

Babel

Version 3.22

2018/06/05

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 (Xe \TeX and Lua \TeX) 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	8
1.7	Basic language selectors	8
1.8	Auxiliary language selectors	9
1.9	More on selection	10
1.10	Shorthands	11
1.11	Package options	14
1.12	The base option	16
1.13	ini files	17
1.14	Selecting fonts	23
1.15	Modifying a language	24
1.16	Creating a language	25
1.17	Digits	27
1.18	Getting the current language name	27
1.19	Hyphenation tools	27
1.20	Selecting scripts	29
1.21	Selecting directions	29
1.22	Language attributes	32
1.23	Hooks	32
1.24	Languages supported by babel	34
1.25	Tips, workarounds, know issues and notes	35
1.26	Current and future work	36
1.27	Tentative and experimental code	37
2	Loading languages with language.dat	38
2.1	Format	39
3	The interface between the core of babel and the language definition files	39
3.1	Guidelines for contributed languages	41
3.2	Basic macros	41
3.3	Skeleton	42
3.4	Support for active characters	43
3.5	Support for saving macro definitions	44
3.6	Support for extending macros	44
3.7	Macros common to a number of languages	44
3.8	Encoding-dependent strings	44
4	Changes	48
4.1	Changes in babel version 3.9	48
4.2	Changes in babel version 3.7	49
II	The code	49
5	Identification and loading of required files	49

6	locale directory	50
7	Tools	50
7.1	Multiple languages	54
8	The Package File (\LaTeX, babel.sty)	55
8.1	base	55
8.2	key=value options and other general option	57
8.3	Conditional loading of shorthands	58
8.4	Language options	59
9	The kernel of Babel (babel.def, common)	62
9.1	Tools	62
9.2	Hooks	65
9.3	Setting up language files	66
9.4	Shorthands	68
9.5	Language attributes	77
9.6	Support for saving macro definitions	80
9.7	Short tags	80
9.8	Hyphens	81
9.9	Multiencoding strings	82
9.10	Macros common to a number of languages	88
9.11	Making glyphs available	88
9.11.1	Quotation marks	89
9.11.2	Letters	90
9.11.3	Shorthands for quotation marks	91
9.11.4	Umlauts and tremas	92
9.12	Layout	93
9.13	Creating languages	94
10	The kernel of Babel (babel.def, only \LaTeX)	101
10.1	The redefinition of the style commands	101
10.2	Cross referencing macros	101
10.3	Marks	105
10.4	Preventing clashes with other packages	106
10.4.1	ifthen	106
10.4.2	varioref	106
10.4.3	hhline	107
10.4.4	hyperref	107
10.4.5	fancyhdr	108
10.5	Encoding and fonts	108
10.6	Basic bidi support	110
10.7	Local Language Configuration	112
11	Multiple languages (switch.def)	113
11.1	Selecting the language	114
11.2	Errors	122
12	Loading hyphenation patterns	123
13	Font handling with fontspec	128

14	Hooks for XeTeX and LuaTeX	131
14.1	XeTeX	131
14.2	Layout	133
14.3	LuaTeX	136
14.4	Layout	141
14.5	Auto bidi with basic-r	143
15	The ‘nil’ language	154
16	Support for Plain T_EX (plain.def)	154
16.1	Not renaming hyphen.tex	154
16.2	Emulating some L ^A T _E X features	155
16.3	General tools	156
16.4	Encoding related macros	159
17	Acknowledgements	162

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	4
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	7
Unknown language ‘LANG’	7
Argument of \language@active@arg” has an extra }	11

Part I

User guide

- This user guide focuses on \LaTeX . There are also some notes on its use with Plain \TeX .
- Changes and new features with relation to version 3.8 are highlighted with **New X.XX**. The most recent features could be still unstable. Please, report any issues you find.
- If you are interested in the \TeX multilingual support, please join the kadingira list on <http://tug.org/mailman/listinfo/kadingira>. You can follow the development of babel on <https://github.com/latex3/latex2e/tree/master/required/babel> (which provides some sample files, too).
- See section 3.1 for contributing a language.
- The first sections describe the traditional way of loading a language (with ldf files). The alternative way based on ini files, which complements the previous one (it will *not* replace it), is described below.

1 The user interface

1.1 Monolingual documents

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

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

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

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

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

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

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 `\languagenam` (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.

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

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

```
\documentclass{article}

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

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

1.3 Modifiers

New 3.9c The basic behavior of some languages can be modified when loading `babel` by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the `main` key 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 luatex

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 `lrmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

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

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 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{\langle language \rangle}`) 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

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

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


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

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

1.6 Plain

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

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

WARNING Not all languages provide a `sty` file and some of them are not compatible with Plain.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` $\langle language \rangle$

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

`\begin{otherlanguage}` $\langle\textit{language}\rangle$... `\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:

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

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

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

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

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

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

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

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

```
text \textde{German text} 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{\langle tag \rangle}`, namely, it is not affected by `\MakeUppercase` (while `\foreignlanguage` is).

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

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

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

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

```
\babelensure{polish}
```

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

⁵With it encoded string may not work as expected.

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

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

1.10 Shorthands

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

NOTE Note the following:

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

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

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

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

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

`\aliasshorthand` $\langle original \rangle \langle alias \rangle$

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

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

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

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

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

`\languageshorthands` $\langle language \rangle$

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).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, `\usesshorthands`.) 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` $\langle shorthand \rangle$

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

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

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

⁶Actually, 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.

⁷Thanks to Enrico Gregorio

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

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

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

1.11 Package options

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

- KeepShorthandsActive** Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
- activeacute** For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
- activegrave** Same for `.
- shorthands=** `<char><char>... | off`
 The only language shorthands activated are those given, like, eg:

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

If ' is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \string (otherwise they will be expanded by L^AT_EX 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

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

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 spoiled by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.

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

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

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

`strings=` generic | unicode | encoded | *<label>* | **

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 `\MakeUpper case` and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).

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

⁹You can use alternatively the package `silence`.

New 3.9g Sets the behavior 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=

New 3.14 Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.21.

layout=

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.21.

1.12 The base option

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

¹⁰Turned off in plain.

¹¹Duplicated options count as several ones.

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

```
\let\macro\relax}
\usepackage[foo,bar]{babel}
```

