to be taken that plain TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain TeX and LaTeX, some of it is for the LaTeX case only.
Plain formats based on etex (etex, xetex, luatex) don't load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

8.1 Tools

```
⟨*core⟩
\ifx\ldf@quit\@undefined
\else
  \expandafter\endinput
\fi
⟨⟨Make sure ProvidesFile is defined⟩⟩
\ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
⟨⟨Load macros for plain if not LaTeX⟩⟩
\ifx\bbl@ifshorthand\@undefined
  \def\bbl@ifshorthand#1#2#3{#2}%
  \def\bbl@opt@safe{BR}
  \def\AfterBabelLanguage#1#2{}
  \let\bbl@afterlang\relax
  \let\bbl@language@opts\@empty
\fi
\input switch.def\relax
\ifx\bbl@languages\@undefined
  \ifx\directlua\@undefined
    \openin1 = language.def
    \ifeof1
      \closein1
      \message{I couldn't find the file language.def}
    \else
      \closein1
      \begingroup
        \def\addlanguage#1#2#3#4#5{%
          \expandafter\ifx\csname lang@#1\endcsname\relax\else
            \global\expandafter\let\csname l@#1\expandafter\endcsname
              \csname lang@#1\endcsname
          \fi}%
        \def\uselanguage#1{}%
        \input language.def
      \endgroup
    \fi
  \fi
  \chardef\l@english\z@
  \fi
  ⟨⟨Load patterns in luatex⟩⟩
  ⟨⟨Basic macros⟩⟩
```

`\addto` For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro `\addto` is introduced. It takes two arguments, a ⟨*control sequence*⟩ and 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. Otherwise the replacement text for the ⟨*control sequence*⟩ is expanded and stored in a token register, together with the TEX-code to be added. Finally the ⟨*control sequence*⟩ is *re*defined, using the contents of the token register.

```
\def\addto#1#2{%
  \ifx#1\@undefined
    \def#1{#2}%
  \else
    \ifx#1\relax
      \def#1{#2}%
    \else
      {\toks@\expandafter{#1#2}%
       \xdef#1{\the\toks@}}%
    \fi
  \fi}
```

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

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

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

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

```
\def\bbl@redefine#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1%
  \expandafter\def\csname\bbl@tempa\endcsname}
```

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

```
\@onlypreamble\bbl@redefine
```

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

```
\def\bbl@redefine@long#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1%
  \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
\@onlypreamble\bbl@redefine@long
```

\bbl@redefinerobust 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␣.

```
\def\bbl@redefinerobust#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \bbl@ifunset{\bbl@tempa\space}%
    {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
     \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
    {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
    \@namedef{\bbl@tempa\space}}
```

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

```
\@onlypreamble\bbl@redefinerobust
```

8.2 Hooks

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

```
\def\AddBabelHook#1#2{%
  \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}{}%
  \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
  \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
  \bbl@ifunset{bbl@ev@#1@#2}%
    {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}%
     \bbl@csarg\newcommand}%
    {\bbl@csarg\let{ev@#1@#2}\relax
     \bbl@csarg\newcommand}%
  {ev@#1@#2}[\bbl@tempb]}
\def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
\def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
\def\bbl@usehooks#1#2{%
  \def\bbl@elt##1{%
    \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}%
  \@nameuse{bbl@ev@#1}}
```

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).

```
\def\bbl@evargs{,% don't delete the comma
  everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
  adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
  beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
  hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0}
```

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

The macro \bbl@e@⟨*language*⟩ contains \bbl@ensure{⟨*include*⟩}{⟨*exclude*⟩}{⟨*fontenc*⟩}, which in 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.

```
\newcommand\babelensure[2][]{%  TODO - revise test files
  \AddBabelHook{babel-ensure}{afterextras}{%
    \ifcase\bbl@select@type
      \@nameuse{bbl@e@\languagename}%
    \fi}%
  \begingroup
    \let\bbl@ens@include\@empty
    \let\bbl@ens@exclude\@empty
    \def\bbl@ens@fontenc{\relax}%
    \def\bbl@tempb##1{%
      \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
    \edef\bbl@tempa{\bbl@tempb#1\@empty}%
    \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
    \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
    \def\bbl@tempc{\bbl@ensure}%
    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
      \expandafter{\bbl@ens@include}}%
    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
      \expandafter{\bbl@ens@exclude}}%
    \toks@\expandafter{\bbl@tempc}%
    \bbl@exp{%
  \endgroup
  \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
\def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
  \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
    \ifx##1\@empty\else
      \in@{##1}{#2}%
      \ifin@\else
        \bbl@ifunset{bbl@ensure@\languagename}%
          {\bbl@exp{%
            \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
              \\\foreignlanguage{\languagename}%
              {\ifx\relax#3\else
                \\\fontencoding{#3}\\\selectfont
               \fi
               ########1}}}}%
          {}%
        \toks@\expandafter{##1}%
        \edef##1{%
           \bbl@csarg\noexpand{ensure@\languagename}%
           {\the\toks@}}%
      \fi
      \expandafter\bbl@tempb
    \fi}%
  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
  \def\bbl@tempa##1{% elt for include list
    \ifx##1\@empty\else
      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
      \ifin@\else
        \bbl@tempb##1\@empty
      \fi
      \expandafter\bbl@tempa
    \fi}%
  \bbl@tempa#1\@empty}
\def\bbl@captionslist{%
  \prefacename\refname\abstractname\bibname\chaptername\appendixname
  \contentsname\listfigurename\listtablename\indexname\figurename
  \tablename\partname\enclname\ccname\headtoname\pagename\seename
  \alsoname\proofname\glossaryname}
```

8.3 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```
\def\bbl@ldfinit{%
  \let\bbl@screset\@empty
  \let\BabelStrings\bbl@opt@string
  \let\BabelOptions\@empty
  \let\BabelLanguages\relax
  \ifx\originalTeX\@undefined
    \let\originalTeX\@empty
  \else
    \originalTeX
  \fi}
\def\LdfInit#1#2{%
  \chardef\atcatcode=\catcode`\@
  \catcode`\@=11\relax
  \chardef\eqcatcode=\catcode`\=
  \catcode`\==12\relax
  \expandafter\if\expandafter\@backslashchar
                 \expandafter\@car\string#2\@nil
    \ifx#2\@undefined\else
      \ldf@quit{#1}%
    \fi
  \else
    \expandafter\ifx\csname#2\endcsname\relax\else
      \ldf@quit{#1}%
    \fi
  \fi
  \bbl@ldfinit}
```

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

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

```
\def\bbl@afterldf#1{%
  \bbl@afterlang
  \let\bbl@afterlang\relax
  \let\BabelModifiers\relax
  \let\bbl@screset\relax}%
\def\ldf@finish#1{%
  \loadlocalcfg{#1}%
  \bbl@afterldf{#1}%
  \expandafter\main@language\expandafter{#1}%
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \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.

```
\@onlypreamble\LdfInit
\@onlypreamble\ldf@quit
\@onlypreamble\ldf@finish
```

`\main@language`
`\bbl@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.

```
\def\main@language#1{%
  \def\bbl@main@language{#1}%
  \let\languagename\bbl@main@language
  \bbl@patterns{\languagename}}
```

We also have to make sure that some code gets executed at the beginning of the document.

```
\AtBeginDocument{%
  \expandafter\selectlanguage\expandafter{\bbl@main@language}%
  \ifcase\bbl@engine\or\pagedir\bodydir\fi}  % TODO - a better place
```

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

```
\def\select@language@x#1{%
  \ifcase\bbl@select@type
    \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
  \else
    \select@language{#1}%
  \fi}
```

8.4 Shorthands

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

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

```
\def\bbl@add@special#1{% 1:a macro like \", \?, etc.
  \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
  \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
  \ifx\nfss@catcodes\@undefined\else % TODO - same for above
    \begingroup
      \catcode`#1\active
      \nfss@catcodes
      \ifnum\catcode`#1=\active
        \endgroup
        \bbl@add\nfss@catcodes{\@makeother#1}%
      \else
        \endgroup
      \fi
  \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.

```
\def\bbl@remove@special#1{%
  \begingroup
    \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
                 \else\noexpand##1\noexpand##2\fi}%
    \def\do{\x\do}%
    \def\@makeother{\x\@makeother}%
  \edef\x{\endgroup
    \def\noexpand\dospecials{\dospecials}%
    \expandafter\ifx\csname @sanitize\endcsname\relax\else
      \def\noexpand\@sanitize{\@sanitize}%
    \fi}%
  \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`).

```
\def\bbl@active@def#1#2#3#4{%
  \@namedef{#3#1}{%
```

```
    \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
      \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
    \else
      \bbl@afterfi\csname#2@sh@#1@\endcsname
    \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.

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

```
\def\initiate@active@char#1{%
  \bbl@ifunset{active@char\string#1}%
    {\bbl@withactive
      {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
    {}}
```

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 treatement to avoid making them `\relax`).

```
\def\@initiate@active@char#1#2#3{%
  \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
  \ifx#1\@undefined
    \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
  \else
    \bbl@csarg\let{oridef@@#2}#1%
    \bbl@csarg\edef{oridef@#2}{%
      \let\noexpand#1%
      \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
  \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*).

```
  \ifx#1#3\relax
    \expandafter\let\csname normal@char#2\endcsname#3%
  \else
    \bbl@info{Making #2 an active character}%
    \ifnum\mathcode`#2="8000
      \@namedef{normal@char#2}{%
        \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
    \else
      \@namedef{normal@char#2}{#3}%
    \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).

```
  \bbl@restoreactive{#2}%
  \AtBeginDocument{%
    \catcode`#2\active
    \if@filesw
      \immediate\write\@mainaux{\catcode`\string#2\active}%
    \fi}%
  \expandafter\bbl@add@special\csname#2\endcsname
  \catcode`#2\active
\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*⟩).

```
\let\bbl@tempa\@firstoftwo
\if\string^#2%
  \def\bbl@tempa{\noexpand\textormath}%
\else
  \ifx\bbl@mathnormal\@undefined\else
    \let\bbl@tempa\bbl@mathnormal
  \fi
\fi
\expandafter\edef\csname active@char#2\endcsname{%
  \bbl@tempa
    {\noexpand\if@safe@actives
       \noexpand\expandafter
       \expandafter\noexpand\csname normal@char#2\endcsname
     \noexpand\else
       \noexpand\expandafter
       \expandafter\noexpand\csname bbl@doactive#2\endcsname
     \noexpand\fi}%
   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
\bbl@csarg\edef{doactive#2}{%
  \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!).

```
\bbl@csarg\edef{active@#2}{%
  \noexpand\active@prefix\noexpand#1%
  \expandafter\noexpand\csname active@char#2\endcsname}%
\bbl@csarg\edef{normal@#2}{%
  \noexpand\active@prefix\noexpand#1%
  \expandafter\noexpand\csname normal@char#2\endcsname}%
\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.

```
\bbl@active@def#2\user@group{user@active}{language@active}%
```

```
  \bbl@active@def#2\language@group{language@active}{system@active}%
  \bbl@active@def#2\system@group{system@active}{normal@char}%
```

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

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

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

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

```
⟨⟨∗More package options⟩⟩ ≡
\DeclareOption{math=active}{}
\DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
⟨⟨/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`.

```
\@ifpackagewith{babel}{KeepShorthandsActive}%
  {\let\bbl@restoreactive\@gobble}%
  {\def\bbl@restoreactive#1{%
     \bbl@exp{%
       \\\AfterBabelLanguage\\\CurrentOption
         {\catcode`#1=\the\catcode`#1\relax}%
       \\\AtEndOfPackage
         {\catcode`#1=\the\catcode`#1\relax}}}%
   \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.

```
\def\bbl@sh@select#1#2{%
  \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
    \bbl@afterelse\bbl@scndcs
  \else
    \bbl@afterfi\csname#1@sh@#2@sel\endcsname
  \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.

```
\def\active@prefix#1{%
  \ifx\protect\@typeset@protect
  \else
```

When \protect is set to \@unexpandable@protect we make sure that the active character is als *not* expanded by inserting \noexpand in front of it. The \@gobble is needed to remove a token such as \activechar: (when the double colon was the active character to be dealt with).

```
    \ifx\protect\@unexpandable@protect
      \noexpand#1%
    \else
      \protect#1%
    \fi
    \expandafter\@gobble
  \fi}
```

\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*⟩.

```
\newif\if@safe@actives
\@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.

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

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

```
\def\bbl@activate#1{%
  \bbl@withactive{\expandafter\let\expandafter}#1%
    \csname bbl@active@\string#1\endcsname}
\def\bbl@deactivate#1{%
  \bbl@withactive{\expandafter\let\expandafter}#1%
    \csname bbl@normal@\string#1\endcsname}
```

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

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

```
\def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
\def\@decl@short#1#2#3\@nil#4{%
  \def\bbl@tempa{#3}%
  \ifx\bbl@tempa\@empty
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
    \bbl@ifunset{#1@sh@\string#2@}{}%
      {\def\bbl@tempa{#4}%
       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
       \else
         \bbl@info
           {Redefining #1 shorthand \string#2\\%
            in language \CurrentOption}%
       \fi}%
    \@namedef{#1@sh@\string#2@}{#4}%
  \else
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
    \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
      {\def\bbl@tempa{#4}%
       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
       \else
         \bbl@info
           {Redefining #1 shorthand \string#2\string#3\\%
            in language \CurrentOption}%
       \fi}%
    \@namedef{#1@sh@\string#2@\string#3@}{#4}%
  \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.

```
\def\textormath{%
  \ifmmode
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi}
```

`\user@group`
`\language@group`
`\system@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'.

```
\def\user@group{user}
\def\language@group{english}
\def\system@group{system}
```

`\useshorthands`

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

```
\def\useshorthands{%
  \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
\def\bbl@usesh@s#1{%
  \bbl@usesh@x
    {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
    {#1}}
\def\bbl@usesh@x#1#2{%
  \bbl@ifshorthand{#2}%
```

```
    {\def\user@group{user}%
     \initiate@active@char{#2}%
     #1%
     \bbl@activate{#2}}%
    {\bbl@error
       {Cannot declare a shorthand turned off (\string#2)}
       {Sorry, but you cannot use shorthands which have been\\%
        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.

```
\def\user@language@group{user@\language@group}
\def\bbl@set@user@generic#1#2{%
  \bbl@ifunset{user@generic@active#1}%
    {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
       \expandafter\noexpand\csname normal@char#1\endcsname}%
     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
       \expandafter\noexpand\csname user@active#1\endcsname}}%
  \@empty}
\newcommand\defineshorthand[3][user]{%
  \edef\bbl@tempa{\zap@space#1 \@empty}%
  \bbl@for\bbl@tempb\bbl@tempa{%
    \if*\expandafter\@car\bbl@tempb\@nil
      \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
      \@expandtwoargs
        \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
    \fi
    \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.

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

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

```
\def\aliasshorthand#1#2{%
  \bbl@ifshorthand{#2}%
    {\expandafter\ifx\csname active@char\string#2\endcsname\relax
       \ifx\document\@notprerr
         \@notshorthand{#2}%
       \else
         \initiate@active@char{#2}%
```

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

```
         \expandafter\let\csname active@char\string#2\expandafter\endcsname
           \csname active@char\string#1\endcsname
         \expandafter\let\csname normal@char\string#2\expandafter\endcsname
           \csname normal@char\string#1\endcsname
         \bbl@activate{#2}%
       \fi
```

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

`\@notshorthand`

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

`\shorthandon`
`\shorthandoff`

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

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

```
\def\bbl@switch@sh#1#2{%
  \ifx#2\@nnil\else
     \bbl@ifunset{bbl@active@\string#2}%
      {\bbl@error
         {I cannot switch `\string#2' on or off--not a shorthand}%
         {This character is not a shorthand. Maybe you made\\%
          a typing mistake? I will ignore your instruction}}%
      {\ifcase#1%
         \catcode`#212\relax
       \or
         \catcode`#2\active
       \or
         \csname bbl@oricat@\string#2\endcsname
         \csname bbl@oridef@\string#2\endcsname
       \fi}%
    \bbl@afterfi\bbl@switch@sh#1%
  \fi}
```

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

```
\def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
\def\bbl@putsh#1{%
  \bbl@ifunset{bbl@active@\string#1}%
     {\bbl@putsh@i#1\@empty\@nnil}%
     {\csname bbl@active@\string#1\endcsname}}
\def\bbl@putsh@i#1#2\@nnil{%
```

```
  \csname\languagename @sh@\string#1@%
    \ifx\@empty#2\else\string#2@\fi\endcsname}
\ifx\bbl@opt@shorthands\@nnil\else
  \let\bbl@s@initiate@active@char\initiate@active@char
  \def\initiate@active@char#1{%
    \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
  \let\bbl@s@switch@sh\bbl@switch@sh
  \def\bbl@switch@sh#1#2{%
    \ifx#2\@nnil\else
      \bbl@afterfi
      \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
    \fi}
  \let\bbl@s@activate\bbl@activate
  \def\bbl@activate#1{%
    \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
  \let\bbl@s@deactivate\bbl@deactivate
  \def\bbl@deactivate#1{%
    \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
\fi
```

\bbl@prim@s
\bbl@pr@m@s

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

```
\def\bbl@prim@s{%
  \prime\futurelet\@let@token\bbl@pr@m@s}
\def\bbl@if@primes#1#2{%
  \ifx#1\@let@token
    \expandafter\@firstoftwo
  \else\ifx#2\@let@token
    \bbl@afterelse\expandafter\@firstoftwo
  \else
    \bbl@afterfi\expandafter\@secondoftwo
  \fi\fi}
\begingroup
  \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
  \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
  \lowercase{%
    \gdef\bbl@pr@m@s{%
      \bbl@if@primes"'%
        \pr@@@s
        {\bbl@if@primes*^\pr@@@t\egroup}}}
\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).