1.13 ini files

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

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

```
\documentclass{book}

\usepackage{babel}
\babelprovide[import=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}	bem	Bemba
agq	Aghem	bez	Bena
ak	Akan	bg	Bulgarian ^{ul}
am	Amharic ^{ul}	bm	Bambara
ar	Arabic ^{ul}	bn	Bangla ^{ul}
ar-DZ	Arabic ^{ul}	bo	Tibetan ^u
ar-MA	Arabic ^{ul}	brx	Bodo
ar-SY	Arabic ^{ul}	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian ^{ul}
asa	Asu	bs	Bosnian ^{ul}
ast	Asturian ^{ul}	ca	Catalan ^{ul}
az-Cyrl	Azerbaijani	ce	Chechen
az-Latn	Azerbaijani	cgg	Chiga
az	Azerbaijani ^{ul}	chr	Cherokee
bas	Basaa	ckb	Central Kurdish
be	Belarusian ^{ul}	cs	Czech ^{ul}

cy	Welsh ^{ul}	hy	Armenian
da	Danish ^{ul}	ia	Interlingua ^{ul}
dav	Taita	id	Indonesian ^{ul}
de-AT	German ^{ul}	ig	Igbo
de-CH	German ^{ul}	ii	Sichuan Yi
de	German ^{ul}	is	Icelandic ^{ul}
dje	Zarma	it	Italian ^{ul}
dsb	Lower Sorbian ^{ul}	ja	Japanese
dua	Duala	jgo	Ngomba
dyo	Jola-Fonyi	jmc	Machame
dz	Dzongkha	ka	Georgian ^{ul}
ebu	Embu	kab	Kabyle
ee	Ewe	kam	Kamba
el	Greek ^{ul}	kde	Makonde
en-AU	English ^{ul}	kea	Kabuverdianu
en-CA	English ^{ul}	khq	Koyra Chiini
en-GB	English ^{ul}	ki	Kikuyu
en-NZ	English ^{ul}	kk	Kazakh
en-US	English ^{ul}	kkj	Kako
en	English ^{ul}	kl	Kalaallisut
eo	Esperanto ^{ul}	kln	Kalenjin
es-MX	Spanish ^{ul}	km	Khmer
es	Spanish ^{ul}	kn	Kannada ^{ul}
et	Estonian ^{ul}	ko	Korean
eu	Basque ^{ul}	kok	Konkani
ewo	Ewondo	ks	Kashmiri
fa	Persian ^{ul}	ksb	Shambala
ff	Fulah	ksf	Bafia
fi	Finnish ^{ul}	ksh	Colognian
fil	Filipino	kw	Cornish
fo	Faroese	ky	Kyrgyz
fr	French ^{ul}	lag	Langi
fr-BE	French ^{ul}	lb	Luxembourgish
fr-CA	French ^{ul}	lg	Ganda
fr-CH	French ^{ul}	lkt	Lakota
fr-LU	French ^{ul}	ln	Lingala
fur	Friulian ^{ul}	lo	Lao ^{ul}
fy	Western Frisian	lrc	Northern Luri
ga	Irish ^{ul}	lt	Lithuanian ^{ul}
gd	Scottish Gaelic ^{ul}	lu	Luba-Katanga
gl	Galician ^{ul}	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian ^{ul}
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa ¹	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ¹

ms-SG	Malay ^l	sl	Slovenian ^{ul}
ms	Malay ^{ul}	smn	Inari Sami
mt	Maltese	sn	Shona
mua	Mundang	so	Somali
my	Burmese	sq	Albanian ^{ul}
mzn	Mazanderani	sr-Cyrl-BA	Serbian ^{ul}
naq	Nama	sr-Cyrl-ME	Serbian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Cyrl-XK	Serbian ^{ul}
nd	North Ndebele	sr-Cyrl	Serbian ^{ul}
ne	Nepali	sr-Latn-BA	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Latn-ME	Serbian ^{ul}
nmg	Kwasio	sr-Latn-XK	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sr-Latn	Serbian ^{ul}
nnh	Ngiemboon	sr	Serbian ^{ul}
nus	Nuer	sv	Swedish ^{ul}
nyn	Nyankole	sw	Swahili
om	Oromo	ta	Tamil ^u
or	Odia	te	Telugu ^{ul}
os	Ossetic	teo	Teso
pa-Arab	Punjabi	th	Thai ^{ul}
pa-Guru	Punjabi	ti	Tigrinya
pa	Punjabi	tk	Turkmen ^{ul}
pl	Polish ^{ul}	to	Tongan
pms	Piedmontese ^{ul}	tr	Turkish ^{ul}
ps	Pashto	twq	Tasawaq
pt-BR	Portuguese ^{ul}	tzm	Central Atlas Tamazight
pt-PT	Portuguese ^{ul}	ug	Uyghur
pt	Portuguese ^{ul}	uk	Ukrainian ^{ul}
qu	Quechua	ur	Urdu ^{ul}
rm	Romansh ^{ul}	uz-Arab	Uzbek
rn	Rundi	uz-Cyrl	Uzbek
ro	Romanian ^{ul}	uz-Latn	Uzbek
rof	Rombo	uz	Uzbek
ru	Russian ^{ul}	vai-Latn	Vai
rw	Kinyarwanda	vai-Vaii	Vai
rwk	Rwa	vai	Vai
sa-Beng	Sanskrit	vi	Vietnamese ^{ul}
sa-Deva	Sanskrit	vun	Vunjo
sa-Gujr	Sanskrit	wae	Walser
sa-Knda	Sanskrit	xog	Soga
sa-Mlym	Sanskrit	yav	Yangben
sa-Telu	Sanskrit	yi	Yiddish
sa	Sanskrit	yo	Yoruba
sah	Sakha	yue	Cantonese
saq	Samburu	zgh	Standard Moroccan Tamazight
sbp	Sangu		
se	Northern Sami ^{ul}	zh-Hans-HK	Chinese
seh	Sena	zh-Hans-MO	Chinese
ses	Koyraboro Senni	zh-Hans-SG	Chinese
sg	Sango	zh-Hans	Chinese
shi-Latn	Tachelhit	zh-Hant-HK	Chinese
shi-Tfng	Tachelhit	zh-Hant-MO	Chinese
shi	Tachelhit	zh-Hant	Chinese
si	Sinhala	zh	Chinese
sk	Slovak ^{ul}	zu	Zulu

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

aghem	centralkurdish
akan	chechen
albanian	cherokee
american	chiga
amharic	chinese-hans-hk
arabic	chinese-hans-mo
arabic-algeria	chinese-hans-sg
arabic-DZ	chinese-hans
arabic-morocco	chinese-hant-hk
arabic-MA	chinese-hant-mo
arabic-syria	chinese-hant
arabic-SY	chinese-simplified-hongkongsarchina
armenian	chinese-simplified-macausarchina
assamese	chinese-simplified-singapore
asturian	chinese-simplified
asu	chinese-traditional-hongkongsarchina
australian	chinese-traditional-macausarchina
austrian	chinese-traditional
azerbaijani-cyrillic	chinese
azerbaijani-cyrl	colognian
azerbaijani-latin	cornish
azerbaijani-latn	croatian
azerbaijani	czech
bafia	danish
bambara	duala
basaa	dutch
basque	dzongkha
belarusian	embu
bemba	english-au
bena	english-australia
bengali	english-ca
bodo	english-canada
bosnian-cyrillic	english-gb
bosnian-cyrl	english-newzealand
bosnian-latin	english-nz
bosnian-latn	english-unitedkingdom
bosnian	english-unitedstates
brazilian	english-us
breton	english
british	esperanto
bulgarian	estonian
burmese	ewe
canadian	ewondo
cantonese	faroesse
catalan	filipino
centralatlastamazight	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
norwegiannorsk
nswissgerman
nuer
nyankole
nynorsk
occitan

oriya
oromo
ossetic
pashto
persian
piedmontese
polish
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo

serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq
telugu
teso
thai
tibetan
tigrinya
tongan
turkish
turkmen
ukenglish
ukrainian
upporsorbian
urdu
usenglish
usorbian
uyghur
uzbek-arab
uzbek-arabic
uzbek-cyrillic
uzbek-cyrl
uzbek-latin
uzbek-latn
uzbek
vai-latin

vai-latn	welsh
vai-vai	westernfrisian
vai-vaii	yangben
vai	yiddish
vietnam	yoruba
vietnamese	zarma
vunjo	zulu afrikaans
walser	

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`.¹³

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

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

If no language is given, then it is considered the default font for the family, activated when a language is selected. On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, `*devanagari`). Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

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

```
\documentclass{article}

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

\babelprovide[import=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.

¹³See also the package `combofont` for a complementary approach.

EXAMPLE Here is how to do it:

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

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

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

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

This makes sure the OpenType script for Devanagari is `deva` and not `dev2` (luatex does not detect automatically the correct script¹⁴).

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

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

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

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

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

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

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

- 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\mylangchaptername{..}
(babel)                Reported on input line 18.
```

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

EXAMPLE If you need a language named arhinish:

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

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

import= *<language-tag>*

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

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

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

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{\the\year}{\the\month}{\the\day}.

captions= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

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

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

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

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

script= *<script-name>*

New 3.15 Sets the script name to be used by fontspec (eg, Devanagar i). Overrides the value in the ini file. This value is particularly important because it sets the writing direction.

`language=` \langle language-name \rangle

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

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

1.17 Digits

New 3.20 A few ini files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering).

For example:

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

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

1.18 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` \langle language \rangle \langle true \rangle \langle false \rangle

If more than one language is used, it might be necessary to know which language is active at a specific time. This can be checked by a call to `\iflanguage`, but note here “language” is used in the T_EX 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.

1.19 Hyphenation tools

`\babelhyphen` * \langle type \rangle

`\babelhyphen` *{<text>}

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

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

With all of them hyphenation in the rest of the word is enabled. If you don’t want 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 `\babelnu1lhyphen`; (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*,¹⁵ 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.¹⁶

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

`\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 assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

1.21 Selecting directions

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

¹⁵With *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.

¹⁶The 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.

¹⁷But still defined for backwards compatibility.

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.¹⁸ This is particularly true for graphical elements, including the picture environment and PDF or PS based graphics.

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

There are some package options controlling bidi writing.

`bidi=` default | basic-r | basic

New 3.14 Selects the bidi algorithm to be used. With `default` the bidi mechanism is just activated (by default it is not), but every change must be marked up. In `xetex` and `pdftex` this is the only option. In `luatex`, `basic-r` provides a simple and fast method for R text, which handles numbers and unmarked L text within an R context. **New 3.19** Finally, `basic` supports both L and R text (see 1.27). (They are named `basic` mainly because they only consider the intrinsic direction of scripts and weak directionality.)

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.¹⁹

```
\documentclass{article}

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

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

\babelfont{rm}{FreeSerif}

\begin{document}

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

`layout=` sectioning | counters | lists | contents | footnotes | captions | columns | extras

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements. You may use several options with a comma-separated list (eg, `layout=counters.contents.sectioning`). This list will be expanded in future releases (tables, captions, etc.). Note not all options are required by all engines.

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

¹⁸A basic stable version for `luatex` is planned before (Northern) Summer 2018. Other engines must wait very likely until (Northern) Winter.

¹⁹At the time of this writing some Arabic fonts are not rendered correctly by the default `luatex` font loader, with misplaced kerns inside some words, so double check the resulting text. It seems a fix is on the way, but in the meanwhile you could have a look at the workaround available on GitHub, under `/required/babel/samples`

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

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

lists required in xetex and pdftex, but only in multilingual documents in luatex.

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

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

footnotes not required in monolingual documents, but it may be useful in multilingual documents in all engines; you may use alternatively `\BabelFootnote` described below (what this options does exactly is also explained there).

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

tabular required in luatex for R `tabular` (it has been tested only with simple tables, so expect some readjustments in the future); ignored in pdftex or xetex (which will not support a similar option in the short term) [New 3.18](#) ,

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

`\babelsublr` `{\lr-text}`

Digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic-r` and, usually, xetex). This command is provided to set `{\lr-text}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `r1` counterpart.

Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. This is by design to provide the proper behaviour in the most usual cases — but if you need to use `\ref` in an L text inside R, it must be marked up explicitly.

`\BabelPatchSection` `{\section-name}`

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

`\BabelFootnote` `{\cmd}{\local-language}{\before}{\after}`

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

New 3.17 Something like:

```
\BabelFootnote{\parsfootnote}{\language}{\footnote}
```

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

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

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

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

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

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

```
\BabelFootnote{\enfootnote}{english}{\footnote}
```

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

1.22 Language attributes

`\languageattribute`

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

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

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

1.23 Hooks

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

`\AddBabelHook` $\langle name \rangle \langle event \rangle \langle code \rangle$

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook` $\langle name \rangle$, `\DisableBabelHook` $\langle name \rangle$.

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:

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the the above, if necessary.

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

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

afterextras Just after executing `\extras` $\langle language \rangle$. 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` $\langle language \rangle$ and `\date` $\langle language \rangle$.

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).

1.24 Languages supported by babel

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

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

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

Most of them work out of the box, but some may require extra fonts, encoding files, a

preprocessor or even a complete framework (like CJK). 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 .

1.25 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 `hline` to make sure `:` has the right one, so if you want to change the catcode of `|` it has to be done using the same method at the proper place, with

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

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

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

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²¹ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesshorthands` 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).

²¹This explains why \LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingsphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the ‘to do’ list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.26 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{babel}

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

\babelfont{rm}{FreeSerif}

\directlua{
local GLYPH = node.id'glyph'
function insertsp (head)
  local size = 0
  for item in node.traverse(head) do
    local i = item.id
    if i == GLYPH 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.²² But that is the easy part, because they don't require modifying the L^AT_EX 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)-ból”, 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.

1.27 Tentative and experimental code

Option `bidi=basic`

New 3.19 With this package option *both* L and R text can be mixed without explicit markup (the latter will be only necessary in some special cases where the Unicode algorithm fails). It is used much like `bidi=basic-r`, but with R text inside L text you may want to map the font so that the correct features are in force. This is accomplished with an option in `\babelprovide`, as illustrated:

```

\documentclass{book}

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

\babelprovide[mapfont=direction]{arabic}

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

\begin{document}

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

\end{document}

```

²²See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those system, however, have limited application to T_EX because their aim is just to display information and not fine typesetting.

What `\mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language (arabic in this case), then change its font to that set for this language’ (here defined via `*arabic`, because Crimson does not provide Arabic letters). Boxes are “black boxes”. Numbers inside an `\hbox` (as for example in a `\ref`) do not know anything about the surrounding chars. So, `\ref{A}-\ref{B}` are not rendered in the visual order A-B, but in the wrong one B-A (because the hyphen does not “see” the digits inside the `\hbox`’es). If you need `\ref` ranges, the best option is to define a dedicated macro like this (to avoid explicit direction changes in the body; here `\text` must be defined to select the main language):

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

In a future a more complete method, reading recursively boxed text, may be added. There are samples on GitHub, under `/required/babel/samples`: `lua-bidibasic.tex` and `lua-secenum.tex`.

Old stuff

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

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

So, for example:

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

Bidi writing in `luatex` is under development, but a basic implementation is almost finished. On the other hand, in `xetex` it is taking its first steps. The latter engine poses quite different challenges. An option to manage document layout in `luatex` (lists, footnotes, etc.) is almost finished, but `xetex` required more work.

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

New 3.9q With `luatex`, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).²³ Until

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

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).²⁴

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 stored²⁵. 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.²⁶ For example:

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

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

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

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

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

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

The *language definition files* (`ldf`) must conform to a number of conventions, because these files have to fill in the gaps left by the common code in `babel.def`, i. e., the definitions of

²⁴The 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`.

²⁵This is because different operating systems sometimes use *very* different file-naming conventions.

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

the macros that produce texts. Also the language-switching possibility which has been built into the babel system has its implications.

The following assumptions are made:

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

Some recommendations:

- The preferred shorthand is `"`, which is not used in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ (quotes are entered as `` `` and `' '`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras\langle lang \rangle` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low level) or the language (high level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁷
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

²⁷But not removed, for backward compatibility.

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

3.1 Guidelines for contributed languages

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

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

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

3.2 Basic macros

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

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

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

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

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

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

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters

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

3.3 Skeleton

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

```

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

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

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

```

```

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

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

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

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

3.4 Support for active characters

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

`\initiate@active@char` The internal macro `\initiate@active@char` is used in language definition files to instruct \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` The command `\bbl@activate` is used to change the way an active character expands.

`\bbl@deactivate` `\bbl@activate` ‘switches on’ the active behavior of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

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

`\bbl@add@special` The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [2, p. 380]
`\bbl@remove@special` It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \LaTeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

Language definition files may want to *redefine* macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this²⁸.

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

`\babel@savevariable` A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\` primitive is considered to be a variable. The macro takes one argument, the $\langle variable \rangle$.
The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

`\bbl@allowhyphens` In several languages compound words are used. This means that when $T_{E}X$ has to hyphenate such a compound word, it only does so at the ‘-’ that is used in such words. To allow hyphenation in the rest of such a compound word, the macro `\bbl@allowhyphens` can be used.

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

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

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

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

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

3.8 Encoding-dependent strings

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

Furthermore, it must be activated explicitly, with the package option `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (and except if forced as described

²⁸This mechanism was introduced by Bernd Raichle.

below). In other words, the old way of defining/switching strings still works and it's used by default.

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

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

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

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

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

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

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

The $\langle category \rangle$ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.²⁹ 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:

²⁹In future releases further categories may be added.

```

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

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

\EndBabelCommands

```

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.³⁰

$\backslash\text{EndBabelCommands}$ Marks the end of the series of blocks.

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

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

³⁰This replaces in 3.9g a short-lived $\backslash\text{UseStrings}$ which has been removed because it did not work.

`\SetString` $\langle macro-name \rangle \langle string \rangle$

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

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

`\SetStringLoop` $\langle macro-name \rangle \langle string-list \rangle$

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

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

#1 is replaced by the roman numeral.

`\SetCase` $[\langle map-list \rangle] \langle toupper-code \rangle \langle tolower-code \rangle$

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would 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 $\langle map-list \rangle$ is a series of macros using the internal format of `\@uclclist` (eg, `\bb\BB\cc\CC`). The mandatory arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in L^AT_EX, 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` $\langle to-lower-macros \rangle$

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

There are three helper macros to be used inside `\SetHyphenMap`:

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

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 two 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 appropriated places in the source code and shown below with `<<name>>`. That brings a little bit of literate programming.

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the `ini` file.

version of the `ini` file

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

encodings a descriptive list of font encodings.

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

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

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

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

7 Tools

```
1 <<version=3.22>>
2 <<date=2018/06/05>>
```

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in

babel.def and in babel.sty, which means in L^AT_EX is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```

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

```

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

```

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

```

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

```

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

```

`\bbl@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.

```

28 \def\bbl@tempa#1{%
29   \long\def\bbl@trim##1##2{%
30     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
31   \def\bbl@trim@c{%
32     \ifx\bbl@trim@a\@sptoken
33       \expandafter\bbl@trim@b
34     \else
35       \expandafter\bbl@trim@b\expandafter#1%
36     \fi}%
37   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
38 \bbl@tempa{ }
39 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
40 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}

```

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

`\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.

```

41 \def\bbl@ifunset#1{%
42   \expandafter\ifx\csname#1\endcsname\relax
43   \expandafter\@firstoftwo
44   \else
45   \expandafter\@secondoftwo
46   \fi}
47 \bbl@ifunset{ifcsname}%
48 {}%
49 {\def\bbl@ifunset#1{%
50   \ifcsname#1\endcsname
51   \expandafter\ifx\csname#1\endcsname\relax
52   \bbl@afterelse\expandafter\@firstoftwo
53   \else
54   \bbl@afterfi\expandafter\@secondoftwo
55   \fi
56   \else
57   \expandafter\@firstoftwo
58   \fi}}
```

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

```

59 \def\bbl@ifblank#1{%
60   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
61 \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).

```

62 \def\bbl@forkv#1#2{%
63   \def\bbl@kvcmd##1##2##3{#2}%
64   \bbl@kvnext#1,\@nil,}
65 \def\bbl@kvnext#1,{%
66   \ifx\@nil#1\relax\else
67   \bbl@ifblank{#1}{}\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
68   \expandafter\bbl@kvnext
69   \fi}
70 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
71   \bbl@trim\def\bbl@forkv@a{#1}%
72   \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).

```

73 \def\bbl@vforeach#1#2{%
74   \def\bbl@forcmd##1{#2}%
75   \bbl@fornext#1,\@nil,}
76 \def\bbl@fornext#1,{%
77   \ifx\@nil#1\relax\else
78   \bbl@ifblank{#1}{}\bbl@trim\bbl@forcmd{#1}}%
79   \expandafter\bbl@fornext
80   \fi}
81 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

`\bbl@replace`

```

82 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
83   \toks@{}}
```

```

84 \def\bb1@replace@aux##1#2##2#2{%
85   \ifx\bb1@nil##2%
86     \toks@\expandafter{\the\toks@##1}%
87   \else
88     \toks@\expandafter{\the\toks@##1#3}%
89     \bb1@afterfi
90     \bb1@replace@aux##2#2%
91   \fi}%
92 \expandafter\bb1@replace@aux#1#2\bb1@nil#2%
93 \edef#1{\the\toks@}

```

`\bb1@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.

```

94 \def\bb1@exp#1{%
95   \begingroup
96   \let\ \noexpand
97   \def\<##1>{\expandafter\noexpand\cname##1\endcname}%
98   \edef\bb1@exp@aux{\endgroup#1}%
99   \bb1@exp@aux}

```

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

```

100 \def\bb1@ifsamestring#1#2{%
101   \begingroup
102   \protected@edef\bb1@tempb{#1}%
103   \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
104   \protected@edef\bb1@tempc{#2}%
105   \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
106   \ifx\bb1@tempb\bb1@tempc
107     \aftergroup\@firstoftwo
108   \else
109     \aftergroup\@secondoftwo
110   \fi
111 \endgroup}
112 \chardef\bb1@engine=%
113 \ifx\directlua\@undefined
114   \ifx\XeTeXinputencoding\@undefined
115     \z@
116   \else
117     \tw@
118   \fi
119 \else
120   \@ne
121 \fi
122 <</Basic macros>>

```

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

```

123 <<(*Make sure ProvidesFile is defined)>> ≡
124 \ifx\ProvidesFile\@undefined
125   \def\ProvidesFile#1[#2 #3 #4]{%
126     \wlog{File: #1 #4 #3 <#2>}%
127     \let\ProvidesFile\@undefined}
128 \fi

```

```
129 <</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`.

```
130 <<*Load patterns in luatex>> ≡
131 \ifx\directlua\@undefined\else
132   \ifx\bb1@luapatterns\@undefined
133     \input luababel.def
134   \fi
135 \fi
136 <</Load patterns in luatex>>
```

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

```
137 <<*Load macros for plain if not LaTeX>> ≡
138 \ifx\AtBeginDocument\@undefined
139   \input plain.def\relax
140 \fi
141 <</Load macros for plain if not LaTeX>>
```

7.1 Multiple languages

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

```
142 <<*Define core switching macros>> ≡
143 \ifx\language\@undefined
144   \csname newcount\endcsname\language
145 \fi
146 <</Define core switching macros>>
```

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

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

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

```
147 <<*Define core switching macros>> ≡
148 \ifx\newlanguage\@undefined
149   \csname newcount\endcsname\last@language
150   \def\addlanguage#1{%
151     \global\advance\last@language\@ne
152     \ifnum\last@language<\@cc1vi
153       \else
154         \errmessage{No room for a new \string\language!}%
155       \fi
156     \global\chardef#1\last@language
157     \wlog{\string#1 = \string\language\the\last@language}}
158   \else
159     \countdef\last@language=19
```

```

160 \def\addlanguage{\alloc@9\language\chardef\@cclvi}
161 \fi
162 <</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 $\text{\LaTeX}2.09$. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

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

8 The Package File (\LaTeX , `babel.sty`)

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

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

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

8.1 base

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