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

\OT1dqpos
\T1dqpos

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

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

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

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

8.5 Language attributes

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

\languageattribute

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

```
\newcommand\languageattribute[2]{%
  \def\bbl@tempc{#1}%
  \bbl@fixname\bbl@tempc
  \bbl@iflanguage\bbl@tempc{%
    \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.

```
    \ifx\bbl@known@attribs\@undefined
      \in@false
    \else
```

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

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

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

```
    \ifin@
      \bbl@warning{%
        You have more than once selected the attribute '##1'\\%
        for language #1}%
    \else
```

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

```
      \bbl@exp{%
        \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
      \edef\bbl@tempa{\bbl@tempc-##1}%
      \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
      {\csname\bbl@tempc @attr@##1\endcsname}%
      {\@attrerr{\bbl@tempc}{##1}}%
   \fi}}}
```

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

```
\@onlypreamble\languageattribute
```

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

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

```
\def\bbl@declare@ttribute#1#2#3{%
  \bbl@xin@{,#2,}{,\BabelModifiers,}%
  \ifin@
    \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
  \fi
  \bbl@add@list\bbl@attributes{#1-#2}%
  \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.

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

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

```
  \ifx\bbl@known@attribs\@undefined
    \in@false
  \else
```

The we need to check the list of known attributes.

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

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

```
  \ifin@
    \bbl@afterelse#3%
  \else
    \bbl@afterfi#4%
  \fi
  }
```

`\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.

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

We first assume the attribute is unknown.

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

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

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

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

```
      \let\bbl@tempa\@firstoftwo
    \else
    \fi}%
```

Finally we execute `\bbl@tempa`.

```
  \bbl@tempa
}
```

`\bbl@clear@ttribs`

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

```
\def\bbl@clear@ttribs{%
  \ifx\bbl@attributes\@undefined\else
    \bbl@loopx\bbl@tempa{\bbl@attributes}{%
      \expandafter\bbl@clear@ttrib\bbl@tempa.
      }%
    \let\bbl@attributes\@undefined
  \fi}
\def\bbl@clear@ttrib#1-#2.{%
  \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
\AtBeginDocument{\bbl@clear@ttribs}
```

8.6 Support for saving macro definitions

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

`\babel@savecnt`
`\babel@beginsave`

The initialization of a new save cycle: reset the counter to zero.

```
\def\babel@beginsave{\babel@savecnt\z@}
```

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

```
\newcount\babel@savecnt
\babel@beginsave
```

`\babel@save`

The macro `\babel@save`⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to `\originalTeX`[27]. 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.

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

`\babel@savevariable`

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

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

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

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

[27] `\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```
\def\bbl@frenchspacing{%
  \ifnum\the\sfcode`\.=\@m
    \let\bbl@nonfrenchspacing\relax
  \else
    \frenchspacing
    \let\bbl@nonfrenchspacing\nonfrenchspacing
  \fi}
\let\bbl@nonfrenchspacing\nonfrenchspacing
```

8.7 Short tags

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

```
\def\babeltags#1{%
  \edef\bbl@tempa{\zap@space#1 \@empty}%
  \def\bbl@tempb##1=##2\@@{%
    \edef\bbl@tempc{%
      \noexpand\newcommand
      \expandafter\noexpand\csname ##1\endcsname{%
        \noexpand\protect
        \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
      \noexpand\newcommand
      \expandafter\noexpand\csname text##1\endcsname{%
        \noexpand\foreignlanguage{##2}}}
    \bbl@tempc}%
  \bbl@for\bbl@tempa\bbl@tempa{%
    \expandafter\bbl@tempb\bbl@tempa\@@}}
```

8.8 Hyphens

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

```
\@onlypreamble\babelhyphenation
\AtEndOfPackage{%
  \newcommand\babelhyphenation[2][\@empty]{%
    \ifx\bbl@hyphenation@\relax
      \let\bbl@hyphenation@\@empty
    \fi
    \ifx\bbl@hyphlist\@empty\else
      \bbl@warning{%
        You must not intermingle \string\selectlanguage\space and\\%
        \string\babelhyphenation\space or some exceptions will not\\%
        be taken into account. Reported}%
    \fi
    \ifx\@empty#1%
      \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
    \else
      \bbl@vforeach{#1}{%
        \def\bbl@tempa{##1}%
        \bbl@fixname\bbl@tempa
        \bbl@iflanguage\bbl@tempa{%
          \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
            \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
              \@empty
              {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
            #2}}}%
    \fi}}
```

\bbl@allowhyphens This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip 0pt plus 0pt[28].

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

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

```
\newcommand\babelnullhyphen{\char\hyphenchar\font}
\def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
\def\bbl@hyphen{%
  \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphen@i#1#2{%
  \bbl@ifunset{bbl@hy@#1#2\@empty}%
    {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
    {\csname bbl@hy@#1#2\@empty\endcsname}}
```

The following two commands are used to wrap the "hyphen" and set the behaviour of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphen 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 discretionaty 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.

```
\def\bbl@usehyphen#1{%
  \leavevmode
  \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
  \nobreak\hskip\z@skip}
\def\bbl@@usehyphen#1{%
  \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
\def\bbl@hyphenchar{%
  \ifnum\hyphenchar\font=\m@ne
    \babelnullhyphen
  \else
    \char\hyphenchar\font
  \fi}
```

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

```
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
\def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
\def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
\def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
\def\bbl@hy@repeat{%
  \bbl@usehyphen{%
    \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@@repeat{%
```

[28] TEX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```
  \bbl@@usehyphen{%
    \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@empty{\hskip\z@skip}
\def\bbl@hy@@empty{\discretionary{}{}{}}
```

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

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

8.9 Multiencoding strings

The aim following commands is to provide a commom interface for strings in several encodings. They also contains several hooks which can be ued 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.

```
\def\bbl@toglobal#1{\global\let#1#1}
\def\bbl@recatcode#1{%
  \@tempcnta="7F
  \def\bbl@tempa{%
    \ifnum\@tempcnta>"FF\else
      \catcode\@tempcnta=#1\relax
      \advance\@tempcnta\@ne
      \expandafter\bbl@tempa
    \fi}%
  \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).

```
\@ifpackagewith{babel}{nocase}%
  {\let\bbl@patchuclc\relax}%
  {\def\bbl@patchuclc{%
    \global\let\bbl@patchuclc\relax
    \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
    \gdef\bbl@uclc##1{%
      \let\bbl@encoded\bbl@encoded@uclc
      \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
        {##1}%
        {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
         \csname\languagename @bbl@uclc\endcsname}%
      {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
    \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
    \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{nocase}{}
⟨⟨/More package options⟩⟩
```

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

```
⟨⟨*More package options⟩⟩ ≡
\let\bbl@opt@strings\@nnil % accept strings=value
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\def\BabelStringsDefault{generic}
⟨⟨/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.

```
\@onlypreamble\StartBabelCommands
\def\StartBabelCommands{%
  \begingroup
  \bbl@recatcode{11}%
  ⟨⟨Macros local to BabelCommands⟩⟩
  \def\bbl@provstring##1##2{%
    \providecommand##1{##2}%
    \bbl@toglobal##1}%
  \global\let\bbl@scafter\@empty
  \let\StartBabelCommands\bbl@startcmds
  \ifx\BabelLanguages\relax
     \let\BabelLanguages\CurrentOption
  \fi
  \begingroup
  \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
  \StartBabelCommands}
\def\bbl@startcmds{%
  \ifx\bbl@screset\@nnil\else
    \bbl@usehooks{stopcommands}{}%
  \fi
  \endgroup
  \begingroup
  \@ifstar
    {\ifx\bbl@opt@strings\@nnil
       \let\bbl@opt@strings\BabelStringsDefault
     \fi
     \bbl@startcmds@i}%
    \bbl@startcmds@i}
\def\bbl@startcmds@i#1#2{%
  \edef\bbl@L{\zap@space#1 \@empty}%
  \edef\bbl@G{\zap@space#2 \@empty}%
  \bbl@startcmds@ii}
```

Parse the encoding info to get the label, input, and font parts.
Select the behaviour of `\SetString`. Thre 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.

```
\newcommand\bbl@startcmds@ii[1][\@empty]{%
  \let\SetString\@gobbletwo
  \let\bbl@stringdef\@gobbletwo
  \let\AfterBabelCommands\@gobble
  \ifx\@empty#1%
    \def\bbl@sc@label{generic}%
    \def\bbl@encstring##1##2{%
      \ProvideTextCommandDefault##1{##2}%
      \bbl@toglobal##1%
      \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
    \let\bbl@sctest\in@true
  \else
    \let\bbl@sc@charset\space % <- zapped below
    \let\bbl@sc@fontenc\space % <-   "      "
    \def\bbl@tempa##1=##2\@nil{%
      \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
    \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
    \def\bbl@tempa##1 ##2{% space -> comma
      ##1%
      \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
    \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
    \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
    \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
    \def\bbl@encstring##1##2{%
      \bbl@foreach\bbl@sc@fontenc{%
        \bbl@ifunset{T@####1}%
          {}%
          {\ProvideTextCommand##1{####1}{##2}%
           \bbl@toglobal##1%
           \expandafter
           \bbl@toglobal\csname####1\string##1\endcsname}}}%
    \def\bbl@sctest{%
      \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
  \fi
  \ifx\bbl@opt@strings\@nnil         % ie, no strings key -> defaults
  \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
    \let\AfterBabelCommands\bbl@aftercmds
    \let\SetString\bbl@setstring
    \let\bbl@stringdef\bbl@encstring
  \else       % ie, strings=value
  \bbl@sctest
  \ifin@
    \let\AfterBabelCommands\bbl@aftercmds
    \let\SetString\bbl@setstring
    \let\bbl@stringdef\bbl@provstring
  \fi\fi\fi
  \bbl@scswitch
  \ifx\bbl@G\@empty
    \def\SetString##1##2{%
      \bbl@error{Missing group for string \string##1}%
        {You must assign strings to some category, typically\\%
         captions or extras, but you set none}}%
  \fi
  \ifx\@empty#1%
    \bbl@usehooks{defaultcommands}{}%
  \else
    \@expandtwoargs
    \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
  \fi}
```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure \⟨*group*⟩⟨*language*⟩ is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing. The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in \BabelLanguages (inside babel) or \date⟨*language*⟩ is defined (after babel has been loaded). There are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded) .

```
\def\bbl@forlang#1#2{%
  \bbl@for#1\bbl@L{%
    \bbl@xin@{,#1,}{,\BabelLanguages,}%
    \ifin@#2\relax\fi}}
\def\bbl@scswitch{%
  \bbl@forlang\bbl@tempa{%
    \ifx\bbl@G\@empty\else
      \ifx\SetString\@gobbletwo\else
        \edef\bbl@GL{\bbl@G\bbl@tempa}%
        \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
        \ifin@\else
          \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
          \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
        \fi
      \fi
    \fi}}
\AtEndOfPackage{%
  \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
  \let\bbl@scswitch\relax}
\@onlypreamble\EndBabelCommands
\def\EndBabelCommands{%
  \bbl@usehooks{stopcommands}{}%
  \endgroup
  \endgroup
  \bbl@scafter}
```

Now we define commands to be used inside \StartBabelCommands.

Strings The following macro is the actual definition of \SetString when it is "active" First save the "switcher". Create it if undefined. Strings are defined only if undefined (ie, like \providescommmand). 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.

```
\def\bbl@setstring#1#2{%
  \bbl@forlang\bbl@tempa{%
    \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
    \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
      {\global\expandafter  % TODO - con \bbl@exp ?
       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
         {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
      {}%
    \def\BabelString{#2}%
    \bbl@usehooks{stringprocess}{}%
    \expandafter\bbl@stringdef
      \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}
```

Now, some addtional 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.

```
\ifx\bbl@opt@strings\relax
```

```
  \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
  \bbl@patchuclc
  \let\bbl@encoded\relax
  \def\bbl@encoded@uclc#1{%
    \@inmathwarn#1%
    \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
      \expandafter\ifx\csname ?\string#1\endcsname\relax
        \TextSymbolUnavailable#1%
      \else
        \csname ?\string#1\endcsname
      \fi
    \else
      \csname\cf@encoding\string#1\endcsname
    \fi}
\else
  \def\bbl@scset#1#2{\def#1{#2}}
\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.

```
⟨⟨*Macros local to BabelCommands⟩⟩ ≡
\def\SetStringLoop##1##2{%
    \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
    \count@\z@
    \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
      \advance\count@\@ne
      \toks@\expandafter{\bbl@tempa}%
      \bbl@exp{%
        \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
        \count@=\the\count@\relax}}}%
⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
\def\bbl@aftercmds#1{%
  \toks@\expandafter{\bbl@scafter#1}%
  \xdef\bbl@scafter{\the\toks@}}
```

Case mapping The command \SetCase provides a way to change the behaviour of \MakeUppercase and \MakeLowercase. \bbl@tempa is set by the patched \@uclclist to the parsing command.

```
⟨⟨*Macros local to BabelCommands⟩⟩ ≡
  \newcommand\SetCase[3][]{%
    \bbl@patchuclc
    \bbl@forlang\bbl@tempa{%
      \expandafter\bbl@encstring
        \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
      \expandafter\bbl@encstring
        \csname\bbl@tempa @bbl@uc\endcsname{##2}%
      \expandafter\bbl@encstring
        \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
⟨⟨/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.

```
⟨⟨*Macros local to BabelCommands⟩⟩ ≡
  \newcommand\SetHyphenMap[1]{%
    \bbl@forlang\bbl@tempa{%
      \expandafter\bbl@stringdef
        \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}
⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
\newcommand\BabelLower[2]{% one to one.
  \ifnum\lccode#1=#2\else
    \babel@savevariable{\lccode#1}%
    \lccode#1=#2\relax
  \fi}
\newcommand\BabelLowerMM[4]{% many-to-many
  \@tempcnta=#1\relax
  \@tempcntb=#4\relax
  \def\bbl@tempa{%
    \ifnum\@tempcnta>#2\else
      \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
      \advance\@tempcnta#3\relax
      \advance\@tempcntb#3\relax
      \expandafter\bbl@tempa
    \fi}%
  \bbl@tempa}
\newcommand\BabelLowerMO[4]{% many-to-one
  \@tempcnta=#1\relax
  \def\bbl@tempa{%
    \ifnum\@tempcnta>#2\else
      \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
      \advance\@tempcnta#3
      \expandafter\bbl@tempa
    \fi}%
  \bbl@tempa}
```

The following package options control the behaviour of hyphenation mapping.

```
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
\DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
\DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
\DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
\DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
⟨⟨/More package options⟩⟩
```

Initial setup to provide a default behaviour if hypenmap is not set.

```
\AtEndOfPackage{%
  \ifx\bbl@opt@hyphenmap\@undefined
    \bbl@xin@{,}{\bbl@language@opts}%
    \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
  \fi}
```

8.10 Macros common to a number of languages

\set@low@box

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

```
\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
    \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.