```

163 (*package)
164 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
165 \ProvidesPackage{babel}[\langle date \rangle \langle version \rangle The Babel package]
166 \@ifpackagewith{babel}{debug}
167   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
168    \let\bbl@debug\@firstofone}
169   {\providecommand\bbl@trace[1]{}%
170    \let\bbl@debug\gobble}
171 \ifx\bbl@switchflag\undefined % Prevent double input
172   \let\bbl@switchflag\relax
173   \input switch.def\relax
174 \fi
175 <<Load patterns in luatex>>
176 <<Basic macros>>
177 \def\AfterBabelLanguage#1{%
178   \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.

```

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

```



```

183     \begingroup
184     \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
185     \wlog{<*languages>}%
186     \bbl@languages
187     \wlog{</languages>}%
188     \endgroup}{}
189 \endgroup
190 \def\bbl@elt#1#2#3#4{%
191   \ifnum#2=\z@
192     \gdef\bbl@nulllanguage{#1}%
193     \def\bbl@elt##1##2##3##4{%
194       \fi}%
195   \bbl@languages
196 \fi
197 \ifodd\bbl@engine
198   \let\bbl@tempa\relax
199   \@ifpackagewith{babel}{bidi=basic}%
200   {\def\bbl@tempa{basic}}%
201   {\@ifpackagewith{babel}{bidi=basic-r}%
202     {\def\bbl@tempa{basic-r}}%
203   {}}
204 \ifx\bbl@tempa\relax\else
205   \let\bbl@beforeforeign\leavevmode
206   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
207   \RequirePackage{luatexbase}%
208   \directlua{
209     require('babel-bidi.lua')
210     require('babel-bidi-\bbl@tempa.lua')
211     luatexbase.add_to_callback('pre_linebreak_filter',
212       Babel.pre_otfload_v,
213       'Babel.pre_otfload_v',
214     luatexbase.priority_in_callback('pre_linebreak_filter',
215       'luaotfload.node_processor') or nil)
216     luatexbase.add_to_callback('hpack_filter',
217       Babel.pre_otfload_h,
218       'Babel.pre_otfload_h',
219     luatexbase.priority_in_callback('hpack_filter',
220       'luaotfload.node_processor') or nil)
221   }
222 \fi
223 \fi

```

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

```

224 \bbl@trace{Defining option 'base'}
225 \@ifpackagewith{babel}{base}{%
226   \ifx\directlua\undefined
227     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
228   \else
229     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
230   \fi
231   \DeclareOption{base}{}%
232   \DeclareOption{showlanguages}{}%
233   \ProcessOptions
234   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
235   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
236   \global\let@ifl@ter@\@ifl@ter
237   \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter@ifl@ter@@}%
238   \endinput}{}%

```

8.2 key=value options and other general option

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

```
239 \bbl@trace{key=value and another general options}
240 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
241 \def\bbl@tempb#1.#2{%
242   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
243 \def\bbl@tempd#1.#2\@nnil{%
244   \ifx\@empty#2%
245     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
246   \else
247     \in@{=}{#1}\ifin@
248     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
249   \else
250     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
251   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
252   \fi
253 \fi}
254 \let\bbl@tempc\@empty
255 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
256 \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.

```
257 \DeclareOption{KeepShorthandsActive}{}
258 \DeclareOption{activeacute}{}
259 \DeclareOption{activegrave}{}
260 \DeclareOption{debug}{}
261 \DeclareOption{noconfigs}{}
262 \DeclareOption{showlanguages}{}
263 \DeclareOption{silent}{}
264 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
265 <<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.

```
266 \let\bbl@opt@shorthands\@nnil
267 \let\bbl@opt@config\@nnil
268 \let\bbl@opt@main\@nnil
269 \let\bbl@opt@headfoot\@nnil
270 \let\bbl@opt@layout\@nnil
```

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

```
271 \def\bbl@tempa#1=#2\bbl@tempa{%
272   \bbl@csarg\ifx{opt@#1}\@nnil
273     \bbl@csarg\edef{opt@#1}{#2}%
274   \else
275     \bbl@error{%
276       Bad option `#1=#2'. Either you have misspelled the\%
```

```

277     key or there is a previous setting of `#1'}`%
278     Valid keys are `shorthands', `config', `strings', `main', \\%
279     `headfoot', `safe', `math', among others.}
280 \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.

```

281 \let\bbl@language@opts\@empty
282 \DeclareOption*{%
283   \bbl@xin@{\string=}{\CurrentOption}%
284   \ifin@
285     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
286   \else
287     \bbl@add@list\bbl@language@opts{\CurrentOption}%
288   \fi}

```

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

```

289 \ProcessOptions*

```

8.3 Conditional loading of shorthands

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

```

290 \bbl@trace{Conditional loading of shorthands}
291 \def\bbl@sh@string#1{%
292   \ifx#1\@empty\else
293     \ifx#1t\string-%
294     \else\ifx#1c\string,%
295     \else\string#1%
296   \fi\fi
297   \expandafter\bbl@sh@string
298 \fi}
299 \ifx\bbl@opt@shorthands\@nnil
300 \def\bbl@ifshorthand#1#2#3{#2}%
301 \else\ifx\bbl@opt@shorthands\@empty
302 \def\bbl@ifshorthand#1#2#3{#3}%
303 \else

```

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

```

304 \def\bbl@ifshorthand#1{%
305   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
306   \ifin@
307     \expandafter\@firstoftwo
308   \else
309     \expandafter\@secondoftwo
310   \fi}

```

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

```

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

```

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

```
313 \bbl@ifshorthand{'}%
314   {\PassOptionsToPackage{activeacute}{babel}}{}
315 \bbl@ifshorthand{`}%
316   {\PassOptionsToPackage{activegrave}{babel}}{}
317 \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.

```
318 \ifx\bbl@opt@headfoot\@nnil\else
319   \g@addto@macro\@resetactivechars{%
320     \set@typeset@protect
321     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
322     \let\protect\noexpand}
323 \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.

```
324 \ifx\bbl@opt@safe\undefined
325   \def\bbl@opt@safe{BR}
326 \fi
327 \ifx\bbl@opt@main\@nnil\else
328   \edef\bbl@language@opts{%
329     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
330     \bbl@opt@main}
331 \fi
```

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

```
332 \bbl@trace{Defining IfBabelLayout}
333 \ifx\bbl@opt@layout\@nnil
334   \newcommand\IfBabelLayout[3]{#3}%
335 \else
336   \newcommand\IfBabelLayout[1]{%
337     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
338     \ifin@
339       \expandafter\@firstoftwo
340     \else
341       \expandafter\@secondoftwo
342     \fi}
343 \fi
```

8.4 Language options

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

```
344 \bbl@trace{Language options}
345 \let\bbl@afterlang\relax
346 \let\BabelModifiers\relax
347 \let\bbl@loaded\@empty
348 \def\bbl@load@language#1{%
349   \InputIfFileExists{#1.ldf}%
350   {\edef\bbl@loaded{\CurrentOption
351     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
```

```

352 \expandafter\let\expandafter\bbl@afterlang
353 \csname\CurrentOption.ldf-h@k\endcsname
354 \expandafter\let\expandafter\BabelModifiers
355 \csname bbl@mod@\CurrentOption\endcsname}%
356 {\bbl@error{%
357   Unknown option '\CurrentOption'. Either you misspelled it\\%
358   or the language definition file \CurrentOption.ldf was not found}{%
359   Valid options are: shorthands=, KeepShorthandsActive,\\%
360   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
361   headfoot=, strings=, config=, hyphenmap=, or a language name.}}

```

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

```

362 \def\bbl@try@load@lang#1#2#3{%
363   \IfFileExists{\CurrentOption.ldf}%
364   {\bbl@load@language{\CurrentOption}}%
365   {#1\bbl@load@language{#2}#3}}
366 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}%}
367 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}%}
368 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}%}
369 \DeclareOption{hebrew}{%
370   \input{rlbabel.def}%
371   \bbl@load@language{hebrew}}
372 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}%}
373 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}%}
374 \DeclareOption{nyorsk}{\bbl@try@load@lang{}{norsk}{}%}
375 \DeclareOption{polutonikogreek}{%
376   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}%}
377 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}%}
378 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}%}
379 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}%}
380 \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.

```

381 \ifx\bbl@opt@config@nnil
382   \@ifpackagewith{babel}{noconfigs}{}%
383   {\InputIfFileExists{bblopts.cfg}%
384     {\typeout{*****^J%
385               * Local config file bblopts.cfg used^^J%
386               *}}%
387     {}}%
388 \else
389   \InputIfFileExists{\bbl@opt@config.cfg}%
390   {\typeout{*****^J%
391             * Local config file \bbl@opt@config.cfg used^^J%
392             *}}%
393   {\bbl@error{%
394     Local config file '\bbl@opt@config.cfg' not found}{%
395     Perhaps you misspelled it.}}%
396 \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.

```

397 \bbl@for\bbl@tempa\bbl@language@opts{%
398   \bbl@ifunset{ds@\bbl@tempa}%
399   {\edef\bbl@tempb{%
400     \noexpand\DeclareOption
401     {\bbl@tempa}%
402     {\noexpand\bbl@load@language{\bbl@tempa}}}%
403   \bbl@tempb}%
404   \@empty}

```

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

```

405 \bbl@foreach\@classoptionslist{%
406   \bbl@ifunset{ds@#1}%
407   {\IfFileExists{#1.ldf}%
408    {\DeclareOption{#1}{\bbl@load@language{#1}}}%
409    {}}%
410   {}}

```

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

```

411 \ifx\bbl@opt@main\@nnil\else
412   \expandafter
413   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
414   \DeclareOption{\bbl@opt@main}{}
415 \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):

```

416 \def\AfterBabelLanguage#1{%
417   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
418 \DeclareOption*{}
419 \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`.