```
\def\save@sf@q#1{\leavevmode
  \begingroup
    \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
  \endgroup}
```

8.11 Making glyphs available

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

8.11.1 Quotation marks

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

```
\ProvideTextCommand{\quotedblbase}{OT1}{%
  \save@sf@q{\set@low@box{\textquotedblright\/}%
    \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.

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

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

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

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

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

```
\ProvideTextCommand{\guillemotleft}{OT1}{%
  \ifmmode
    \ll
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guillemotright}{OT1}{%
  \ifmmode
    \gg
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
  \fi}
```

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

```
\ProvideTextCommandDefault{\guillemotleft}{%
  \UseTextSymbol{OT1}{\guillemotleft}}
\ProvideTextCommandDefault{\guillemotright}{%
  \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\ProvideTextCommand{\guilsinglleft}{OT1}{%
  \ifmmode
    <%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guilsinglright}{OT1}{%
  \ifmmode
    >%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
  \fi}
```

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

```
\ProvideTextCommandDefault{\guilsinglleft}{%
  \UseTextSymbol{OT1}{\guilsinglleft}}
\ProvideTextCommandDefault{\guilsinglright}{%
  \UseTextSymbol{OT1}{\guilsinglright}}
```

8.11.2 Letters

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

```
\DeclareTextCommand{\ij}{OT1}{%
  i\kern-0.02em\bbl@allowhyphens j}
\DeclareTextCommand{\IJ}{OT1}{%
  I\kern-0.02em\bbl@allowhyphens J}
\DeclareTextCommand{\ij}{T1}{\char188}
\DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
\ProvideTextCommandDefault{\ij}{%
  \UseTextSymbol{OT1}{\ij}}
\ProvideTextCommandDefault{\IJ}{%
  \UseTextSymbol{OT1}{\IJ}}
```

\dj \DJ The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in the OT1 encoding by default.
Some code to construct these glyphs for the OT1 encoding was made available to me by Stipcevic Mario, (stipcevic@olimp.irb.hr).

```
\def\crrtic@{\hrule height0.1ex width0.3em}
\def\crttic@{\hrule height0.1ex width0.33em}
\def\ddj@{%
  \setbox0\hbox{d}\dimen@=\ht0
  \advance\dimen@1ex
  \dimen@.45\dimen@
  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
  \advance\dimen@ii.5ex
  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
\def\DDJ@{%
  \setbox0\hbox{D}\dimen@=.55\ht0
  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
  \advance\dimen@ii.15ex %            correction for the dash position
  \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
  \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
%
\DeclareTextCommand{\dj}{OT1}{\ddj@ d}
\DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
\ProvideTextCommandDefault{\dj}{%
  \UseTextSymbol{OT1}{\dj}}
\ProvideTextCommandDefault{\DJ}{%
  \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.

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

8.11.3 Shorthands for quotation marks

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

\glq
\grq
The 'german' single quotes.

```
\ProvideTextCommandDefault{\glq}{%
  \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
\ProvideTextCommand{\grq}{T1}{%
  \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
\ProvideTextCommand{\grq}{TU}{%
  \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
\ProvideTextCommand{\grq}{OT1}{%
  \save@sf@q{\kern-.0125em
    \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
    \kern.07em\relax}}
\ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq
\grqq
The 'german' double quotes.

```
\ProvideTextCommandDefault{\glqq}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
\ProvideTextCommand{\grqq}{T1}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\ProvideTextCommand{\grqq}{TU}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\ProvideTextCommand{\grqq}{OT1}{%
  \save@sf@q{\kern-.07em
    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
    \kern.07em\relax}}
\ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq \frq The ‘french’ single guillemets.

```
\ProvideTextCommandDefault{\flq}{%
  \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
\ProvideTextCommandDefault{\frq}{%
  \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq \frqq The ‘french’ double guillemets.

```
\ProvideTextCommandDefault{\flqq}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\ProvideTextCommandDefault{\frqq}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

8.11.4 Umlauts and tremas

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

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

```
\def\umlauthigh{%
  \def\bbl@umlauta##1{\leavevmode\bgroup%
    \expandafter\accent\csname\f@encoding dqpos\endcsname
    ##1\bbl@allowhyphens\egroup}%
  \let\bbl@umlaute\bbl@umlauta}
\def\umlautlow{%
  \def\bbl@umlauta{\protect\lower@umlaut}}
\def\umlautelow{%
  \def\bbl@umlaute{\protect\lower@umlaut}}
\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.

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

```
\def\lower@umlaut#1{%
  \leavevmode\bgroup
    \U@D 1ex%
    {\setbox\z@\hbox{%
      \expandafter\char\csname\f@encoding dqpos\endcsname}%
      \dimen@ -.45ex\advance\dimen@\ht\z@
      \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
    \expandafter\accent\csname\f@encoding dqpos\endcsname
    \fontdimen5\font\U@D #1%
  \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).

```
\AtBeginDocument{%
  \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
  \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
  \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
  \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
  \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
  \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
  \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
  \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
  \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
  \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
  \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
}
```

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

```
\ifx\l@english\@undefined
  \chardef\l@english\z@
\fi
\main@language{english}
```

Now we load definition files for engines.

```
\ifcase\bbl@engine
  \input txtbabel.def
\or
  \input luababel.def
\or
  \input xebabel.def
\fi
```

9 The kernel of Babel (`babel.def`, only LaTeX)

9.1 The redefinition of the style commands

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

```
{\def\format{lplain}
\ifx\fmtname\format
\else
  \def\format{LaTeX2e}
  \ifx\fmtname\format
  \else
    \aftergroup\endinput
  \fi
\fi}
```

9.2 Creating languages

\babelprovide is a general purpose tool for creating languages. Currently it just creates the language infrastructure, but in the future it will be able to read data from ini files, as well as to create variants. Unlike the nil pseudo-language, captions are defined, but with a warning to invite the user to provide the real string.

```
\newcommand\babelprovide[2][]{%
  \let\bbl@savelangname\languagename
  \def\languagename{#2}%
  \let\bbl@KVP@captions\@nil
  \let\bbl@KVP@import\@nil
  \let\bbl@KVP@main\@nil
  \let\bbl@KVP@script\@nil
  \let\bbl@KVP@language\@nil
  \let\bbl@KVP@dir\@nil
  \let\bbl@KVP@hyphenrules\@nil
  \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}%  TODO - error handling
  \ifx\bbl@KVP@captions\@nil
    \let\bbl@KVP@captions\bbl@KVP@import
  \fi
  \bbl@ifunset{date#2}%
    {\bbl@provide@new{#2}}%
    {\bbl@ifblank{#1}%
      {\bbl@error
        {If you want to modify `#2' you must tell how in\\%
         the optional argument. Currently there are three\\%
         options: captions=lang-tag, hyphenrules=lang-list\\%
         import=lang-tag}%
        {Use this macro as documented}}%
      {\bbl@provide@renew{#2}}}%
  \bbl@exp{\\\babelensure[exclude=\\\today]{#2}}%
  \ifx\bbl@KVP@script\@nil\else
    \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
  \fi
  \ifx\bbl@KVP@language\@nil\else
    \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
  \fi
  \let\languagename\bbl@savelangname}
```

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

```
\def\bbl@provide@new#1{%
  \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
  \@namedef{extras#1}{}%
  \@namedef{noextras#1}{}%
  \StartBabelCommands*{#1}{captions}%
    \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
      \def\bbl@tempb##1{%             elt for \bbl@captionslist
        \ifx##1\@empty\else
          \bbl@exp{%
            \\\SetString\\##1{%
              \\\bbl@nocaption{\bbl@stripslash##1}{\<#1\bbl@stripslash##1>}}}%
          \expandafter\bbl@tempb
        \fi}%
      \expandafter\bbl@tempb\bbl@captionslist\@empty
    \else
```

```
      \bbl@read@ini{\bbl@KVP@captions}%  Here all letters cat = 11
      \bbl@after@ini
      \bbl@savestrings
    \fi
  \StartBabelCommands*{#1}{date}%
    \ifx\bbl@KVP@import\@nil
      \bbl@exp{%
        \\\SetString\\\today{\\\bbl@nocaption{today}{\<#1today>}}}%
    \else
      \bbl@savetoday
      \bbl@savedate
    \fi
  \EndBabelCommands
  \bbl@exp{%
    \def\<#1hyphenmins>{%
      {\bbl@ifunset{bbl@lfthm@#1}{2}{\@nameuse{bbl@lfthm@#1}}}%
      {\bbl@ifunset{bbl@rgthm@#1}{3}{\@nameuse{bbl@rgthm@#1}}}}}%
  \bbl@provide@hyphens{#1}%
  \ifx\bbl@KVP@main\@nil\else
     \expandafter\main@language\expandafter{#1}%
  \fi}
\def\bbl@provide@renew#1{%
  \ifx\bbl@KVP@captions\@nil\else
    \StartBabelCommands*{#1}{captions}%
      \bbl@read@ini{\bbl@KVP@captions}%   Here all letters cat = 11
      \bbl@after@ini
      \bbl@savestrings
    \EndBabelCommands
 \fi
 \ifx\bbl@KVP@import\@nil\else
   \StartBabelCommands*{#1}{date}%
     \bbl@savetoday
     \bbl@savedate
   \EndBabelCommands
  \fi
  \bbl@provide@hyphens{#1}}
```

The hyphenrules option is handled with an auxiliary macro.

```
\def\bbl@provide@hyphens#1{%
  \let\bbl@tempa\relax
  \ifx\bbl@KVP@hyphenrules\@nil\else
    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
    \bbl@foreach\bbl@KVP@hyphenrules{%
      \ifx\bbl@tempa\relax    %  if not yet found
        \bbl@ifsamestring{##1}{+}%
          {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
          {}%
        \bbl@ifunset{l@##1}%
           {}%
           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
      \fi}%
  \fi
  \ifx\bbl@tempa\relax %        if no opt or no language in opt found
    \ifx\bbl@KVP@import\@nil\else % if importing
      \bbl@exp{%                and hyphenrules is not empty
        \\\bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
          {}%
          {\let\\\bbl@tempa\<l@\@nameuse{bbl@hyphr@\languagename}>}}%
    \fi
  \fi
  \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
    {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
       {}}%                      so, l@<lang> is ok - nothing to do
    {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}%  found in opt list or ini
```

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

```
\def\bbl@read@ini#1{%
  \openin1=babel-#1.ini
  \ifeof1
    \bbl@error
      {There is no ini file for the requested language\\%
       (#1). Perhaps you misspelled it or your installation\\%
       is not complete.}%
      {Fix the name or reinstall babel.}%
  \else
    \let\bbl@section\@empty
    \let\bbl@savestrings\@empty
    \let\bbl@savetoday\@empty
    \let\bbl@savedate\@empty
    \let\bbl@inireader\bbl@iniskip
    \bbl@info{Importing data from babel-#1.ini for \languagename}%
    \loop
      \endlinechar\m@ne
      \read1 to \bbl@line
      \endlinechar`\^^M
    \if T\ifeof1F\fi T\relax % Trick, because inside \loop
      \ifx\bbl@line\@empty\else
        \expandafter\bbl@iniline\bbl@line\bbl@iniline
      \fi
    \repeat
  \fi}
\def\bbl@iniline#1\bbl@iniline{%
  \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inireader}#1\@@}% ]
```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the posibility 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.

```
\def\bbl@iniskip#1\@@{}%      if starts with ;
\def\bbl@inisec[#1]#2\@@{%    if starts with opening bracket
  \@nameuse{bbl@secpost@\bbl@section}%  ends previous section
  \def\bbl@section{#1}%
  \@nameuse{bbl@secpre@\bbl@section}%   starts current section
  \bbl@ifunset{bbl@secline@#1}%
    {\let\bbl@inireader\bbl@iniskip}%
    {\bbl@exp{\let\\\bbl@inireader\<bbl@secline@#1>}}}
```

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

```
\def\bbl@inikv#1=#2\@@{%     key=value
  \bbl@trim@def\bbl@tempa{#1}%
  \bbl@trim\toks@{#2}%
  \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.

```
\def\bbl@exportkey#1#2#3{%
  \bbl@ifunset{bbl@@kv@#2}%
```

```
    {\bbl@csarg\gdef{#1@\languagename}{#3}}%
    {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
       \bbl@csarg\gdef{#1@\languagename}{#3}%
     \else
       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
     \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`.

```
\let\bbl@secline@identification\bbl@inikv
\def\bbl@secpost@identification{%
  \bbl@exportkey{lname}{identification.name.english}{}%
  \bbl@exportkey{lbcp}{identification.tag.bcp47}{}%
  \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
  \bbl@exportkey{sname}{identification.script.name}{}%
  \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
  \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
\let\bbl@secline@typography\bbl@inikv
\def\bbl@after@ini{%
  \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
  \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
  \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
  \def\bbl@tempa{0.9}%
  \bbl@csarg\ifx{@kv@identification.version}\bbl@tempa
    \bbl@warning{%
      The `\languagename' date format may not be suitable\\%
      for proper typesetting, and therefore it very likely will\\%
      change in a future release. Reported}%
  \fi
  \bbl@toglobal\bbl@savetoday
  \bbl@toglobal\bbl@savedate}
```

Now `captions` and `captions.licr`, depending on the engine. And also for dates. They rely on a few auxilary macros.