```

420 \ifx\bbl@opt@main\@nnil
421   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
422   \let\bbl@tempc\@empty
423   \bbl@for\bbl@tempb\bbl@tempa{%
424     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
425     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
426   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
427   \expandafter\bbl@tempa\bbl@loaded,\@nnil
428   \ifx\bbl@tempb\bbl@tempc\else
429     \bbl@warning{%
430       Last declared language option is `'\bbl@tempc',\%
431       but the last processed one was `'\bbl@tempb'.\%
432       The main language cannot be set as both a global\%
433       and a package option. Use `main=\bbl@tempc' as\%
434       option. Reported}%
435   \fi
436 \else
437   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}

```

```

438 \ExecuteOptions{\bbl@opt@main}
439 \DeclareOption*{}
440 \ProcessOptions*
441 \fi
442 \def\AfterBabelLanguage{%
443   \bbl@error
444   {Too late for \string\AfterBabelLanguage}%
445   {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.

```

446 \ifx\bbl@main@language\undefined
447   \bbl@info{%
448     You haven't specified a language. I'll use 'nil'\%
449     as the main language. Reported}
450   \bbl@load@language{nil}
451 \fi
452 \endpackage
453 \*core

```

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

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `babel.def` contains most of the code, while `switch.def` defines the language switching commands; both can be read at run time. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns (by default, it also inputs `switch.def`, for “historical reasons”, but it is not necessary). When `babel.def` is loaded it checks if the current version of `switch.def` is in the format; if not, it is loaded. A further file, `babel.sty`, contains \LaTeX -specific stuff. Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don’t load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

9.1 Tools

```

454 \ifx\ldf@quit\undefined
455 \else
456   \expandafter\endinput
457 \fi
458 \langle\langle Make sure ProvidesFile is defined \rangle\rangle
459 \ProvidesFile{babel.def}[\langle\langle date \rangle\rangle] [\langle\langle version \rangle\rangle] Babel common definitions]
460 \langle\langle Load macros for plain if not LaTeX \rangle\rangle

```

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

```

461 \ifx\bbl@ifshorthand\undefined
462   \let\bbl@opt@shorthands\@nil

```

```

463 \def\bbl@ifshorthand#1#2#3{#2}%
464 \let\bbl@language@opts\@empty
465 \ifx\babeloptionstrings\@undefined
466   \let\bbl@opt@strings\@nnil
467   \else
468     \let\bbl@opt@strings\babeloptionstrings
469   \fi
470 \def\BabelStringsDefault{generic}
471 \def\bbl@tempa{normal}
472 \ifx\babeloptionmath\bbl@tempa
473   \def\bbl@mathnormal{\noexpand\textormath}
474   \fi
475 \def\AfterBabelLanguage#1#2{}
476 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
477 \let\bbl@afterlang\relax
478 \def\bbl@opt@safe{BR}
479 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
480 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
481 \fi

And continue.
482 \ifx\bbl@switchflag\@undefined % Prevent double input
483   \let\bbl@switchflag\relax
484   \input switch.def\relax
485 \fi
486 \bbl@trace{Compatibility with language.def}
487 \ifx\bbl@languages\@undefined
488   \ifx\directlua\@undefined
489     \openin1 = language.def
490     \ifeof1
491       \closein1
492       \message{I couldn't find the file language.def}
493     \else
494       \closein1
495       \begingroup
496         \def\addlanguage#1#2#3#4#5{%
497           \expandafter\ifx\csname lang@#1\endcsname\relax\else
498             \global\expandafter\let\csname l@#1\endcsname
499               \csname lang@#1\endcsname
500           \fi}%
501         \def\uselanguage#1{}\fi}%
502         \input language.def
503       \endgroup
504     \fi
505   \fi
506   \chardef\l@english\z@
507 \fi
508 <<Load patterns in luatex>>
509 <<Basic macros>>

```

\addto For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro \addto is introduced. It takes two arguments, a *<control sequence>* and T_EX-code to be added to the *<control sequence>*. If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to \relax, in which case a circular definition results. The net result is a stack overflow. Otherwise the replacement text for the *<control sequence>* is expanded and stored in a token register, together with the T_EX-code to be added. Finally the *<control sequence>* is redefined, using the contents of the token register.


```

510 \def\addto#1#2{%
511   \ifx#1\undefined
512     \def#1{#2}%
513   \else
514     \ifx#1\relax
515       \def#1{#2}%
516     \else
517       {\toks@\expandafter{#1#2}%
518        \xdef#1{\the\toks@}}%
519     \fi
520 \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.

```

521 \def\bbl@withactive#1#2{%
522   \begingroup
523   \lccode`~=#2\relax
524   \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@. . .`

```

525 \def\bbl@redefine#1{%
526   \edef\bbl@tempa{\bbl@stripslash#1}%
527   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
528   \expandafter\def\csname\bbl@tempa\endcsname}

```

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

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

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

```

530 \def\bbl@redefine@long#1{%
531   \edef\bbl@tempa{\bbl@stripslash#1}%
532   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
533   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
534 \@onlypreamble\bbl@redefine@long

```

`\bbl@redefineroobust` For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo_`. So it is necessary to check whether `\foo_` exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define `\foo_`.

```

535 \def\bbl@redefineroobust#1{%
536   \edef\bbl@tempa{\bbl@stripslash#1}%
537   \bbl@ifunset{\bbl@tempa\space}%
538   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
539    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
540   {\bbl@exp{\let\<org@\bbl@tempa\<\bbl@tempa\space>}}%
541   \@namedef{\bbl@tempa\space}}

```

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

```
542 \@onlypreamble\bbl@redefineroobust
```

9.2 Hooks

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

```

543 \bbl@trace{Hooks}
544 \def\AddBabelHook#1#2{%
545   \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}{}%
546   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
547   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
548   \bbl@ifunset{bbl@ev@#1@#2}%
549     {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}}%
550     \bbl@csarg\newcommand}%
551     {\bbl@csarg\let{ev@#1@#2}\relax
552     \bbl@csarg\newcommand}%
553     {ev@#1@#2}[\bbl@tempb]}
554 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
555 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
556 \def\bbl@usehooks#1#2{%
557   \def\bbl@elt##1{%
558     \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}%
559   \@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).

```

560 \def\bbl@evargs{,% don't delete the comma
561   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
562   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
563   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
564   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 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.

```

565 \bbl@trace{Defining babelensure}
566 \newcommand\babelensure[2][{}]{% TODO - revise test files
567   \AddBabelHook{babel-ensure}{afterextras}{%
568     \ifcase\bbl@select@type
569       \@nameuse{bbl@e@\language\name}%
570     \fi}%
571   \begingroup
572     \let\bbl@ens@include\@empty
573     \let\bbl@ens@exclude\@empty
574     \def\bbl@ens@fontenc{\relax}%
575     \def\bbl@tempb##1{%
576       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%

```

```

577 \edef\bb1@tempa{\bb1@tempb#1\@empty}%
578 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ens@##1}{##2}}%
579 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
580 \def\bb1@tempc{\bb1@ensure}%
581 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
582 \expandafter{\bb1@ens@include}}%
583 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
584 \expandafter{\bb1@ens@exclude}}%
585 \toks@\expandafter{\bb1@tempc}%
586 \bb1@exp{%
587 \endgroup
588 \def\<bb1@e@#2>{\the\toks@{\bb1@ens@fontenc}}%
589 \def\bb1@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
590 \def\bb1@tempb##1{% elt for (excluding) \bb1@captionslist list
591 \ifx##1\@empty\else
592 \in@{##1}{#2}%
593 \ifin\@else
594 \bb1@ifunset{bb1@ensure@\languagename}%
595 {\bb1@exp{%
596 \\\DeclareRobustCommand\<bb1@ensure@\languagename>[1]{%
597 \\\foreignlanguage{\languagename}%
598 {\ifx\relax#3\else
599 \\\fontencoding{#3}\selectfont
600 \fi
601 #####1}}}%
602 }%
603 \toks@\expandafter{##1}%
604 \edef##1{%
605 \bb1@csarg\noexpand{ensure@\languagename}%
606 {\the\toks@}}%
607 \fi
608 \expandafter\bb1@tempb
609 \fi}%
610 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
611 \def\bb1@tempa##1{% elt for include list
612 \ifx##1\@empty\else
613 \bb1@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
614 \ifin\@else
615 \bb1@tempb##1\@empty
616 \fi
617 \expandafter\bb1@tempa
618 \fi}%
619 \bb1@tempa#1\@empty}
620 \def\bb1@captionslist{%
621 \prefacename\refname\abstractname\bibname\chaptername\appendixname
622 \contentsname\listfigurename\listtablename\indexname\figurename
623 \tablename\partname\enclname\ccname\headtoname\pagename\seename
624 \alsoname\proofname\glossaryname}

```

9.3 Setting up language files

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

At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save

its name as the last called option, even if not loaded.

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

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

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

Finally we check \originalTeX.

```
625 \bbl@trace{Macros for setting language files up}
626 \def\bbl@ldfinit{%
627   \let\bbl@screset\@empty
628   \let\BabelStrings\bbl@opt@string
629   \let\BabelOptions\@empty
630   \let\BabelLanguages\relax
631   \ifx\originalTeX\@undefined
632     \let\originalTeX\@empty
633   \else
634     \originalTeX
635   \fi}
636 \def\LdfInit#1#2{%
637   \chardef\atcatcode=\catcode`\@
638   \catcode`\@=11\relax
639   \chardef\eqcatcode=\catcode`\=
640   \catcode`\>=12\relax
641   \expandafter\if\expandafter\@backslashchar
642     \expandafter\@car\string#2\@nil
643     \ifx#2\@undefined\else
644       \ldf@quit{#1}%
645     \fi
646   \else
647     \expandafter\ifx\csname#2\endcsname\relax\else
648       \ldf@quit{#1}%
649     \fi
650   \fi
651   \bbl@ldfinit}
```

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

```
652 \def\ldf@quit#1{%
653   \expandafter\main@language\expandafter{#1}%
654   \catcode`\@=\atcatcode \let\atcatcode\relax
655   \catcode`\>=\eqcatcode \let\eqcatcode\relax
656   \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.

```
657 \def\bbl@afterldf#1{%
658   \bbl@afterlang
659   \let\bbl@afterlang\relax
```

```

660 \let\BabelModifiers\relax
661 \let\bbl@screset\relax}%
662 \def\ldf@finish#1{%
663 \loadlocalcfg{#1}%
664 \bbl@afterldf{#1}%
665 \expandafter\main@language\expandafter{#1}%
666 \catcode`\@=\atcatcode \let\atcatcode\relax
667 \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 .

```

668 \@onlypreamble\LdfInit
669 \@onlypreamble\ldf@quit
670 \@onlypreamble\ldf@finish

```

`\main@language` This command should be used in the various language definition files. It stores its argument in `\bbl@main@language`; to be used to switch to the correct language at the beginning of the document.

```

671 \def\main@language#1{%
672 \def\bbl@main@language{#1}%
673 \let\languagename\bbl@main@language
674 \bbl@patterns{\languagename}}

```

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

```

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

```

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

```

678 \def\select@language@x#1{%
679 \ifcase\bbl@select@type
680 \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
681 \else
682 \select@language{#1}%
683 \fi}

```

9.4 Shorthands

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

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

```

684 \bbl@trace{Shorhands}
685 \def\bbl@add@special#1{% 1:a macro like "\, \?, etc.
686 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
687 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
688 \ifx\nfss@catcodes\undefined\else % TODO - same for above
689 \begingroup
690 \catcode`#1\active
691 \nfss@catcodes
692 \ifnum\catcode`#1=\active

```

```

693     \endgroup
694     \bbl@add\nfss@catcodes{\@makeother#1}%
695     \else
696     \endgroup
697     \fi
698 \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.

```

699 \def\bbl@remove@special#1{%
700   \begingroup
701   \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
702     \else\noexpand##1\noexpand##2\fi}%
703   \def\do{\x\do}%
704   \def\@makeother{\x\@makeother}%
705   \edef\x{\endgroup
706     \def\noexpand\dospecials{\dospecials}%
707     \expandafter\ifx\csname @sanitize\endcsname\relax\else
708       \def\noexpand\@sanitize{\@sanitize}%
709     \fi}%
710   \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).

```

711 \def\bbl@active@def#1#2#3#4{%
712   \@namedef{#3#1}{%
713     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
714       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
715     \else
716       \bbl@afterfi\csname#2@sh@#1\endcsname
717     \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.

```

718   \long\@namedef{#3@arg#1}##1{%
719     \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
720       \bbl@afterelse\csname#4#1\endcsname##1%
721     \else

```

```

722     \bbl@afterfi\csname#2@sh#1@\string##1@endcsname
723     \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.

```

724 \def\initiate@active@char#1{%
725   \bbl@ifunset{active@char\string#1}%
726   {\bbl@withactive
727     {\expandafter\@initiate@active@char\expandafter}\string#1#1}%
728   {}}

```

The very first thing to do is saving the original catcode and the original definition, even if not active, which is possible (undefined characters require a special treatment to avoid making them \relax).

```

729 \def\@initiate@active@char#1#2#3{%
730   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
731   \ifx#1\@undefined
732     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
733   \else
734     \bbl@csarg\let{oridef@@#2}#1%
735     \bbl@csarg\edef{oridef@#2}{%
736       \let\noexpand#1%
737       \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
738   \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).

```

739   \ifx#1#3\relax
740     \expandafter\let\csname normal@char#2@endcsname#3%
741   \else
742     \bbl@info{Making #2 an active character}%
743     \ifnum\mathcode`#2="8000
744       \@namedef{normal@char#2}{%
745         \textormath{#3}{\csname bbl@oridef@@#2@endcsname}}%
746     \else
747       \@namedef{normal@char#2}{#3}%
748   \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).

```

749   \bbl@restoreactive{#2}%
750   \AtBeginDocument{%
751     \catcode`#2\active
752     \if@filesw
753       \immediate\write\@mainaux{\catcode`\string#2\active}%
754     \fi}%
755   \expandafter\bbl@add@special\csname#2@endcsname
756   \catcode`#2\active

```

757 \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⟩`).

```
758 \let\bbl@tempa\@firstoftwo
759 \if\string^#2%
760   \def\bbl@tempa{\noexpand\textormath}%
761 \else
762   \ifx\bbl@mathnormal\@undefined\else
763     \let\bbl@tempa\bbl@mathnormal
764   \fi
765 \fi
766 \expandafter\edef\csname active@char#2\endcsname{%
767   \bbl@tempa
768     {\noexpand\if@safe@actives
769       \noexpand\expandafter
770       \expandafter\noexpand\csname normal@char#2\endcsname
771       \noexpand\else
772         \noexpand\expandafter
773         \expandafter\noexpand\csname bbl@doactive#2\endcsname
774         \noexpand\fi}%
775   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
776 \bbl@csarg\edef{doactive#2}{%
777   \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

$$\backslash\active@prefix\langle char\rangle\backslash\normal@char\langle char\rangle$$

(where `\active@char⟨char⟩` is *one* control sequence!).

```
778 \bbl@csarg\edef{active@#2}{%
779   \noexpand\active@prefix\noexpand#1%
780   \expandafter\noexpand\csname active@char#2\endcsname}%
781 \bbl@csarg\edef{normal@#2}{%
782   \noexpand\active@prefix\noexpand#1%
783   \expandafter\noexpand\csname normal@char#2\endcsname}%
784 \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.

```
785 \bbl@active@def#2\user@group{user@active}{language@active}%
786 \bbl@active@def#2\language@group{language@active}{system@active}%
787 \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.

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

```
792 \if\string'#2%
793   \let\prim@s\bb1@prim@s
794   \let\active@math@prime#1%
795 \fi
796 \bb1@usehooks{initiateactive}{{#1}{#2}{#3}}
```

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

```
797 <<{*More package options}>> ≡
798 \DeclareOption{math=active}{}
799 \DeclareOption{math=normal}{\def\bb1@mathnormal{\noexpand\textormath}}
800 <</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.

```
801 \@ifpackagewith{babel}{KeepShorthandsActive}%
802   {\let\bb1@restoreactive\@gobble}%
803   {\def\bb1@restoreactive#1{%
804     \bb1@exp{%
805       \\\AfterBabelLanguage\\CurrentOption
806       {\catcode`#1=\the\catcode`#1\relax}%
807       \\\AtEndOfPackage
808       {\catcode`#1=\the\catcode`#1\relax}}}%
809   \AtEndOfPackage{\let\bb1@restoreactive\@gobble}}
```

`\bb1@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 `\bb1@firstcs` or `\bb1@scndcs`. Hence two more arguments need to follow it.

```
810 \def\bb1@sh@select#1#2{%
811   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
812     \bb1@afterelse\bb1@scndcs
813   \else
814     \bb1@afterfi\csname#1@sh@#2@sel\endcsname
815   \fi}
```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`.

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

When `\protect` is set to `\@unexpandable@protect` we make sure that the active character is also *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).

```
819   \ifx\protect\@unexpandable@protect
820     \noexpand#1%
```

```

821 \else
822 \protect#1%
823 \fi
824 \expandafter\@gobble
825 \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⟩`.

```

826 \newif\if@safe@actives
827 \@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.

```

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

```

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

```

829 \def\bbl@activate#1{%
830 \bbl@withactive{\expandafter\let\expandafter}#1%
831 \csname bbl@active@\string#1\endcsname}
832 \def\bbl@deactivate#1{%
833 \bbl@withactive{\expandafter\let\expandafter}#1%
834 \csname bbl@normal@\string#1\endcsname}

```

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

`\bbl@scndcs`

```

835 \def\bbl@firstcs#1#2{\csname#1\endcsname}
836 \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.

```

837 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
838 \def\@decl@short#1#2#3\@nil#4{%
839 \def\bbl@tempa{#3}%
840 \ifx\bbl@tempa\@empty
841 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
842 \bbl@ifunset{#1@sh@\string#2@}{}%
843 {\def\bbl@tempa{#4}%
844 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
845 \else
846 \bbl@info
847 {Redefining #1 shorthand \string#2\\
848 in language \CurrentOption}%
849 \fi}%
850 \@namedef{#1@sh@\string#2@}{#4}%
851 \else
852 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
853 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%

```

```

854     {\def\bbl@tempa{#4}%
855     \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
856     \else
857     \bbl@info
858     {Redefining #1 shorthand \string#2\string#3\%
859     in language \CurrentOption}%
860     \fi}%
861     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
862     \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.

```

863 \def\textormath{%
864   \ifmmode
865     \expandafter\@secondoftwo
866   \else
867     \expandafter\@firstoftwo
868   \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’.

```

869 \def\user@group{user}
870 \def\language@group{english}
871 \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.

```

872 \def\useshorthands{%
873   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
874 \def\bbl@usesh@s#1{%
875   \bbl@usesh@x
876   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
877   {#1}}
878 \def\bbl@usesh@x#1#2{%
879   \bbl@ifshorthand{#2}%
880   {\def\user@group{user}%
881    \initiate@active@char{#2}%
882    #1%
883    \bbl@activate{#2}}%
884   {\bbl@error
885    {Cannot declare a shorthand turned off (\string#2)}
886    {Sorry, but you cannot use shorthands which have been\%
887     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.

```

888 \def\user@language@group{user@\language@group}
889 \def\bbl@set@user@generic#1#2{%
890   \bbl@ifunset{user@generic@active#1}%

```

```

891   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
892   \bbl@active@def#1\user@group{user@generic@active}{language@active}%
893   \expandafter\edef\csname#2@sh@#1@@\endcsname{%
894   \expandafter\noexpand\csname normal@char#1\endcsname}%
895   \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
896   \expandafter\noexpand\csname user@active#1\endcsname}}%
897   \@empty}
898 \newcommand\defineshorthand[3][user]{%
899   \edef\bbl@tempa{\zap@space#1 \@empty}%
900   \bbl@for\bbl@tempb\bbl@tempa{%
901     \if*\expandafter\@car\bbl@tempb\@nil
902       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
903       \@expandtwoargs
904       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
905     \fi
906     \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.

```

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

```

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

```

908 \def\aliasshorthand#1#2{%
909   \bbl@ifshorthand{#2}%
910   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
911     \ifx\document\@notprerr
912       \@notshorthand{#2}%
913     \else
914       \initiate@active@char{#2}%

```

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

```

915     \expandafter\let\csname active@char\string#2\endcsname
916     \csname active@char\string#1\endcsname
917     \expandafter\let\csname normal@char\string#2\endcsname
918     \csname normal@char\string#1\endcsname
919     \bbl@activate{#2}%
920   \fi
921 \fi}%
922 {\bbl@error
923   {Cannot declare a shorthand turned off (\string#2)}
924   {Sorry, but you cannot use shorthands which have been\\%
925     turned off in the package options}}

```

`\@notshorthand`

```

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

```

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

```

933 \newcommand*\shorthandon[1]{\bbl@switch@sh@ne#1\@nnil}
934 \DeclareRobustCommand*\shorthandoff{%
935   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
936 \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.