```
\ifcase\bbl@engine
  \bbl@csarg\def{secline@captions.licr}#1=#2\@@{%
    \bbl@ini@captions@aux{#1}{#2}}
  \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%       for defaults
    \bbl@ini@dategreg#1...\relax{#2}}
  \bbl@csarg\def{secline@date.gregorian.licr}#1=#2\@@{%  override
    \bbl@ini@dategreg#1...\relax{#2}}
\else
  \def\bbl@secline@captions#1=#2\@@{%
    \bbl@ini@captions@aux{#1}{#2}}
  \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%
    \bbl@ini@dategreg#1...\relax{#2}}
\fi
```

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

```
\def\bbl@ini@captions@aux#1#2{%
  \bbl@trim@def\bbl@tempa{#1}%
  \bbl@ifblank{#2}%
    {\bbl@exp{%
       \toks@{\\\bbl@nocaption{\bbl@tempa}\<\languagename\bbl@tempa name>}}}%
    {\bbl@trim\toks@{#2}}%
  \bbl@exp{%
    \\\bbl@add\\\bbl@savestrings{%
      \\\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.

```
\bbl@csarg\def{secpre@date.gregorian.licr}{%
  \ifcase\bbl@engine\let\bbl@savedate\@empty\fi}
\def\bbl@ini@dategreg#1.#2.#3.#4\relax#5{% TODO - ignore with 'captions'
  \bbl@trim@def\bbl@tempa{#1.#2}%
  \bbl@ifsamestring{\bbl@tempa}{months.wide}%
    {\bbl@trim@def\bbl@tempa{#3}%
     \bbl@trim\toks@{#5}%
     \bbl@exp{%
      \\\bbl@add\\\bbl@savedate{%
        \\\SetString\<month\romannumeral\bbl@tempa name>{\the\toks@}}}}%
    {\bbl@ifsamestring{\bbl@tempa}{date.long}%
      {\bbl@trim@def\bbl@toreplace{#5}%
       \bbl@TG@@date
       \global\bbl@csarg\let{date@\languagename}\bbl@toreplace
       \bbl@exp{%
         \gdef\<\languagename date>####1####2####3{%
           \<bbl@ensure@\languagename>{%
             \<bbl@date@\languagename>{####1}{####2}{####3}}}%
         \\\bbl@add\\\bbl@savetoday{%
           \\\SetString\\\today{%
             \<\languagename date>{\year}{\month}{\day}}}}}}%
      {}}
```

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” inconsistenly in either in the date or in the month name.

```
\newcommand\BabelDateSpace{\nobreakspace{}}
\newcommand\BabelDateDot{.\@}
\newcommand\BabelDated[1]{{\number#1}}
\newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
\newcommand\BabelDateM[1]{{\number#1}}
\newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
\newcommand\BabelDateMMMM[1]{{%
  \csname month\romannumeral\month name\endcsname}}%
\newcommand\BabelDatey[1]{{\number#1}}%
\newcommand\BabelDateyy[1]{{%
  \ifnum#1<10 0\number#1 %
  \else\ifnum#1<100 \number#1 %
  \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
  \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
  \else
    \bbl@error
      {Currently two-digit years are restricted to the\\
       range 0-9999.}%
      {There is little you can do. Sorry.}%
  \fi\fi\fi\fi}}
\newcommand\BabelDateyyyy[1]{{\number#1}}
\def\bbl@replace@finish@iii#1{%
  \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
\def\bbl@TG@@date{%
  \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
  \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
  \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
  \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
  \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
  \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
  \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
  \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
  \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
  \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
% Note after \bbl@replace \toks@ contains the resulting string.
% TODO - Using this implicit behavior doesn't seem a good idea.
  \bbl@replace@finish@iii\bbl@toreplace}
```

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

```
\def\bbl@provide@lsys#1{%
  \bbl@ifunset{bbl@lname@#1}%
    {\bbl@ini@ids{#1}}%
    {}%
  \bbl@csarg\let{lsys@#1}\@empty
  \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
  \bbl@ifunset{bbl@sotf#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
  \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
  \bbl@ifunset{bbl@lname@#1}{}%
    {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
  \bbl@csarg\bbl@toglobal{lsys@#1}}%
 %  \bbl@exp{% TODO - should be global
 %    \<keys_if_exist:nnF>{fontspec-opentype/Script}{\bbl@cs{sname@#1}}%
 %      {\\\newfontscript{\bbl@cs{sname@#1}}{\bbl@cs{sotf@#1}}}%
 %    \<keys_if_exist:nnF>{fontspec-opentype/Language}{\bbl@cs{lname@#1}}%
 %      {\\\newfontlanguage{\bbl@cs{lname@#1}}{\bbl@cs{lotf@#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.

```
\def\bbl@ini@ids#1{%
  \def\BabelBeforeIni##1##2{%
    \begingroup
      \bbl@add\bbl@secpost@identification{%
        \def\bbl@iniline########1\bbl@iniline{}}%
      \catcode`\[=12 \catcode`\]=12 \catcode`\==12
      \bbl@read@ini{##1}%
    \endgroup}%           boxed, to avoid extra spaces:
  \setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}{}}}
```

9.3 Cross referencing macros

The LaTeX book states:

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

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.
When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category 'letter' or 'other'.
The only way to accomplish this in most cases is to use the trick described in the TeXbook [2] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, '`\meaning\A`' with `\A` defined as '`\def\A#1{\B}`' expands to the characters '`macro:#1->\B`' with all category codes set to 'other' or 'space'.

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

```
%\bbl@redefine\newlabel#1#2{%
%  \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

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

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

```
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
\DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
\DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
⟨⟨/More package options⟩⟩
```

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

```
\ifx\bbl@opt@safe\@empty\else
  \def\@newl@bel#1#2#3{%
   {\@safe@activestrue
    \bbl@ifunset{#1@#2}%
       \relax
       {\gdef\@multiplelabels{%
          \@latex@warning@no@line{There were multiply-defined labels}}%
        \@latex@warning@no@line{Label `#2' multiply defined}}%
    \global\@namedef{#1@#2}{#3}}}
```

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

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

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

```
  \def\@testdef#1#2#3{%
    \@safe@activestrue
```

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

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

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

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

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

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

We do the same for \bbl@tempb.

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

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

```
    \ifx\bbl@tempa\bbl@tempb
    \else
      \@tempswatrue
    \fi}
\fi
```

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

```
\bbl@xin@{R}\bbl@opt@safe
\ifin@
  \bbl@redefinerobust\ref#1{%
    \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
  \bbl@redefinerobust\pageref#1{%
    \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
\else
  \let\org@ref\ref
  \let\org@pageref\pageref
\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.

```
\bbl@xin@{B}\bbl@opt@safe
\ifin@
  \bbl@redefine\@citex[#1]#2{%
    \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
    \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.

```
  \AtBeginDocument{%
    \@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.)

```
    \def\@citex[#1][#2]#3{%
      \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
      \org@@citex[#1][#2]{\@tempa}}%
    }{}}
```

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

```
  \AtBeginDocument{%
    \@ifpackageloaded{cite}{%
      \def\@citex[#1]#2{%
```

```
        \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
      }{}}
```

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

```
  \bbl@redefine\nocite#1{%
    \@safe@activestrue\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@activestrue 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.

```
  \bbl@redefine\bibcite{%
```

We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

```
    \bbl@cite@choice
    \bibcite}
```

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

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

\bbl@cite@choice The macro \bbl@cite@choice determines which definition of \bibcite is needed.

```
  \def\bbl@cite@choice{%
```

First we give \bibcite its default definition.

```
    \global\let\bibcite\bbl@bibcite
```

Then, when natbib is loaded we restore the original definition of \bibcite.

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

For cite we do the same.

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

Make sure this only happens once.

```
    \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.

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

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

```
  \bbl@redefine\@bibitem#1{%
    \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
\else
  \let\org@nocite\nocite
  \let\org@@citex\@citex
  \let\org@bibcite\bibcite
  \let\org@@bibitem\@bibitem
\fi
```

9.4 Marks

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

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

```
\bbl@redefine\markright#1{%
  \bbl@ifblank{#1}%
    {\org@markright{}}%
    {\toks@{#1}%
     \bbl@exp{%
       \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
         {\\\protect\\\bbl@restore@actives\the\toks@}}}}}
```

\markboth
\@mkboth 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 neeed to do that again with the new definition of \markboth.

```
\ifx\@mkboth\markboth
  \def\bbl@tempc{\let\@mkboth\markboth}
\else
  \def\bbl@tempc{}
\fi
```

Now we can start the new definition of \markboth

```
\bbl@redefine\markboth#1#2{%
  \protected@edef\bbl@tempb##1{%
    \protect\foreignlanguage{\languagename}{\protect\bbl@restore@actives##1}}%
  \bbl@ifblank{#1}%
    {\toks@{}}%
    {\toks@\expandafter{\bbl@tempb{#1}}}%
  \bbl@ifblank{#2}%
    {\@temptokena{}}%
    {\@temptokena\expandafter{\bbl@tempb{#2}}}%
  \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
```

and copy it to \@mkboth if necessary.

```
\bbl@tempc
```

9.5 Preventing clashes with other packages

9.5.1 ifthen

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

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

In order for this to work the argument of `\isodd` needs to be fully expandable. With the above redefinition of `\pageref` it is not in the case of this example. To overcome that, we add some code to the definition of `\ifthenelse` to make things work.
The first thing we need to do is check if the package `ifthen` is loaded. This should be done at `\begin{document}` time.

```
\bbl@xin@{R}\bbl@opt@safe
\ifin@
  \AtBeginDocument{%
    \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```
      \bbl@redefine@long\ifthenelse#1#2#3{%
```

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

```
        \let\bbl@temp@pref\pageref
        \let\pageref\org@pageref
        \let\bbl@temp@ref\ref
        \let\ref\org@ref
```

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

```
        \@safe@activestrue
        \org@ifthenelse{#1}%
          {\let\pageref\bbl@temp@pref
           \let\ref\bbl@temp@ref
           \@safe@activesfalse
           #2}%
          {\let\pageref\bbl@temp@pref
           \let\ref\bbl@temp@ref
           \@safe@activesfalse
           #3}%
        }%
      }{}%
    }
```

9.5.2 `varioref`

`\@@vpageref`
`\vrefpagenum`
`\Ref`

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

```
  \AtBeginDocument{%
    \@ifpackageloaded{varioref}{%
      \bbl@redefine\@@vpageref#1[#2]#3{%
        \@safe@activestrue
        \org@@@vpageref{#1}[#2]{#3}%
        \@safe@activesfalse}%
```

The same needs to happen for `\vrefpagenum`.

```
      \bbl@redefine\vrefpagenum#1#2{%
        \@safe@activestrue
        \org@vrefpagenum{#1}{#2}%
        \@safe@activesfalse}%
```

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

```
      \expandafter\def\csname Ref \endcsname#1{%
        \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
      }{}%
    }
\fi
```

9.5.3 hhline

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

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

```
\AtEndOfPackage{%
  \AtBeginDocument{%
    \@ifpackageloaded{hhline}%
```

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

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

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

```
         \makeatletter
         \def\@currname{hhline}\input{hhline.sty}\makeatother
       \fi}%
      {}}}
```

9.5.4 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 removed for the moment because hyperref is expecting it.

```
\AtBeginDocument{%
  \ifx\pdfstringdefDisableCommands\@undefined\else
    \pdfstringdefDisableCommands{\languageshorthands{system}}%
  \fi}
```

9.5.5 fancyhdr

\FOREIGNLANGUAGE The package fancyhdr treats the running head and fout 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.

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

```
\def\substitutefontfamily#1#2#3{%
  \lowercase{\immediate\openout15=#1#2.fd\relax}%
  \immediate\write15{%
    \string\ProvidesFile{#1#2.fd}%
    [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
     \space generated font description file]^^J
    \string\DeclareFontFamily{#1}{#2}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
    }%
  \closeout15
  }
```

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

```
\@onlypreamble\substitutefontfamily
```

9.6 Encoding and fonts

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

```
\newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,}
\let\org@TeX\TeX
\let\org@LaTeX\LaTeX
\let\ensureascii\@firstofone
\AtBeginDocument{%
  \in@false
  \bbl@foreach\BabelNonASCII{% is there a non-ascii enc?
    \ifin@\else
      \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
    \fi}%
  \ifin@ % if a non-ascii has been loaded
    \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
    \DeclareTextCommandDefault{\TeX}{\org@TeX}%
    \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
    \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
    \def\bbl@tempc#1ENC.DEF#2\@@{%
      \ifx\@empty#2\else
        \bbl@ifunset{T@#1}%
          {}%
          {\bbl@xin@{,#1,}{,\BabelNonASCII,}%
           \ifin@
             \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
             \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
           \else
             \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
           \fi}%
      \fi}%
    \bbl@foreach\@filelist{\bbl@tempb#1\@@}%  TODO - \@@ de mas??
    \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,}%
    \ifin@\else
      \edef\ensureascii#1{{%
        \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
    \fi
  \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.

```
\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.

```
\AtBeginDocument{%
  \@ifpackageloaded{fontspec}%
    {\xdef\latinencoding{%
       \ifx\UTFencname\@undefined
         EU\ifcase\bbl@engine\or2\or1\fi
       \else
         \UTFencname
       \fi}}%
    {\gdef\latinencoding{OT1}%
     \ifx\cf@encoding\bbl@t@one
       \xdef\latinencoding{\bbl@t@one}%
     \else
       \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
     \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.

```
\DeclareRobustCommand{\latintext}{%
  \fontencoding{\latinencoding}\selectfont
  \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.

```
\ifx\@undefined\DeclareTextFontCommand
  \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
\else
  \DeclareTextFontCommand{\textlatin}{\latintext}
\fi
```

9.7 Basic bidi support

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

- pdftex provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.

- xetex is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour TEX grouping.

- luatex can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As LuaTEX-ja shows, vertical typesetting is posible, too. Its main drawback is font handling is often considered to be less mature than xetex (but there are steps to make HarfBuzz, the xetex font engine, available in luatex).

```
\def\bbl@alscripts{,Arabic,Syriac,Thaana,}
\def\bbl@rscripts{%
  ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
  Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
  Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
  Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
  Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
  Old South Arabian,}%
\def\bbl@provide@dirs#1{%
  \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
  \ifin@
    \global\bbl@csarg\chardef{wdir@#1}\@ne
    \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
    \ifin@
      \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
    \fi
  \else
    \global\bbl@csarg\chardef{wdir@#1}\z@
  \fi}
\def\bbl@switchdir{%
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
  \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
  \bbl@exp{\\\bbl@setdirs\bbl@cs{wdir@\languagename}}}
\def\bbl@setdirs#1{% TODO - math
  \ifcase\bbl@select@type % TODO - strictly, not the right test
    \bbl@bodydir{#1}%
    \bbl@pardir{#1}%
  \fi
  \bbl@textdir{#1}}
\ifodd\bbl@engine  % luatex=1
  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
  \DisableBabelHook{babel-bidi}
  \def\bbl@getluadir#1{%
    \directlua{
      if tex.#1dir == 'TLT' then
        tex.sprint('0')
      elseif tex.#1dir == 'TRT' then
        tex.sprint('1')
      end}}
  \def\bbl@setdir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
    \ifcase#3\relax
      \ifcase\bbl@getluadir{#1}\relax\else
        #2 TLT\relax
      \fi
    \else
      \ifcase\bbl@getluadir{#1}\relax
        #2 TRT\relax
      \fi
    \fi}
  \def\bbl@textdir#1{%
    \bbl@setdir{text}\textdir{#1}% TODO - ?\linedir
    \setattribute\bbl@attr@dir{#1}}
  \def\bbl@pardir{\bbl@setdir{par}\pardir}
  \def\bbl@bodydir{\bbl@setdir{body}\bodydir}
  \def\bbl@pagedir{\bbl@setdir{page}\pagedir}
  \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%%
\else % pdftex=0, xetex=2
  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
  \DisableBabelHook{babel-bidi}
  \newcount\bbl@dirlevel
  \chardef\bbl@thetextdir\z@
  \chardef\bbl@thepardir\z@
  \def\bbl@textdir#1{%
    \ifcase#1\relax
       \chardef\bbl@thetextdir\z@
       \bbl@textdir@i\beginL\endL
     \else
       \chardef\bbl@thetextdir\@ne
       \bbl@textdir@i\beginR\endR
    \fi}
  \def\bbl@textdir@i#1#2{%
    \ifhmode
      \ifnum\currentgrouplevel>\z@
        \ifnum\currentgrouplevel=\bbl@dirlevel
          \bbl@error{Multiple bidi settings inside a group}%
            {I'll insert a new group, but expect wrong results.}%
          \bgroup\aftergroup#2\aftergroup\egroup
        \else
          \ifcase\currentgrouptype\or % 0 bottom
            \aftergroup#2% 1 simple {}
          \or
            \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
          \or
            \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
          \or\or\or % vbox vtop align
          \or
            \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
          \or\or\or\or\or\or % output math disc insert vcent mathchoice
          \or
            \aftergroup#2% 14 \begingroup
          \else
            \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
          \fi
        \fi
        \bbl@dirlevel\currentgrouplevel
      \fi
      #1%
    \fi}
  \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
  \let\bbl@bodydir\@gobble
  \let\bbl@pagedir\@gobble
  \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}
```

The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled.

```
  \def\bbl@xebidipar{%
```

```
    \let\bbl@xebidipar\relax
    \TeXXeTstate\@ne
    \def\bbl@xeeverypar{%
      \ifcase\bbl@thepardir\else
        {\setbox\z@\lastbox\beginR\box\z@}%
      \fi
      \ifcase\bbl@thetextdir\else\beginR\fi}%
    \let\bbl@severypar\everypar
    \newtoks\everypar
    \everypar=\bbl@severypar
    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
\fi
```

A tool for weak L (mainly digits).