```

937 \def\bbl@switch@sh#1#2{%
938   \ifx#2\@nnil\else
939     \bbl@ifunset{\bbl@active@\string#2}%
940     {\bbl@error
941       {I cannot switch '\string#2' on or off--not a shorthand}%
942       {This character is not a shorthand. Maybe you made\\
943         a typing mistake? I will ignore your instruction}}%
944     {\ifcase#1%
945       \catcode`#212\relax
946     \or
947       \catcode`#2\active
948     \or
949       \csname bbl@oricat@\string#2\endcsname
950       \csname bbl@oridef@\string#2\endcsname
951     \fi}%
952   \bbl@afterfi\bbl@switch@sh#1%
953 \fi}

```

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

```

954 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
955 \def\bbl@putsh#1{%
956   \bbl@ifunset{\bbl@active@\string#1}%
957   {\bbl@putsh@i#1\@empty\@nnil}%
958   {\csname bbl@active@\string#1\endcsname}}
959 \def\bbl@putsh@i#1#2\@nnil{%
960   \csname\languagename @sh@\string#1@%
961     \ifx\@empty#2\else\string#2@\fi\endcsname}
962 \ifx\bbl@opt@shorthands\@nnil\else
963   \let\bbl@s@initiate@active@char\initiate@active@char
964   \def\initiate@active@char#1{%
965     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
966 \let\bbl@s@switch@sh\bbl@switch@sh
967 \def\bbl@switch@sh#1#2{%
968   \ifx#2\@nnil\else
969     \bbl@afterfi
970     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
971   \fi}
972 \let\bbl@s@activate\bbl@activate
973 \def\bbl@activate#1{%
974   \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
975 \let\bbl@s@deactivate\bbl@deactivate
976 \def\bbl@deactivate#1{%

```

```

977 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
978 \fi

```

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

```

979 \def\bbl@prim@s{%
980 \prime\futurelet\@let@token\bbl@pr@m@s}
981 \def\bbl@if@primes#1#2{%
982 \ifx#1\@let@token
983 \expandafter\@firstoftwo
984 \else\ifx#2\@let@token
985 \bbl@afterelse\expandafter\@firstoftwo
986 \else
987 \bbl@afterfi\expandafter\@secondoftwo
988 \fi\fi}
989 \begingroup
990 \catcode\^=7 \catcode\*= \active \lccode\*= \^
991 \catcode\'=12 \catcode\"= \active \lccode\"= \'
992 \lowercase{%
993 \gdef\bbl@pr@m@s{%
994 \bbl@if@primes" "%
995 \pr@@@s
996 {\bbl@if@primes*\^ \pr@@@t\egroup}}
997 \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).

```

998 \initiate@active@char{~}
999 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1000 \bbl@activate{~}

```

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

```

1001 \expandafter\def\csname OT1dqpos\endcsname{127}
1002 \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

```

1003 \ifx\f@encoding\@undefined
1004 \def\f@encoding{OT1}
1005 \fi

```

9.5 Language attributes

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

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

```

1006 \bbl@trace{Language attributes}
1007 \newcommand\languageattribute[2]{%
1008   \def\bbl@tempc{#1}%
1009   \bbl@fixname\bbl@tempc
1010   \bbl@iflanguage\bbl@tempc{%
1011     \bbl@vforeach{#2}{%

```

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

```

1012     \if\bbl@known@attrs\@undefined
1013       \in@false
1014     \else

```

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

```

1015       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1016     \fi

```

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

```

1017     \ifin@
1018       \bbl@warning{%
1019         You have more than once selected the attribute '##1'\%
1020         for language #1. Reported}%
1021     \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.

```

1022     \bbl@exp{%
1023       \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1024     \edef\bbl@tempa{\bbl@tempc-##1}%
1025     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1026     {\csname\bbl@tempc @attr##1\endcsname}%
1027     {\@attrerr{\bbl@tempc}{##1}}%
1028   \fi}}

```

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

```

1029 \@onlypreamble\languageattribute

```

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

```

1030 \newcommand*{\@attrerr}[2]{%
1031   \bbl@error
1032   {The attribute #2 is unknown for language #1.}%
1033   {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}`.

```

1034 \def\bbl@declare@ttribute#1#2#3{%
1035   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1036   \ifin@
1037     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1038   \fi
1039   \bbl@add@list\bbl@attributes{#1-#2}%
1040   \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret T_EX 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.

```
1041 \def\bbl@ifattributeset#1#2#3#4{%
```

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

```
1042 \ifx\bbl@known@attrs\@undefined
```

```
1043 \in@false
```

```
1044 \else
```

The we need to check the list of known attributes.

```
1045 \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
```

```
1046 \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`'.

```
1047 \ifin@
```

```
1048 \bbl@afterelse#3%
```

```
1049 \else
```

```
1050 \bbl@afterfi#4%
```

```
1051 \fi
```

```
1052 }
```

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

```
1053 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1054 \let\bbl@tempa\@secondoftwo
```

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

```
1055 \bbl@loopx\bbl@tempb{#2}{%
```

```
1056 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
```

```
1057 \ifin@
```

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

```
1058 \let\bbl@tempa\@firstoftwo
```

```
1059 \else
```

```
1060 \fi}%
```

Finally we execute `\bbl@tempa`.

```
1061 \bbl@tempa
```

```
1062 }
```

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

```
1063 \def\bbl@clear@ttribs{%
```

```
1064 \ifx\bbl@attributes\@undefined\else
```

```
1065 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
```

```
1066 \expandafter\bbl@clear@ttrib\bbl@tempa.
```

```
1067 }%
```

```
1068 \let\bbl@attributes\@undefined
```

```
1069 \fi}
```

```
1070 \def\bbl@clear@ttrib#1-#2.{%
```

```
1071 \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
```

```
1072 \AtBeginDocument{\bbl@clear@ttribs}
```


9.6 Support for saving macro definitions

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

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.
`\babel@beginsave` 1073 `\bbl@trace{Macros for saving definitions}`
1074 `\def\babel@beginsave{\babel@savecnt\z@}`

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

```
1075 \newcount\babel@savecnt
1076 \babel@beginsave
```

`\babel@save` The macro `\babel@save⟨curname⟩` saves the current meaning of the control sequence `⟨curname⟩` to `\originalTeX`³². 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.

```
1077 \def\babel@save#1{%
1078   \expandafter\let\curname\babel@number\babel@savecnt\endcurname#1\relax
1079   \toks@\expandafter{\originalTeX\let#1=}%
1080   \bbl@exp{%
1081     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1082   \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.

```
1083 \def\babel@savevariable#1{%
1084   \toks@\expandafter{\originalTeX #1}%
1085   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```
1086 \def\bbl@frenchspacing{%
1087   \ifnum\the\sffcode`\.=@m
1088     \let\bbl@nonfrenchspacing\relax
1089   \else
1090     \frenchspacing
1091     \let\bbl@nonfrenchspacing\nonfrenchspacing
1092   \fi}
1093 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

9.7 Short tags

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

```
1094 \bbl@trace{Short tags}
1095 \def\babeltags#1{%
```

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

```

1096 \edef\bbl@tempa{\zap@space#1 \@empty}%
1097 \def\bbl@tempb##1=##2\@@{%
1098   \edef\bbl@tempc{%
1099     \noexpand\newcommand
1100     \expandafter\noexpand\csname ##1\endcsname{%
1101       \noexpand\protect
1102       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1103     \noexpand\newcommand
1104     \expandafter\noexpand\csname text##1\endcsname{%
1105       \noexpand\foreignlanguage{##2}}
1106     \bbl@tempc}%
1107 \bbl@for\bbl@tempa\bbl@tempa{%
1108   \expandafter\bbl@tempb\bbl@tempa\@@}}

```

9.8 Hyphens

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

```

1109 \bbl@trace{Hyphens}
1110 \@onlypreamble\babelhyphenation
1111 \AtEndOfPackage{%
1112   \newcommand\babelhyphenation[2][\@empty]{%
1113     \ifx\bbl@hyphenation@\relax
1114       \let\bbl@hyphenation@\@empty
1115     \fi
1116     \ifx\bbl@hyphlist\@empty\else
1117       \bbl@warning{%
1118         You must not intermingle \string\selectlanguage\space and\%
1119         \string\babelhyphenation\space or some exceptions will not\%
1120         be taken into account. Reported}%
1121     \fi
1122     \ifx\@empty#1%
1123       \protected@edef\bbl@hyphenation@\{\bbl@hyphenation@\space#2}%
1124     \else
1125       \bbl@vforeach{#1}{%
1126         \def\bbl@tempa{##1}%
1127         \bbl@fixname\bbl@tempa
1128         \bbl@iflanguage\bbl@tempa{%
1129           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1130             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1131             \@empty
1132             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1133             #2}}}%
1134     \fi}}

```

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

```

1135 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
1136 \def\bbl@t@one{T1}
1137 \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.