```
\DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
```

9.8 Layout

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

```
\newcommand\BabelPatchSection[1]{%
  \@ifundefined{#1}{}{%
    \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
    \@namedef{#1}{%
      \@ifstar{\bbl@presec@s{#1}}%
              {\@dblarg{\bbl@presec@x{#1}}}}}}
\def\bbl@presec@x#1[#2]#3{%
  \let\bbl@savetemplang\languagename
  \select@language@x\bbl@main@language
  \@nameuse{bbl@ss@#1}%
    [\foreignlanguage{\bbl@savetemplang}{#2}]%
    {\foreignlanguage{\bbl@savetemplang}{#3}}%
  \select@language@x\bbl@savetemplang}
\def\bbl@presec@s#1#2{%
  \let\bbl@savetemplang\languagename
  \select@language@x\bbl@main@language
  \@nameuse{bbl@ss@#1}*{\foreignlanguage{\languagename}{#2}}%
  \select@language@x\bbl@savetemplang}
\IfBabelLayout{sectioning}%   at begin document ???
  {\BabelPatchSection{part}%
   \BabelPatchSection{chapter}%
   \BabelPatchSection{section}%
   \BabelPatchSection{subsection}%
   \BabelPatchSection{subsubsection}%
   \BabelPatchSection{paragraph}%
   \BabelPatchSection{subparagraph}}{}
```

9.9 Local Language Configuration

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

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

```
\ifx\loadlocalcfg\@undefined
```

```
  \@ifpackagewith{babel}{noconfigs}%
    {\let\loadlocalcfg\@gobble}%
    {\def\loadlocalcfg#1{%
      \InputIfFileExists{#1.cfg}%
        {\typeout{*************************************^^J%
               * Local config file #1.cfg used^^J%
               *}}%
        \@empty}}
\fi
```

Just to be compatible with LaTeX 2.09 we add a few more lines of code:

```
\ifx\@unexpandable@protect\@undefined
  \def\@unexpandable@protect{\noexpand\protect\noexpand}
  \long\def\protected@write#1#2#3{%
    \begingroup
      \let\thepage\relax
      #2%
      \let\protect\@unexpandable@protect
      \edef\reserved@a{\write#1{#3}}%
      \reserved@a
    \endgroup
    \if@nobreak\ifvmode\nobreak\fi\fi}
\fi
⟨/core⟩
```

10 Multiple languages (switch.def)

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.

```
⟨*kernel⟩
⟨⟨Make sure ProvidesFile is defined⟩⟩
\ProvidesFile{switch.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel switching mechanism]
⟨⟨Load macros for plain if not LaTeX⟩⟩
⟨⟨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.

```
\def\bbl@version{⟨⟨version⟩⟩}
\def\bbl@date{⟨⟨date⟩⟩}
\def\adddialect#1#2{%
  \global\chardef#1#2\relax
  \bbl@usehooks{adddialect}{{#1}{#2}}%
  \wlog{\string#1 = a dialect from \string\language#2}}
```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises and error. The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intented 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.

```
\def\bbl@fixname#1{%
  \begingroup
    \def\bbl@tempe{l@}%
    \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
    \bbl@tempd
      {\lowercase\expandafter{\bbl@tempd}%
         {\uppercase\expandafter{\bbl@tempd}%
           \@empty
           {\edef\bbl@tempd{\def\noexpand#1{#1}}%
            \uppercase\expandafter{\bbl@tempd}}}%
         {\edef\bbl@tempd{\def\noexpand#1{#1}}%
          \lowercase\expandafter{\bbl@tempd}}}%
      \@empty
    \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
  \bbl@tempd}
\def\bbl@iflanguage#1{%
  \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}
```

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

```
\def\iflanguage#1{%
  \bbl@iflanguage{#1}{%
    \ifnum\csname l@#1\endcsname=\language
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}}
```

10.1 Selecting the language

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

To allow the call of \selectlanguage either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character. To convert a control sequence to a string, we use the \string primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer \escapechar to a character number, we have to compare this number with the character of the string. To do this we have to use TeX's backquote notation to specify the character as a number.

If the first character of the \string'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or \escapechar is set to a value outside of the character range 0–255.

If the user gives an empty argument, we provide a default argument for \string. This argument should expand to nothing.

```
\let\bbl@select@type\z@
\edef\selectlanguage{%
  \noexpand\protect
  \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.

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

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

```
\ifx\documentclass\@undefined
  \def\xstring{\string\string\string}
\else
  \let\xstring\string
\fi
```

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

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

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

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

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

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

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

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

```
\let\bbl@ifrestoring\@secondoftwo
\def\bbl@pop@language{%
  \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
  \let\bbl@ifrestoring\@firstoftwo
  \expandafter\bbl@set@language\expandafter{\languagename}%
  \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.

```
\expandafter\def\csname selectlanguage \endcsname#1{%
  \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
  \bbl@push@language
  \aftergroup\bbl@pop@language
  \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 historial 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 \languagename are not well defined. 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.

```
\def\BabelContentsFiles{toc,lof,lot}
\def\bbl@set@language#1{%
  \edef\languagename{%
    \ifnum\escapechar=\expandafter`\string#1\@empty
    \else\string#1\@empty\fi}%
  \select@language{\languagename}%
  \expandafter\ifx\csname date\languagename\endcsname\relax\else
    \if@filesw
      \protected@write\@auxout{}{\string\babel@aux{\languagename}{}}%
      \bbl@usehooks{write}{}%
    \fi
  \fi}
\def\select@language#1{%
  \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
  \edef\languagename{#1}%
  \bbl@fixname\languagename
  \bbl@iflanguage\languagename{%
    \expandafter\ifx\csname date\languagename\endcsname\relax
      \bbl@error
        {Unknown language `#1'. Either you have\\%
         misspelled its name, it has not been installed,\\%
         or you requested it in a previous run. Fix its name,\\%
         install it or just rerun the file, respectively}%
        {You may proceed, but expect wrong results}%
    \else
      \let\bbl@select@type\z@
      \expandafter\bbl@switch\expandafter{\languagename}%
    \fi}}
\def\babel@aux#1#2{%
  \select@language{#1}%
  \bbl@foreach\BabelContentsFiles{%
    \@writefile{##1}{\babel@toc{#1}{#2}}}} %% TODO - ok in plain? \string?
\def\babel@toc#1#2{%
  \select@language{#1}{#2}}
```

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

```
\let\select@language@x\select@language
```

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

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

Then we have to *re*define \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.

```
\def\bbl@switch#1{%
  \originalTeX
  \expandafter\def\expandafter\originalTeX\expandafter{%
    \csname noextras#1\endcsname
    \let\originalTeX\@empty
    \babel@beginsave}%
  \bbl@usehooks{afterreset}{}%
  \languageshorthands{none}%
  \ifcase\bbl@select@type
    \ifhmode
      \hskip\z@skip % trick to ignore spaces
      \csname captions#1\endcsname\relax
      \csname date#1\endcsname\relax
      \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
    \else
      \csname captions#1\endcsname\relax
      \csname date#1\endcsname\relax
    \fi
  \fi
  \bbl@usehooks{beforeextras}{}%
  \csname extras#1\endcsname\relax
  \bbl@usehooks{afterextras}{}%
  \ifcase\bbl@opt@hyphenmap\or
    \def\BabelLower##1##2{\lccode##1=##2\relax}%
    \ifnum\bbl@hymapsel>4\else
      \csname\languagename @bbl@hyphenmap\endcsname
    \fi
    \chardef\bbl@opt@hyphenmap\z@
  \else
    \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
      \csname\languagename @bbl@hyphenmap\endcsname
    \fi
  \fi
  \global\let\bbl@hymapsel\@cclv
  \bbl@patterns{#1}%
  \babel@savevariable\lefthyphenmin
  \babel@savevariable\righthyphenmin
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
    \set@hyphenmins\tw@\thr@@\relax
  \else
    \expandafter\expandafter\expandafter\set@hyphenmins
      \csname #1hyphenmins\endcsname\relax
  \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.

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

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

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

```
\expandafter\def\csname otherlanguage*\endcsname#1{%
  \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
  \foreign@language{#1}}
```

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

```
\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 behaviour 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.

```
\let\bbl@beforeforeign\@empty
\edef\foreignlanguage{%
  \noexpand\protect
  \expandafter\noexpand\csname foreignlanguage \endcsname}
\expandafter\def\csname foreignlanguage \endcsname{%
  \@ifstar\bbl@foreign@s\bbl@foreign@x}
\def\bbl@foreign@x#1#2{%
  \begingroup
    \let\BabelText\@firstofone
    \bbl@beforeforeign
    \foreign@language{#1}%
    \bbl@usehooks{foreign}{}%
    \BabelText{#2}% Now in horizontal mode!
  \endgroup}
\def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
  \begingroup
    {\par}%
    \let\BabelText\@firstofone
    \foreign@language{#1}%
    \bbl@usehooks{foreign*}{}%
    \bbl@dirparastext
    \BabelText{#2}% Still in vertical mode!
    {\par}%
  \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`.

```
\def\foreign@language#1{%
  \edef\languagename{#1}%
  \bbl@fixname\languagename
  \bbl@iflanguage\languagename{%
    \expandafter\ifx\csname date\languagename\endcsname\relax
      \bbl@warning
        {Unknown language `#1'. Either you have\\%
         misspelled its name, it has not been installed,\\%
         or you requested it in a previous run. Fix its name,\\%
         install it or just rerun the file, respectively.\\%
         I'll proceed, but expect wrong results.\\%
         Reported}%
    \fi
    \let\bbl@select@type\@ne
    \expandafter\bbl@switch\expandafter{\languagename}}}
```

`\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.

```
\let\bbl@hyphlist\@empty
\let\bbl@hyphenation@\relax
\let\bbl@pttnlist\@empty
\let\bbl@patterns@\relax
\let\bbl@hymapsel=\@cclv
\def\bbl@patterns#1{%
  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
      \csname l@#1\endcsname
      \edef\bbl@tempa{#1}%
    \else
      \csname l@#1:\f@encoding\endcsname
      \edef\bbl@tempa{#1:\f@encoding}%
    \fi
  \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
  \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
    \begingroup
      \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
      \ifin@\else
        \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
        \hyphenation{%
          \bbl@hyphenation@
          \@ifundefined{bbl@hyphenation@#1}%
            \@empty
            {\space\csname bbl@hyphenation@#1\endcsname}}%
        \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
      \fi
    \endgroup}}
```

hyphenrules The environment hyphenrules can be used to select *just* the hyphenation rules. This environment does *not* change \languagename and when the hyphenation rules specified were not loaded it has no effect. Note however, \lccode’s and font encodings are not set at all, so in most cases you should use otherlanguage*.

```
\def\hyphenrules#1{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf
  \bbl@iflanguage\bbl@tempf{%
    \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
    \languageshorthands{none}%
    \bbl@ifunset{\bbl@tempf hyphenmins}%
      {\set@hyphenmins\tw@\thr@@\relax}%
      {\bbl@exp{\\\set@hyphenmins\@nameuse{\bbl@tempf hyphenmins}}}}}
\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.

```
\def\providehyphenmins#1#2{%
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
    \@namedef{#1hyphenmins}{#2}%
  \fi}
```

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

```
\def\set@hyphenmins#1#2{%
  \lefthyphenmin#1\relax
  \righthyphenmin#2\relax}
```

\ProvidesLanguage The identification code for each file is something that was introduced in LaTeX 2ε. 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.

```
\ifx\ProvidesFile\@undefined
  \def\ProvidesLanguage#1[#2 #3 #4]{%
    \wlog{Language: #1 #4 #3 <#2>}%
    }
\else
  \def\ProvidesLanguage#1{%
    \begingroup
      \catcode`\ 10 %
      \@makeother\/%
      \@ifnextchar[%]
        {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
  \def\@provideslanguage#1[#2]{%
    \wlog{Language: #1 #2}%
    \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
    \endgroup}
\fi
```

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

```
\def\LdfInit{%
  \chardef\atcatcode=\catcode`\@
  \catcode`\@=11\relax
  \input babel.def\relax
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \LdfInit}
```

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

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

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

```
\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':

```
\providecommand\setlocale{%
  \bbl@error
    {Not yet available}%
    {Find an armchair, sit down and wait}}
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale
```

10.2 Errors

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

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

```
\edef\bbl@nulllanguage{\string\language=0}
```

```
\ifx\PackageError\@undefined
  \def\bbl@error#1#2{%
    \begingroup
      \newlinechar=`\^^J
      \def\\{^^J(babel) }%
      \errhelp{#2}\errmessage{\\#1}%
    \endgroup}
  \def\bbl@warning#1{%
    \begingroup
      \newlinechar=`\^^J
      \def\\{^^J(babel) }%
      \message{\\#1}%
    \endgroup}
  \def\bbl@info#1{%
    \begingroup
      \newlinechar=`\^^J
      \def\\{^^J}%
      \wlog{#1}%
    \endgroup}
\else
  \def\bbl@error#1#2{%
    \begingroup
      \def\\{\MessageBreak}%
      \PackageError{babel}{#1}{#2}%
    \endgroup}
  \def\bbl@warning#1{%
    \begingroup
      \def\\{\MessageBreak}%
      \PackageWarning{babel}{#1}%
    \endgroup}
  \def\bbl@info#1{%
    \begingroup
      \def\\{\MessageBreak}%
      \PackageInfo{babel}{#1}%
    \endgroup}
\fi
\@ifpackagewith{babel}{silent}
  {\let\bbl@info\@gobble
   \let\bbl@warning\@gobble}
  {}
\def\bbl@nocaption#1#2{% 1: text to be printed 2: caption macro \langXname
  \gdef#2{\textbf{?#1?}}%
  #2%
  \bbl@warning{%
    \string#2 not set. Please, define\\%
    it in the preamble with something like:\\%
    \string\renewcommand\string#2{..}\\%
    Reported}}
\def\@nolanerr#1{%
  \bbl@error
    {You haven't defined the language #1\space yet}%
    {Your command will be ignored, type <return> to proceed}}
\def\@nopatterns#1{%
  \bbl@warning
    {No hyphenation patterns were preloaded for\\%
     the language `#1' into the format.\\%
     Please, configure your TeX system to add them and\\%
     rebuild the format. Now I will use the patterns\\%
     preloaded for \bbl@nulllanguage\space instead}}
```

```
\let\bbl@usehooks\@gobbletwo
⟨/kernel⟩
```

11 Loading hyphenation patterns

The following code is meant to be read by iniTeX because it should instruct 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.
`toks8` stores info to be shown when the program is run.
We want to add a message to the message LaTeX 2.09 puts in the `\everyjob` register. This could be done by the following code:

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

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before LaTeX fills the register.
There are some problems with this approach though.

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

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, `\dump`. Therefore the original `\dump` is saved in `\org@dump` and a new definition is supplied.
To make sure that LaTeX 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.
This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.
Then everything is restored to the old situation and the format is dumped.

```
⟨∗patterns⟩
⟨⟨Make sure ProvidesFile is defined⟩⟩
\ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
\xdef\bbl@format{\jobname}
\ifx\AtBeginDocument\@undefined
  \def\@empty{}
  \let\orig@dump\dump
  \def\dump{%
    \ifx\@ztryfc\@undefined
    \else
      \toks0=\expandafter{\@preamblecmds}%
      \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
      \def\@begindocumenthook{}%
    \fi
    \let\dump\orig@dump\let\orig@dump\@undefined\dump}
```

```
\fi
⟨⟨Define core switching macros⟩⟩
\toks8{Babel <<@version@>> and hyphenation patterns for }%
```

`\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.

```
\def\process@line#1#2 #3 #4 {%
  \ifx=#1%
    \process@synonym{#2}%
  \else
    \process@language{#1#2}{#3}{#4}%
  \fi
  \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.

```
\toks@{}
\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.

```
\def\process@synonym#1{%
  \ifnum\last@language=\m@ne
    \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
  \else
    \expandafter\chardef\csname l@#1\endcsname\last@language
    \wlog{\string\l@#1=\string\language\the\last@language}%
    \expandafter\let\csname #1hyphenmins\expandafter\endcsname
      \csname\languagename hyphenmins\endcsname
    \let\bbl@elt\relax
    \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
  \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 'name' of the language that will be loaded now is added to the token register `\toks8`. and finally 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 behaviour depending on the given encoding (it is set to empty if no encoding is given).
Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. TEX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\`⟨*lang*⟩`hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the \lccode en \uccode arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the \patterns command acts globally so its effect will be remembered.
Then we globally store the settings of \lefthyphenmin and \righthyphenmin and close the group.
When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)
\bbl@languages saves a snapshot of the loaded languagues 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.

```
\def\process@language#1#2#3{%
  \expandafter\addlanguage\csname l@#1\endcsname
  \expandafter\language\csname l@#1\endcsname
  \edef\languagename{#1}%
  \bbl@hook@everylanguage{#1}%
  \bbl@get@enc#1::\@@@
  \begingroup
      \lefthyphenmin\m@ne
      \bbl@hook@loadpatterns{#2}%
      \ifnum\lefthyphenmin=\m@ne
      \else
        \expandafter\xdef\csname #1hyphenmins\endcsname{%
          \the\lefthyphenmin\the\righthyphenmin}%
    \fi
  \endgroup
  \def\bbl@tempa{#3}%
  \ifx\bbl@tempa\@empty\else
    \bbl@hook@loadexceptions{#3}%
  \fi
  \let\bbl@elt\relax
  \edef\bbl@languages{%
      \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
  \ifnum\the\language=\z@
    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
      \set@hyphenmins\tw@\thr@@\relax
    \else
      \expandafter\expandafter\expandafter\set@hyphenmins
        \csname #1hyphenmins\endcsname
    \fi
    \the\toks@
    \toks@{}%
  \fi}
```

\bbl@get@enc
\bbl@hyph@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.

```
\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.

```
\def\bbl@hook@everylanguage#1{}
\def\bbl@hook@loadpatterns#1{\input #1\relax}
\let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
```

```
\let\bbl@hook@loadkernel\bbl@hook@loadpatterns
\begingroup
  \def\AddBabelHook#1#2{%
    \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
      \def\next{\toks1}%
    \else
      \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
    \fi
    \next}
  \ifx\directlua\@undefined
    \ifx\XeTeXinputencoding\@undefined\else
      \input xebabel.def
    \fi
  \else
    \input luababel.def
  \fi
  \openin1 = babel-\bbl@format.cfg
  \ifeof1
  \else
    \input babel-\bbl@format.cfg\relax
  \fi
  \closein1
\endgroup
\bbl@hook@loadkernel{switch.def}
```

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

```
\openin1 = language.dat
```

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

```
\def\languagename{english}%
\ifeof1
  \message{I couldn't find the file language.dat,\space
           I will try the file hyphen.tex}
  \input hyphen.tex\relax
  \chardef\l@english\z@
\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.

```
  \last@language\m@ne
```

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

```
  \loop
```

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

```
    \endlinechar\m@ne
    \read1 to \bbl@line
    \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.

```
    \if T\ifeof1F\fi T\relax
      \ifx\bbl@line\@empty\else
```

```
        \edef\bbl@line{\bbl@line\space\space\space}%
        \expandafter\process@line\bbl@line\relax
      \fi
  \repeat
```

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

```
  \begingroup
    \def\bbl@elt#1#2#3#4{%
      \global\language=#2\relax
      \gdef\languagename{#1}%
      \def\bbl@elt##1##2##3##4{}}%
    \bbl@languages
  \endgroup
\fi
```

and close the configuration file.

```
\closein1
```

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

```
\if/\the\toks@/\else
  \errhelp{language.dat loads no language, only synonyms}
  \errmessage{Orphan language synonym}
\fi
\advance\last@language\@ne
\edef\bbl@tempa{%
  \everyjob{%
    \the\everyjob
    \ifx\typeout\@undefined
      \immediate\write16%
    \else
      \noexpand\typeout
    \fi
    {\the\toks8 \the\last@language\space language(s) loaded.}}}
\advance\last@language\m@ne
\bbl@tempa
```

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.

```
\let\bbl@line\@undefined
\let\process@line\@undefined
\let\process@synonym\@undefined
\let\process@language\@undefined
\let\bbl@get@enc\@undefined
\let\bbl@hyph@enc\@undefined
\let\bbl@tempa\@undefined
\let\bbl@hook@loadkernel\@undefined
\let\bbl@hook@everylanguage\@undefined
\let\bbl@hook@loadpatterns\@undefined
\let\bbl@hook@loadexceptions\@undefined
⟨/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].

```
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{bidi=basic-r}%
  {\let\bbl@beforeforeign\leavevmode
   \newattribute\bbl@attr@dir
   \bbl@exp{\output{\bodydir\pagedir\the\output}}%
   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
\DeclareOption{bidi=default}%
  {\let\bbl@beforeforeign\leavevmode
   \ifodd\bbl@engine
     \newattribute\bbl@attr@dir
     \bbl@exp{\output{\bodydir\pagedir\the\output}}%
   \fi
   \AtEndOfPackage{%
     \EnableBabelHook{babel-bidi}%
     \ifodd\bbl@engine\else
       \bbl@xebidipar
     \fi}}
⟨⟨/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.

```
⟨⟨*Font selection⟩⟩ ≡
\@onlypreamble\babelfont
\newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
  \edef\bbl@tempa{#1}%
  \def\bbl@tempb{#2}%
  \ifx\fontspec\@undefined
    \usepackage{fontspec}%
  \fi
  \EnableBabelHook{babel-fontspec}%
  \bbl@bblfont}
\newcommand\bbl@bblfont[2][]{% 1=features 2=fontname
  \bbl@ifunset{\bbl@tempb family}{\bbl@providefam{\bbl@tempb}}{}%
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
  \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
    {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
     \bbl@exp{%
       \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
       \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
    {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
       \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:

```
\def\bbl@providefam#1{%
  \bbl@exp{%
    \\\newcommand\<#1default>{}% Just define it
    \\\bbl@add@list\\\bbl@font@fams{#1}%
    \\\DeclareRobustCommand\<#1family>{%
      \\\not@math@alphabet\<#1family>\relax
      \\\fontfamily\<#1default>\\\selectfont}%
    \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}}
```

The following macro is activated when the hook `babel-fontspec` is enabled.

```
\def\bbl@switchfont{%
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
  \bbl@exp{%  eg Arabic -> arabic
    \lowercase{\edef\\\bbl@tempa{\bbl@cs{sname@\languagename}}}}%
  \bbl@foreach\bbl@font@fams{%
    \bbl@ifunset{bbl@##1dflt@\languagename}%    (1) language?
      {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%   (2) from script?
         {\bbl@ifunset{bbl@##1dflt@}%           2=F - (3) from generic?
           {}%                                  123=F - nothing!
           {\bbl@exp{%                          3=T - from generic
              \global\let\<bbl@##1dflt@\languagename>%
                         \<bbl@##1dflt@>}}}%
         {\bbl@exp{%                            2=T - from script
            \global\let\<bbl@##1dflt@\languagename>%
                       \<bbl@##1dflt@*\bbl@tempa>}}}%
      {}}%                               1=T - language, already defined
  \def\bbl@tempa{%
    \bbl@warning{The current font is not a standard family.\\%
      Script and Language are not applied. Consider defining\\%
      a new family with \string\babelfont,}}%
  \bbl@foreach\bbl@font@fams{%     don't gather with prev for
    \bbl@ifunset{bbl@##1dflt@\languagename}%
      {\bbl@cs{famrst@##1}%
       \global\bbl@csarg\let{famrst@##1}\relax}%
      {\bbl@exp{% order is relevant
         \\\bbl@add\\\originalTeX{%
           \\\bbl@font@rst{\bbl@cs{##1dflt@\languagename}}%
                          \<##1default>\<##1family>{##1}}%
         \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
                        \<##1default>\<##1family>}}}%
  \bbl@ifrestoring{}{\bbl@tempa}}%
```

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.

```
\def\bbl@font@set#1#2#3{%
  \bbl@xin@{<>}{#1}%
  \ifin@
    \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1}%
  \fi
  \bbl@exp{%
    \def\\#2{#1}%        eg, \rmdefault{\bbl@rm1dflt@lang}
    \\\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\\bbl@tempa\relax}{}}}
\def\bbl@fontspec@set#1#2#3{%
  \bbl@exp{\<fontspec_set_family:Nnn>\\#1%
    {\bbl@cs{lsys@\languagename},#2}}{#3}%
  \bbl@toglobal#1}%
```

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.

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

```
\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 :-).

```
\newcommand\babelFSstore[2][]{%
  \bbl@ifblank{#1}%
    {\bbl@csarg\def{sname@#2}{Latin}}%
    {\bbl@csarg\def{sname@#2}{#1}}%
  \bbl@provide@dirs{#2}%
  \bbl@csarg\ifnum{wdir@#2}>\z@
    \let\bbl@beforeforeign\leavevmode
    \EnableBabelHook{babel-bidi}%
  \fi
  \bbl@foreach{#2}{%
    \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
    \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
    \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
\def\bbl@FSstore#1#2#3#4{%
  \bbl@csarg\edef{#2default#1}{#3}%
  \expandafter\addto\csname extras#1\endcsname{%
    \let#4#3%
    \ifx#3\f@family
      \edef#3{\csname bbl@#2default#1\endcsname}%
      \fontfamily{#3}\selectfont
    \else
      \edef#3{\csname bbl@#2default#1\endcsname}%
    \fi}%
  \expandafter\addto\csname noextras#1\endcsname{%
    \ifx#3\f@family
      \fontfamily{#4}\selectfont
    \fi
    \let#3#4}}
\let\bbl@langfeatures\@empty
\def\babelFSfeatures{% make sure \fontspec is redefined once
  \let\bbl@ori@fontspec\fontspec
  \renewcommand\fontspec[1][]{%
    \bbl@ori@fontspec[\bbl@langfeatures##1]}
  \let\babelFSfeatures\bbl@FSfeatures
  \babelFSfeatures}
\def\bbl@FSfeatures#1#2{%
  \expandafter\addto\csname extras#1\endcsname{%
    \babel@save\bbl@langfeatures
    \edef\bbl@langfeatures{#2,}}}
⟨⟨/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.
LaTeX sets many "codes" just before loading hyphen.cfg. That is not a problem in luatex, but in xetex they must be reset to the proper value. Most of the work is done in xe(la)tex.ini, so here we just "undo" some of the changes done by LaTeX. Anyway, for consistency LuaTeX also resets the catcodes.

```
⟨⟨*Restore Unicode catcodes before loading patterns⟩⟩ ≡
  \begingroup
```

```
      % Reset chars "80-"C0 to category "other", no case mapping:
    \catcode`\@=11 \count@=128
    \loop\ifnum\count@<192
      \global\uccode\count@=0 \global\lccode\count@=0
      \global\catcode\count@=12 \global\sfcode\count@=1000
      \advance\count@ by 1 \repeat
      % Other:
    \def\O ##1 {%
      \global\uccode"##1=0 \global\lccode"##1=0
      \global\catcode"##1=12 \global\sfcode"##1=1000 }%
      % Letter:
    \def\L ##1 ##2 ##3 {\global\catcode"##1=11
      \global\uccode"##1="##2
      \global\lccode"##1="##3
      % Uppercase letters have sfcode=999:
      \ifnum"##1="##3 \else \global\sfcode"##1=999 \fi }%
      % Letter without case mappings:
    \def\l ##1 {\L ##1 ##1 ##1 }%
    \l 00AA
    \L 00B5 039C 00B5
    \l 00BA
    \O 00D7
    \l 00DF
    \O 00F7
    \L 00FF 0178 00FF
  \endgroup
  \input #1\relax
⟨⟨/Restore Unicode catcodes before loading patterns⟩⟩
```

Now, the code.

```
⟨*xetex⟩
\def\BabelStringsDefault{unicode}
\let\xebbl@stop\relax
\AddBabelHook{xetex}{encodedcommands}{%
  \def\bbl@tempa{#1}%
  \ifx\bbl@tempa\@empty
    \XeTeXinputencoding"bytes"%
  \else
    \XeTeXinputencoding"#1"%
  \fi
  \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
\AddBabelHook{xetex}{stopcommands}{%
  \xebbl@stop
  \let\xebbl@stop\relax}
\AddBabelHook{xetex}{loadkernel}{%
⟨⟨Restore Unicode catcodes before loading patterns⟩⟩}
\ifx\DisableBabelHook\@undefined\endinput\fi
\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\DisableBabelHook{babel-fontspec}
⟨⟨Font selection⟩⟩
\input txtbabel.def
⟨/xetex⟩
```

13.2 Layout

In progress.

Unfortunately, for proper support for xetex lots of macros and packages must be patched somehow. At least at this stage, babel will not do it and therefore a package similar to bidi will be required. Any help in making babel and bidi collaborate will be wlecome. Note as well, elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

`\bbl@startskip` and `\bbl@endskip` are available to package authors. Thanks to the TeX expansion mechanims the following constructs are valid: `\adim\bbl@startskip`, `\advance\bbl@startskip\adim`, `\bbl@startskip\adim`.

Consider `txtbabel` as a shorthand for *tex–xet babel*.

```
⟨*texxet⟩
\ifx\bbl@opt@layout\@nil\endinput\fi  % No layout
\def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
\def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
\def\@hangfrom#1{%
  \setbox\@tempboxa\hbox{{#1}}%
  \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
  \noindent\box\@tempboxa}
\def\raggedright{%
  \let\\\@centercr
  \bbl@startskip\z@skip
  \@rightskip\@flushglue
  \bbl@endskip\@rightskip
  \parindent\z@
  \parfillskip\bbl@startskip}
\def\raggedleft{%
  \let\\\@centercr
  \bbl@startskip\@flushglue
  \bbl@endskip\z@skip
  \parindent\z@
  \parfillskip\bbl@endskip}
\IfBabelLayout{lists}
  {\def\list#1#2{%
    \ifnum \@listdepth >5\relax
      \@toodeep
    \else
      \global\advance\@listdepth\@ne
    \fi
    \rightmargin\z@
    \listparindent\z@
    \itemindent\z@
    \csname @list\romannumeral\the\@listdepth\endcsname
    \def\@itemlabel{#1}%
    \let\makelabel\@mklab
    \@nmbrlistfalse
    #2\relax
    \@trivlist
    \parskip\parsep
    \parindent\listparindent
    \advance\linewidth-\rightmargin
    \advance\linewidth-\leftmargin
    \advance\@totalleftmargin
      \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi
    \parshape\@ne\@totalleftmargin\linewidth
    \ignorespaces}%
  \def\labelenumii{)\theenumii(}%
  \def\p@enumiii{\p@enumii)\theenumii(}}
  {}
\IfBabelLayout{contents}
  {\def\@dottedtocline#1#2#3#4#5{%
     \ifnum#1>\c@tocdepth\else
```

```
       \vskip \z@ \@plus.2\p@
       {\bbl@startskip#2\relax
        \bbl@endskip\@tocrmarg
        \parfillskip-\bbl@endskip
        \parindent#2\relax
        \@afterindenttrue
        \interlinepenalty\@M
        \leavevmode
        \@tempdima#3\relax
        \advance\bbl@startskip\@tempdima
        \null\nobreak\hskip-\bbl@startskip
        {#4}\nobreak
        \leaders\hbox{%
          $\m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
          \hfill\nobreak
          \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
          \par}%
     \fi}%
   \def\babel@toc#1{%
     \select@language{#1}%
     \bbl@ifunset{bbl@wdir@\bbl@main@language}%
       {\bbl@provide@dirs{\bbl@main@language}}%
       {}%
     \bbl@exp{%
       \\\bbl@pardir\bbl@cs{wdir@\bbl@main@language}%
       \\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
  {}
\IfBabelLayout{columns}
  {\def\@outputdblcol{%
     \if@firstcolumn
       \global\@firstcolumnfalse
       \global\setbox\@leftcolumn\copy\@outputbox
       \splitmaxdepth\maxdimen
       \vbadness\maxdimen
       \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
       \setbox\@outputbox\vsplit\@outputbox to\maxdimen
       \toks@\expandafter{\topmark}%
       \xdef\@firstcoltopmark{\the\toks@}%
       \toks@\expandafter{\splitfirstmark}%
       \xdef\@firstcolfirstmark{\the\toks@}%
       \ifx\@firstcolfirstmark\@empty
         \global\let\@setmarks\relax
       \else
         \gdef\@setmarks{%
           \let\firstmark\@firstcolfirstmark
           \let\topmark\@firstcoltopmark}%
       \fi
     \else
       \global\@firstcolumntrue
       \setbox\@outputbox\vbox{%
         \hb@xt@\textwidth{%
           \hskip\columnwidth
           \hfil
           {\normalcolor\vrule \@width\columnseprule}%
           \hfil
           \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
           \hskip-\textwidth
           \hb@xt@\columnwidth{\box\@outputbox \hss}%
           \hskip\columnsep
           \hskip\columnwidth}}%
       \@combinedblfloats
       \@setmarks
       \@outputpage
       \begingroup
         \@dblfloatplacement
         \@startdblcolumn
         \@whilesw\if@fcolmade \fi{\@outputpage
         \@startdblcolumn}%
       \endgroup
     \fi}}%
  {}
⟨/texxet⟩
```

13.3 LuaTeX

The new loader for luatex is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the hyphenmins stuff, which is under the direct control of babel).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, the are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they has been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). For the moment, a dangerous approach is used – just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

```
⟨*luatex⟩
\ifx\AddBabelHook\@undefined
\begingroup
  \toks@{}
  \count@\z@ % 0=start, 1=0th, 2=normal
  \def\bbl@process@line#1#2 #3 #4 {%
    \ifx=#1%
      \bbl@process@synonym{#2}%
    \else
      \bbl@process@language{#1#2}{#3}{#4}%
    \fi
    \ignorespaces}
  \def\bbl@manylang{%
    \ifnum\bbl@last>\@ne
      \bbl@info{Non-standard hyphenation setup}%
    \fi
    \let\bbl@manylang\relax}
  \def\bbl@process@language#1#2#3{%
    \ifcase\count@
      \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
    \or
      \count@\tw@
    \fi
    \ifnum\count@=\tw@
      \expandafter\addlanguage\csname l@#1\endcsname
      \language\allocationnumber
      \chardef\bbl@last\allocationnumber
      \bbl@manylang
      \let\bbl@elt\relax
      \xdef\bbl@languages{%
        \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
    \fi
    \the\toks@
    \toks@{}}
  \def\bbl@process@synonym@aux#1#2{%
    \global\expandafter\chardef\csname l@#1\endcsname#2\relax
    \let\bbl@elt\relax
    \xdef\bbl@languages{%
      \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
  \def\bbl@process@synonym#1{%
    \ifcase\count@
      \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
    \or
      \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
    \else
      \bbl@process@synonym@aux{#1}{\the\bbl@last}%
    \fi}
  \ifx\bbl@languages\@undefined % Just a (sensible?) guess
    \chardef\l@english\z@
    \chardef\l@USenglish\z@
    \chardef\bbl@last\z@
    \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
    \gdef\bbl@languages{%
      \bbl@elt{english}{0}{hyphen.tex}{}%
      \bbl@elt{USenglish}{0}{}{}}
  \else
    \global\let\bbl@languages@format\bbl@languages
    \def\bbl@elt#1#2#3#4{% Remove all except language 0
      \ifnum#2>\z@\else
        \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
      \fi}%
    \xdef\bbl@languages{\bbl@languages}%
  \fi
  \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
  \bbl@languages
  \openin1=language.dat
  \ifeof1
    \bbl@warning{I couldn't find language.dat. No additional\\%
```

```
                 patterns loaded. Reported}%
  \else
    \loop
      \endlinechar\m@ne
      \read1 to \bbl@line
      \endlinechar`\^^M
      \if T\ifeof1F\fi T\relax
        \ifx\bbl@line\@empty\else
          \edef\bbl@line{\bbl@line\space\space\space}%
          \expandafter\bbl@process@line\bbl@line\relax
        \fi
    \repeat
  \fi
\endgroup
\def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
\ifx\babelcatcodetablenum\@undefined
  \def\babelcatcodetablenum{5211}
\fi
\def\bbl@luapatterns#1#2{%
  \bbl@get@enc#1::\@@@
  \setbox\z@\hbox\bgroup
    \begingroup
      \ifx\catcodetable\@undefined
        \let\savecatcodetable\luatexsavecatcodetable
        \let\initcatcodetable\luatexinitcatcodetable
        \let\catcodetable\luatexcatcodetable
      \fi
      \savecatcodetable\babelcatcodetablenum\relax
      \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
      \catcodetable\numexpr\babelcatcodetablenum+1\relax
      \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
      \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
      \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
      \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
      \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
      \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
      \input #1\relax
      \catcodetable\babelcatcodetablenum\relax
    \endgroup
    \def\bbl@tempa{#2}%
    \ifx\bbl@tempa\@empty\else
      \input #2\relax
    \fi
  \egroup}%
\def\bbl@patterns@lua#1{%
  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
    \csname l@#1\endcsname
    \edef\bbl@tempa{#1}%
  \else
    \csname l@#1:\f@encoding\endcsname
    \edef\bbl@tempa{#1:\f@encoding}%
  \fi\relax
  \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
  \@ifundefined{bbl@hyphendata@\the\language}%
    {\def\bbl@elt##1##2##3##4{%
       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
         \def\bbl@tempb{##3}%
         \ifx\bbl@tempb\@empty\else % if not a synonymous
           \def\bbl@tempc{{##3}{##4}}%
         \fi
         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
       \fi}%
     \bbl@languages
     \@ifundefined{bbl@hyphendata@\the\language}%
       {\bbl@info{No hyphenation patterns were set for\\%
                  language ‘\bbl@tempa’. Reported}}%
       {\expandafter\expandafter\expandafter\bbl@luapatterns
          \csname bbl@hyphendata@\the\language\endcsname}{}}
\endinput\fi
\begingroup
\catcode`\%=12
\catcode`\'=12
\catcode`\"=12
\catcode`\:=12
\directlua{
  Babel = Babel or {}
  function Babel.bytes(line)
    return line:gsub("(.)",
      function (chr) return unicode.utf8.char(string.byte(chr)) end)
  end
  function Babel.begin_process_input()
    if luatexbase and luatexbase.add_to_callback then
      luatexbase.add_to_callback('process_input_buffer',
                                 Babel.bytes,'Babel.bytes')
    else
      Babel.callback = callback.find('process_input_buffer')
      callback.register('process_input_buffer',Babel.bytes)
    end
  end
  function Babel.end_process_input ()
    if luatexbase and luatexbase.remove_from_callback then
      luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
    else
      callback.register('process_input_buffer',Babel.callback)
    end
  end
  function Babel.addpatterns(pp, lg)
    local lg = lang.new(lg)
    local pats = lang.patterns(lg) or ''
    lang.clear_patterns(lg)
    for p in pp:gmatch('[^%s]+') do
      ss = ''
      for i in string.utfcharacters(p:gsub('%d', '')) do
         ss = ss .. '%d?' .. i
      end
      ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
      ss = ss:gsub('%.%%d%?$', '%%.')
      pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
      if n == 0 then
        tex.sprint(
          [[\string\csname\space bbl@info\endcsname{New pattern: ]]
          .. p .. [[}]])
        pats = pats .. ' ' .. p
      else
        tex.sprint(
          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
          .. p .. [[}]])
      end
    end
    lang.patterns(lg, pats)
  end
}
\endgroup
\def\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook{luatex}{encodedcommands}{%
  \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
  \ifx\bbl@tempa\bbl@tempb\else
    \directlua{Babel.begin_process_input()}%
    \def\luabbl@stop{%
      \directlua{Babel.end_process_input()}}%
  \fi}%
\AddBabelHook{luatex}{stopcommands}{%
  \luabbl@stop
  \let\luabbl@stop\relax}
\AddBabelHook{luatex}{patterns}{%
  \@ifundefined{bbl@hyphendata@\the\language}%
    {\def\bbl@elt##1##2##3##4{%
       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
         \def\bbl@tempb{##3}%
         \ifx\bbl@tempb\@empty\else % if not a synonymous
           \def\bbl@tempc{{##3}{##4}}%
         \fi
         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
       \fi}%
     \bbl@languages
     \@ifundefined{bbl@hyphendata@\the\language}%
       {\bbl@info{No hyphenation patterns were set for\\%
                  language ‘#2’. Reported}}%
       {\expandafter\expandafter\expandafter\bbl@luapatterns
          \csname bbl@hyphendata@\the\language\endcsname}}{}%
  \@ifundefined{bbl@patterns@}{}{%
    \begingroup
      \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
      \ifin@\else
        \ifx\bbl@patterns@\@empty\else
           \directlua{ Babel.addpatterns(
             [[\bbl@patterns@]], \number\language) }%
        \fi
        \@ifundefined{bbl@patterns@#1}%
          \@empty
          {\directlua{ Babel.addpatterns(
               [[\space\csname bbl@patterns@#1\endcsname]],
               \number\language) }}%
        \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
      \fi
    \endgroup}}
\AddBabelHook{luatex}{everylanguage}{%
  \def\process@language##1##2##3{%
    \def\process@line####1####2 ####3 ####4 {}}}
\AddBabelHook{luatex}{loadpatterns}{%
   \input #1\relax
   \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
     {{#1}{}}}
\AddBabelHook{luatex}{loadexceptions}{%
   \input #1\relax
   \def\bbl@tempb##1##2{{##1}{#1}}%
   \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
     {\expandafter\expandafter\expandafter\bbl@tempb
      \csname bbl@hyphendata@\the\language\endcsname}}
```

\babelpatterns This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```
\@onlypreamble\babelpatterns
\AtEndOfPackage{%
  \newcommand\babelpatterns[2][\@empty]{%
    \ifx\bbl@patterns@\relax
      \let\bbl@patterns@\@empty
    \fi
    \ifx\bbl@pttnlist\@empty\else
      \bbl@warning{%
        You must not intermingle \string\selectlanguage\space and\\%
        \string\babelpatterns\space or some patterns will not\\%
        be taken into account. Reported}%
    \fi
    \ifx\@empty#1%
      \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
    \else
      \edef\bbl@tempb{\zap@space#1 \@empty}%
      \bbl@for\bbl@tempa\bbl@tempb{%
        \bbl@fixname\bbl@tempa
        \bbl@iflanguage\bbl@tempa{%
          \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
            \@ifundefined{bbl@patterns@\bbl@tempa}%
              \@empty
              {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
            #2}}}%
    \fi}}
```

Common stuff.

```
\AddBabelHook{luatex}{loadkernel}{%
⟨⟨Restore Unicode catcodes before loading patterns⟩⟩}
\ifx\DisableBabelHook\@undefined\endinput\fi
\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\DisableBabelHook{babel-fontspec}
⟨⟨Font selection⟩⟩
```

14 Bidi support in luatex

14.1 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.) and with `bidi=basic-r`, 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`.

```
\ifx\bbl@opt@layout\@nil\endinput\fi  % No layout
\def\@hangfrom#1{%
```

```
  \setbox\@tempboxa\hbox{{#1}}%
  \hangindent\wd\@tempboxa
  \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
    \shapemode\@ne
  \fi
  \noindent\box\@tempboxa}
\IfBabelLayout{lists}
  {\def\list#1#2{%
     \ifnum \@listdepth >5\relax
       \@toodeep
     \else
       \global\advance\@listdepth\@ne
     \fi
     \rightmargin\z@
     \listparindent\z@
     \itemindent\z@
     \csname @list\romannumeral\the\@listdepth\endcsname
     \def\@itemlabel{#1}%
     \let\makelabel\@mklab
     \@nmbrlistfalse
     #2\relax
     \@trivlist
     \parskip\parsep
     \parindent\listparindent
     \advance\linewidth -\rightmargin
     \advance\linewidth -\leftmargin
     \advance\@totalleftmargin \leftmargin
     \parshape \@ne
     \@totalleftmargin \linewidth
     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
       \shapemode\tw@
     \fi
     \ignorespaces}}
  {}
\IfBabelLayout{contents}
  {\def\babel@toc#1{%
     \select@language{#1}%
     \bbl@ifunset{bbl@wdir@\bbl@main@language}%
       {\bbl@provide@dirs{\bbl@main@language}}%
       {}%
     \bbl@exp{%
       \\\bbl@pardir\bbl@cs{wdir@\bbl@main@language}%
       \\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
  {}
% only if bidi=default %%%%%%%%%%%%%%% :
% \def\labelenumii{)\theenumii(}  % luas, pdf, no xe, luar
% \def\p@enumiii{\p@enumii)\theenumii(} % luas, pdf, no xe, luar
⟨/luatex⟩
```

14.2 Auto bidi with `basic-r`

The file babel-bidi.lua currently only contains data. It's a large and boring file and it's not shown here. See the generated file.
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.

```
⟨*basic-r⟩
Babel = Babel or {}

require('babel-bidi.lua')

local characters = Babel.characters
local ranges = Babel.ranges

local DIR = node.id("dir")

local function dir_mark(head, from, to, outer)
  dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
  local d = node.new(DIR)
  d.dir = '+' .. dir
  node.insert_before(head, from, d)
  d = node.new(DIR)
  d.dir = '-' .. dir
  node.insert_after(head, to, d)
end

function Babel.pre_otfload(head)
  local first_n, last_n           -- first and last char with nums
  local last_es                   -- an auxiliary 'last' used with nums
  local first_d, last_d           -- first and last char in L/R block
  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):

```
  local strong = ('TRT' == tex.pardir) and 'r' or 'l'
  local strong_lr = (strong == 'l') and 'l' or 'r'
  local outer = strong

  local new_dir = false
  local first_dir = false

```

```
local last_lr

local type_n = ''

for item in node.traverse(head) do

  -- three cases: glyph, dir, otherwise
  if item.id == node.id'glyph' then

    local chardata = characters[item.char]
    dir = chardata and chardata.d or nil
    if not dir then
      for nn, et in ipairs(ranges) do
        if item.char < et[1] then
          break
        elseif item.char <= et[2] then
          dir = et[3]
          break
        end
      end
    end
    dir = dir or 'l'
```

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.

```
    if new_dir then
      attr_dir = 0
      for at in node.traverse(item.attr) do
        if at.number == luatexbase.registernumber'bbl@attr@dir' then
          attr_dir = at.value
        end
      end
      texio.write_nl(attr_dir)
      if attr_dir == 1 then
        strong = 'r'
      elseif attr_dir == 2 then
        strong = 'al'
      else
        strong = 'l'
      end
      strong_lr = (strong == 'l') and 'l' or 'r'
      outer = strong_lr
      new_dir = false
    end
    if dir == 'nsm' then dir = strong end             -- W1
```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```
    dir_real = dir               -- We need dir_real to set strong below
    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:

```
    if strong == 'al' then
      if dir == 'en' then dir = 'an' end                -- W2
      if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
      strong_lr = 'r'                                   -- W3
    end
```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```
    elseif item.id == node.id'dir' then
      new_dir = true
      dir = nil
    else
      dir = nil          -- Not a char
    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 behaviour could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```
    if dir == 'en' or dir == 'an' or dir == 'et' then
      if dir ~= 'et' then
        type_n = dir
      end
      first_n = first_n or item
      last_n = last_es or item
      last_es = nil
    elseif dir == 'es' and last_n then -- W3+W6
      last_es = item
    elseif dir == 'cs' then            -- it's right - do nothing
    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
      if strong_lr == 'r' and type_n ~= '' then
        dir_mark(head, first_n, last_n, 'r')
      elseif strong_lr == 'l' and first_d and type_n == 'an' then
        dir_mark(head, first_n, last_n, 'r')
        dir_mark(head, first_d, last_d, outer)
        first_d, last_d = nil, nil
      elseif strong_lr == 'l' and type_n ~= '' then
        last_d = last_n
      end
      type_n = ''
      first_n, last_n = nil, nil
    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:

```
    if dir == 'l' or dir == 'r' then
      if dir ~= outer then
        first_d = first_d or item
        last_d = item
      elseif first_d and dir ~= strong_lr then
        dir_mark(head, first_d, last_d, outer)
        first_d, last_d = nil, nil
     end
    end
```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it’s clearly <r> and <l>, resptly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when `last_lr` is nil) of an R text, they are mirrored directly.
TODO - numbers in R mode are processed. It doesn’t hurt, but should not be done.

```
    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
      item.char = characters[item.char] and
                  characters[item.char].m or item.char
    elseif (dir or new_dir) and last_lr ~= item then
      local mir = outer .. strong_lr .. (dir or outer)
      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
        for ch in node.traverse(node.next(last_lr)) do
          if ch == item then break end
          if ch.id == node.id'glyph' then
            ch.char = characters[ch.char].m or ch.char
          end
        end
      end
    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`).

```
    if dir == 'l' or dir == 'r' then
      last_lr = item
      strong = dir_real            -- Don't search back - best save now
      strong_lr = (strong == 'l') and 'l' or 'r'
    elseif new_dir then
      last_lr = nil
    end
  end
```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```
  if last_lr and outer == 'r' then
    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
      ch.char = characters[ch.char].m or ch.char
    end
  end
  if first_n then
    dir_mark(head, first_n, last_n, outer)
  end
  if first_d then
    dir_mark(head, first_d, last_d, outer)
  end
```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```
  return node.prev(head) or head
end
⟨/basic-r⟩
```

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.

```
⟨*nil⟩
\ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
\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.

```
\ifx\l@nohyphenation\@undefined
   \@nopatterns{nil}
   \adddialect\l@nil0
\else
   \let\l@nil\l@nohyphenation
\fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the 'nil' language.

`\captionnil`
`\datenil`

```
\let\captionsnil\@empty
\let\datenil\@empty
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{nil}
⟨/nil⟩
```

16 Support for Plain TEX (`plain.def`)

16.1 Not renaming `hyphen.tex`

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based TEX-format. When asked he responded:

> That file name is "sacred", and if anybody changes it they will cause severe upward/downward compatibility headaches.
>
> People can have a file localhyphen.tex or whatever they like, but they mustn't diddle with hyphen.tex (or plain.tex except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to acheive the desired effect, based on the `babel` package. If you load each of them with iniTEX, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.
As these files are going to be read as the first thing iniTEX sees, we need to set some category codes just to be able to change the definition of `\input`

```
⟨*bplain | blplain⟩
\catcode`\{=1 % left brace is begin-group character
\catcode`\}=2 % right brace is end-group character
\catcode`\#=6 % hash mark is macro parameter character
```

Now let's see if a file called `hyphen.cfg` can be found somewhere on TEX's input path by trying to open it for reading...

```
\openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
\ifeof0
\else
```

When hyphen.cfg could be opened we make sure that *it* will be read instead of the file hyphen.tex which should (according to Don Knuth's ruling) contain the american English hyphenation patterns and nothing else.

We do this by first saving the original meaning of \input (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
  \let\a\input
```

Then \input is defined to forget about its argument and load hyphen.cfg instead.

```
  \def\input #1 {%
    \let\input\a
    \a hyphen.cfg
```

Once that's done the original meaning of \input can be restored and the definition of \a can be forgotten.

```
    \let\a\undefined
  }
\fi
⟨/bplain | blplain⟩
```

Now that we have made sure that hyphen.cfg will be loaded at the right moment it is time to load plain.tex.

```
⟨bplain⟩\a plain.tex
⟨blplain⟩\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.

```
⟨bplain⟩\def\fmtname{babel-plain}
⟨blplain⟩\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 LaTeX features

The following code duplicates or emulates parts of LaTeX 2_ε that are needed for babel.

```
⟨*plain⟩
\def\@empty{}
\def\loadlocalcfg#1{%
  \openin0#1.cfg
  \ifeof0
    \closein0
  \else
    \closein0
    {\immediate\write16{*************************************}%
     \immediate\write16{* Local config file #1.cfg used}%
     \immediate\write16{*}%
     }
    \input #1.cfg\relax
  \fi
  \@endofldf}
```

16.3 General tools

A number of LaTeX macro's that are needed later on.

```
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
```

```
\def\@nnil{\@nil}
\def\@gobbletwo#1#2{}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@star@or@long#1{%
  \@ifstar
  {\let\l@ngrel@x\relax#1}%
  {\let\l@ngrel@x\long#1}}
\let\l@ngrel@x\relax
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\let\@typeset@protect\relax
\let\protected@edef\edef
\long\def\@gobble#1{}
\edef\@backslashchar{\expandafter\@gobble\string\\}
\def\strip@prefix#1>{}
\def\g@addto@macro#1#2{{%
    \toks@\expandafter{#1#2}%
    \xdef#1{\the\toks@}}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{%
  \expandafter\ifx\csname#1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\def\@expandtwoargs#1#2#3{%
  \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{%
  #1%
  \ifx#2\@empty\else\expandafter\zap@space\fi
  #2}
```

LaTeX 2$_\varepsilon$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
\ifx\@preamblecmds\@undefined
  \def\@preamblecmds{}
\fi
\def\@onlypreamble#1{%
  \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
    \@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble
```

Mimick LaTeX's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```
\def\begindocument{%
  \@begindocumenthook
  \global\let\@begindocumenthook\@undefined
  \def\do##1{\global\let##1\@undefined}%
  \@preamblecmds
  \global\let\do\noexpand}
```

```
\ifx\@begindocumenthook\@undefined
  \def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
```

We also have to mimick LaTeX's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```
\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\def\@endofldf{}
\@onlypreamble\@endofldf
\let\bbl@afterlang\@empty
\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.

```
\ifx\if@filesw\@undefined
  \expandafter\let\csname if@filesw\expandafter\endcsname
    \csname iffalse\endcsname
\fi
```

Mimick LaTeX's commands to define control sequences.

```
\def\newcommand{\@star@or@long\new@command}
\def\new@command#1{%
  \@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]{%
  \@ifnextchar [{\@xargdef#1[#2]}%
                {\@argdef#1[#2]}}
\long\def\@argdef#1[#2]#3{%
  \@yargdef#1\@ne{#2}{#3}}
\long\def\@xargdef#1[#2][#3]#4{%
  \expandafter\def\expandafter#1\expandafter{%
    \expandafter\@protected@testopt\expandafter #1%
    \csname\string#1\expandafter\endcsname{#3}}%
  \expandafter\@yargdef \csname\string#1\endcsname
  \tw@{#2}{#4}}
\long\def\@yargdef#1#2#3{%
  \@tempcnta#3\relax
  \advance \@tempcnta \@ne
  \let\@hash@\relax
  \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
  \@tempcntb #2%
  \@whilenum\@tempcntb <\@tempcnta
  \do{%
    \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
    \advance\@tempcntb \@ne}%
  \let\@hash@##%
  \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
\def\providecommand{\@star@or@long\provide@command}
\def\provide@command#1{%
  \begingroup
    \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
  \endgroup
  \expandafter\@ifundefined\@gtempa
    {\def\reserved@a{\new@command#1}}%
    {\let\reserved@a\relax
     \def\reserved@a{\new@command\reserved@a}}%
   \reserved@a}%

\def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
\def\declare@robustcommand#1{%
   \edef\reserved@a{\string#1}%
   \def\reserved@b{#1}%
   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
   \edef#1{%
      \ifx\reserved@a\reserved@b
```

```
         \noexpand\x@protect
         \noexpand#1%
      \fi
      \noexpand\protect
      \expandafter\noexpand\csname\bbl@stripslash#1 \endcsname
   }%
   \expandafter\new@command\csname\bbl@stripslash#1 \endcsname
}
\def\x@protect#1{%
   \ifx\protect\@typeset@protect\else
      \@x@protect#1%
   \fi
}
\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`.

```
\def\bbl@tempa{\csname newif\endcsname\ifin@}
\ifx\in@\@undefined
  \def\in@#1#2{%
    \def\in@@##1#1##2##3\in@@{%
      \ifx\in@##2\in@false\else\in@true\fi}%
    \in@@#2#1\in@\in@@}
\else
  \let\bbl@tempa\@empty
\fi
\bbl@tempa
```

LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```
\def\@ifpackagewith#1#2#3#4{#3}
```

The LaTeX macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain TeX but we need the macro to be defined as a no-op.

```
\def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their LaTeX 2_ε versions; just enough to make things work in plain TeXenvironments.

```
\ifx\@tempcnta\@undefined
  \csname newcount\endcsname\@tempcnta\relax
\fi
\ifx\@tempcntb\@undefined
  \csname newcount\endcsname\@tempcntb\relax
\fi
```

To prevent wasting two counters in LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
\ifx\bye\@undefined
  \advance\count10 by -2\relax
```

```
\fi
\ifx\@ifnextchar\@undefined
  \def\@ifnextchar#1#2#3{%
    \let\reserved@d=#1%
    \def\reserved@a{#2}\def\reserved@b{#3}%
    \futurelet\@let@token\@ifnch}
  \def\@ifnch{%
    \ifx\@let@token\@sptoken
      \let\reserved@c\@xifnch
    \else
      \ifx\@let@token\reserved@d
        \let\reserved@c\reserved@a
      \else
        \let\reserved@c\reserved@b
      \fi
    \fi
    \reserved@c}
  \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
  \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
\fi
\def\@testopt#1#2{%
  \@ifnextchar[{#1}{#1[#2]}}
\def\@protected@testopt#1{%
  \ifx\protect\@typeset@protect
    \expandafter\@testopt
  \else
    \@x@protect#1%
  \fi}
\long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
     #2\relax}\fi}
\long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
         \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain TEX environment.

```
\def\DeclareTextCommand{%
   \@dec@text@cmd\providecommand
}
\def\ProvideTextCommand{%
   \@dec@text@cmd\providecommand
}
\def\DeclareTextSymbol#1#2#3{%
   \@dec@text@cmd\chardef#1{#2}#3\relax
}
\def\@dec@text@cmd#1#2#3{%
   \expandafter\def\expandafter#2%
      \expandafter{%
         \csname#3-cmd\expandafter\endcsname
         \expandafter#2%
         \csname#3\string#2\endcsname
      }%
%   \let\@ifdefinable\@rc@ifdefinable
   \expandafter#1\csname#3\string#2\endcsname
}
\def\@current@cmd#1{%
  \ifx\protect\@typeset@protect\else
      \noexpand#1\expandafter\@gobble
  \fi
}
\def\@changed@cmd#1#2{%
   \ifx\protect\@typeset@protect
      \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
         \expandafter\ifx\csname ?\string#1\endcsname\relax
            \expandafter\def\csname ?\string#1\endcsname{%
               \@changed@x@err{#1}%
            }%
         \fi
         \global\expandafter\let
           \csname\cf@encoding \string#1\expandafter\endcsname
           \csname ?\string#1\endcsname
      \fi
      \csname\cf@encoding\string#1%
        \expandafter\endcsname
   \else
      \noexpand#1%
   \fi
}
\def\@changed@x@err#1{%
    \errhelp{Your command will be ignored, type <return> to proceed}%
    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
\def\DeclareTextCommandDefault#1{%
   \DeclareTextCommand#1?%
}
\def\ProvideTextCommandDefault#1{%
   \ProvideTextCommand#1?%
}
\expandafter\let\csname OT1-cmd\endcsname\@current@cmd
\expandafter\let\csname?-cmd\endcsname\@changed@cmd
\def\DeclareTextAccent#1#2#3{%
  \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
}
\def\DeclareTextCompositeCommand#1#2#3#4{%
   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
   \edef\reserved@b{\string##1}%
   \edef\reserved@c{%
     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
   \ifx\reserved@b\reserved@c
      \expandafter\expandafter\expandafter\ifx
         \expandafter\@car\reserved@a\relax\relax\@nil
         \@text@composite
      \else
         \edef\reserved@b##1{%
            \def\expandafter\noexpand
               \csname#2\string#1\endcsname####1{%
               \noexpand\@text@composite
                  \expandafter\noexpand\csname#2\string#1\endcsname
                  ####1\noexpand\@empty\noexpand\@text@composite
                  {##1}%
            }%
         }%
         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
      \fi
      \expandafter\def\csname\expandafter\string\csname
         #2\endcsname\string#1-\string#3\endcsname{#4}
   \else
     \errhelp{Your command will be ignored, type <return> to proceed}%
     \errmessage{\string\DeclareTextCompositeCommand\space used on
         inappropriate command \protect#1}
   \fi
}
\def\@text@composite#1#2#3\@text@composite{%
   \expandafter\@text@composite@x
      \csname\string#1-\string#2\endcsname
}
\def\@text@composite@x#1#2{%
   \ifx#1\relax
      #2%
   \else
      #1%
   \fi
}
%
\def\@strip@args#1:#2-#3\@strip@args{#2}
\def\DeclareTextComposite#1#2#3#4{%
   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
   \bgroup
      \lccode`\@=#4%
      \lowercase{%
   \egroup
      \reserved@a @%
   }%
}
%
\def\UseTextSymbol#1#2{%
%   \let\@curr@enc\cf@encoding
%   \@use@text@encoding{#1}%
   #2%
%   \@use@text@encoding\@curr@enc
}
\def\UseTextAccent#1#2#3{%
%   \let\@curr@enc\cf@encoding
%   \@use@text@encoding{#1}%
%   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
%   \@use@text@encoding\@curr@enc
}
\def\@use@text@encoding#1{%
%   \edef\f@encoding{#1}%
%   \xdef\font@name{%
%       \csname\curr@fontshape/\f@size\endcsname
%   }%
%   \pickup@font
%   \font@name
%   \@@enc@update
}
\def\DeclareTextSymbolDefault#1#2{%
   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
}
\def\DeclareTextAccentDefault#1#2{%
   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
}
\def\cf@encoding{OT1}
```

Currently we only use the LaTeX 2ε method for accents for those that are known to be made active in *some* language definition file.

```
\DeclareTextAccent{\"}{OT1}{127}
```

```
\DeclareTextAccent{\'}{OT1}{19}
\DeclareTextAccent{\^}{OT1}{94}
\DeclareTextAccent{\`}{OT1}{18}
\DeclareTextAccent{\~}{OT1}{126}
```

The following control sequences are used in `babel.def` but are not defined for plain TeX.

```
\DeclareTextSymbol{\textquotedblleft}{OT1}{92}
\DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
\DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
\DeclareTextSymbol{\textquoteright}{OT1}{`\'}
\DeclareTextSymbol{\i}{OT1}{16}
\DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the LaTeX-control sequence `\scriptsize` to be available. Because plain TeX doesn't have such a sofisticated font mechanism as LaTeX has, we just `\let` it to `\sevenrm`.

```
\ifx\scriptsize\@undefined
  \let\scriptsize\sevenrm
\fi
```

16.5 Babel options

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

```
\let\bbl@opt@shorthands\@nnil
\def\bbl@ifshorthand#1#2#3{#2}%
\ifx\babeloptionstrings\@undefined
  \let\bbl@opt@strings\@nnil
\else
  \let\bbl@opt@strings\babeloptionstrings
\fi
\def\bbl@tempa{normal}
\ifx\babeloptionmath\bbl@tempa
  \def\bbl@mathnormal{\noexpand\textormath}
\fi
\def\BabelStringsDefault{generic}
\ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
\let\bbl@afterlang\relax
\let\bbl@language@opts\@empty
\ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
\def\AfterBabelLanguage#1#2{}
⟨/plain⟩
```

17 Acknowledgements

I would like to thank all who volunteered as β-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.
During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

[1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.

[2] Donald E. Knuth, *The TEXbook,* Addison-Wesley, 1986.

[3] Leslie Lamport, *LATEX, A document preparation System*, Addison-Wesley, 1986.

[4] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst.* SDU Uitgeverij ('s-Gravenhage, 1988).

[5] Hubert Partl, *German TEX*, *TUGboat* 9 (1988) #1, p. 70–72.

[6] Leslie Lamport, in: TEXhax Digest, Volume 89, #13, 17 February 1989.

[7] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national LATEX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

[8] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.

[9] Joachim Schrod, *International LATEX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

[10] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using LATEX*, Springer, 2002, p. 301–373.