³³ \TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1138 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1139 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
1140 \def\bb1@hyphen{%
1141   \ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i \empty}}
1142 \def\bb1@hyphen@i#1#2{%
1143   \bb1@ifunset{bb1@hy@#1#2\empty}%
1144   {\csname bb1@#1usehyphen\endcsname{\discretionary{#2}{#2}}}%
1145   {\csname bb1@hy@#1#2\empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more 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 discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

```

1146 \def\bb1@usehyphen#1{%
1147   \leavevmode
1148   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1149   \nobreak\hskip\z@skip}
1150 \def\bb1@@usehyphen#1{%
1151   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1152 \def\bb1@hyphenchar{%
1153   \ifnum\hyphenchar\font=\m@ne
1154     \babelnullhyphen
1155   \else
1156     \char\hyphenchar\font
1157   \fi}

```

Finally, we define the hyphen “types”. Their names will not change, so you may use them in ldf’s. After a space, the \mbox in \bb1@hy@nobreak is redundant.

```

1158 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1159 \def\bb1@hy@@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1160 \def\bb1@hy@hard{\bb1@usehyphen\bb1@hyphenchar}
1161 \def\bb1@hy@@hard{\bb1@@usehyphen\bb1@hyphenchar}
1162 \def\bb1@hy@nobreak{\bb1@usehyphen{\mbox{\bb1@hyphenchar}}}
1163 \def\bb1@hy@@nobreak{\mbox{\bb1@hyphenchar}}
1164 \def\bb1@hy@repeat{%
1165   \bb1@usehyphen{%
1166     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
1167 \def\bb1@hy@@repeat{%
1168   \bb1@@usehyphen{%
1169     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
1170 \def\bb1@hy@empty{\hskip\z@skip}
1171 \def\bb1@hy@@empty{\discretionary{}{}{}}

```

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

```

1172 \def\bb1@disc#1#2{\nobreak\discretionary{#2-}{#1}\bb1@allowhyphens}

```

9.9 Multiencoding strings

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

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

```

1173 \bbl@trace{Multiencoding strings}
1174 \def\bbl@tglobal#1{\global\let#1#1}
1175 \def\bbl@recatcode#1{%
1176   \@tempcnta="7F
1177   \def\bbl@tempa{%
1178     \ifnum\@tempcnta>"FF\else
1179       \catcode\@tempcnta=#1\relax
1180       \advance\@tempcnta\@ne
1181       \expandafter\bbl@tempa
1182     \fi}%
1183   \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).

```

1184 \@ifpackagewith{babel}{nocase}%
1185   {\let\bbl@patchuclc\relax}%
1186   {\def\bbl@patchuclc{%
1187     \global\let\bbl@patchuclc\relax
1188     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1189     \gdef\bbl@uclc##1{%
1190       \let\bbl@encoded\bbl@encoded@uclc
1191       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1192       {##1}%
1193       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1194         \csname\languagename @bbl@uclc\endcsname}%
1195       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1196     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1197     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}%
1198 <<(*More package options)>> ≡
1199 \DeclareOption{nocase}{}
1200 <</More package options>>

```

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

```

1201 <<(*More package options)>> ≡
1202 \let\bbl@opt@strings\@nnil % accept strings=value
1203 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1204 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1205 \def\BabelStringsDefault{generic}
1206 <</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.

```
1207 \@onlypreamble\StartBabelCommands
```

```

1208 \def\StartBabelCommands{%
1209   \begingroup
1210   \bbl@recatcode{11}%
1211   <<Macros local to BabelCommands>>
1212   \def\bbl@provstring##1##2{%
1213     \providecommand##1{##2}%
1214     \bbl@toglobal##1}%
1215   \global\let\bbl@scafter\@empty
1216   \let\StartBabelCommands\bbl@startcmds
1217   \ifx\BabelLanguages\relax
1218     \let\BabelLanguages\CurrentOption
1219   \fi
1220   \begingroup
1221   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1222   \StartBabelCommands}
1223 \def\bbl@startcmds{%
1224   \ifx\bbl@screset\@nnil\else
1225     \bbl@usehooks{stopcommands}{}%
1226   \fi
1227   \endgroup
1228   \begingroup
1229   \@ifstar
1230     {\ifx\bbl@opt@strings\@nnil
1231       \let\bbl@opt@strings\BabelStringsDefault
1232     \fi
1233     \bbl@startcmds@i}%
1234   \bbl@startcmds@i}
1235 \def\bbl@startcmds@i#1#2{%
1236   \edef\bbl@L{\zap@space#1 \@empty}%
1237   \edef\bbl@G{\zap@space#2 \@empty}%
1238   \bbl@startcmds@ii}

```

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

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

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

```

1239 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1240   \let\SetString@gobbletwo
1241   \let\bbl@stringdef@gobbletwo
1242   \let\AfterBabelCommands@gobble
1243   \ifx\@empty#1%
1244     \def\bbl@sc@label{generic}%
1245     \def\bbl@encstring##1##2{%
1246       \ProvideTextCommandDefault##1{##2}%
1247       \bbl@toglobal##1%
1248       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1249     \let\bbl@sctest\in@true
1250   \else
1251     \let\bbl@sc@charset\space % <- zapped below
1252     \let\bbl@sc@fontenc\space % <- " "
1253     \def\bbl@tempa##1=##2\@nil{%
1254       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%

```

```

1255 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1256 \def\bbl@tempa##1 ##2{% space -> comma
1257   ##1%
1258   \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1259 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1260 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1261 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1262 \def\bbl@encstring##1##2{%
1263   \bbl@foreach\bbl@sc@fontenc{%
1264     \bbl@ifunset{T@###1}%
1265     {}%
1266     {\ProvideTextCommand##1{###1}{##2}%
1267     \bbl@tglobal##1%
1268     \expandafter
1269     \bbl@tglobal\csname###1\string##1\endcsname}}}%
1270 \def\bbl@sctest{%
1271   \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1272 \fi
1273 \ifx\bbl@opt@strings\@nil % ie, no strings key -> defaults
1274 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1275   \let\AfterBabelCommands\bbl@aftercmds
1276   \let\SetString\bbl@setstring
1277   \let\bbl@stringdef\bbl@encstring
1278 \else % ie, strings=value
1279 \bbl@sctest
1280 \ifin@
1281   \let\AfterBabelCommands\bbl@aftercmds
1282   \let\SetString\bbl@setstring
1283   \let\bbl@stringdef\bbl@provstring
1284 \fi\fi\fi
1285 \bbl@scswitch
1286 \ifx\bbl@G\@empty
1287   \def\SetString##1##2{%
1288     \bbl@error{Missing group for string \string##1}%
1289     {You must assign strings to some category, typically\\%
1290     captions or extras, but you set none}}%
1291 \fi
1292 \ifx\@empty#1%
1293   \bbl@usehooks{defaultcommands}{}%
1294 \else
1295   \@expandtwoargs
1296   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1297 \fi}

```

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

```

1298 \def\bbl@forlang#1##2{%
1299   \bbl@for#1\bbl@L{%
1300     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1301     \ifin@#2\relax\fi}}
1302 \def\bbl@scswitch{%
1303   \bbl@forlang\bbl@tempa{%

```

```

1304 \ifx\bb1@G\@empty\else
1305 \ifx\SetString\@gobb1etwo\else
1306 \edef\bb1@GL{\bb1@G\bb1@tempa}%
1307 \bb1@xin@{\bb1@GL,}{,\bb1@screset,}%
1308 \ifin@\else
1309 \global\expandafter\let\csname\bb1@GL\endcsname\@undefined
1310 \xdef\bb1@screset{\bb1@screset,\bb1@GL}%
1311 \fi
1312 \fi
1313 \fi}}
1314 \AtEndOfPackage{%
1315 \def\bb1@forlang#1#2{\bb1@for#1\bb1@L{\bb1@ifunset{date#1}{#2}}}%
1316 \let\bb1@scswitch\relax}
1317 \@onlypreamble\EndBabelCommands
1318 \def\EndBabelCommands{%
1319 \bb1@usehooks{stopcommands}{}%
1320 \endgroup
1321 \endgroup
1322 \bb1@scafter}

```

Now we define commands to be used inside \StartBabelCommands.

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

```

1323 \def\bb1@setstring#1#2{%
1324 \bb1@forlang\bb1@tempa{%
1325 \edef\bb1@LC{\bb1@tempa\bb1@stripslash#1}%
1326 \bb1@ifunset{\bb1@LC}% eg, \germanchaptername
1327 {\global\expandafter % TODO - con \bb1@exp ?
1328 \bb1@add\csname\bb1@G\bb1@tempa\expandafter\endcsname\expandafter
1329 {\expandafter\bb1@scset\expandafter#1\csname\bb1@LC\endcsname}}}%
1330 {}}%
1331 \def\BabelString{#2}%
1332 \bb1@usehooks{stringprocess}{}%
1333 \expandafter\bb1@stringdef
1334 \csname\bb1@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include \bb1@encoded for string to be expanded in case transformations. It is \relax by default, but in \MakeUppercase and \MakeLowercase its value is a modified expandable \@changed@cmd.

```

1335 \ifx\bb1@opt@strings\relax
1336 \def\bb1@scset#1#2{\def#1{\bb1@encoded#2}}
1337 \bb1@patchuclc
1338 \let\bb1@encoded\relax
1339 \def\bb1@encoded@uclc#1{%
1340 \@inmathwarn#1%
1341 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1342 \expandafter\ifx\csname ?\string#1\endcsname\relax
1343 \TextSymbolUnavailable#1%
1344 \else
1345 \csname ?\string#1\endcsname
1346 \fi
1347 \else
1348 \csname\cf@encoding\string#1\endcsname

```

```

1349   \fi}
1350 \else
1351   \def\bbl@scset#1#2{\def#1{#2}}
1352 \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.

```

1353 <<(*Macros local to BabelCommands)>> ≡
1354 \def\SetStringLoop##1##2{%
1355   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1356   \count@\z@
1357   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1358     \advance\count@\@ne
1359     \toks@\expandafter{\bbl@tempa}%
1360     \bbl@exp{%
1361       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1362       \count@=\the\count@\relax}}}%
1363 <</Macros local to BabelCommands>>

```

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

```

1364 \def\bbl@aftercmds#1{%
1365   \toks@\expandafter{\bbl@scafter#1}%
1366   \xdef\bbl@scafter{\the\toks@}}

```

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

```

1367 <<(*Macros local to BabelCommands)>> ≡
1368 \newcommand\SetCase[3][]{%
1369   \bbl@patchuclc
1370   \bbl@forlang\bbl@tempa{%
1371     \expandafter\bbl@encstring
1372     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1373     \expandafter\bbl@encstring
1374     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1375     \expandafter\bbl@encstring
1376     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1377 <</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.

```

1378 <<(*Macros local to BabelCommands)>> ≡
1379 \newcommand\SetHyphenMap[1]{%
1380   \bbl@forlang\bbl@tempa{%
1381     \expandafter\bbl@stringdef
1382     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1383 <</Macros local to BabelCommands>>

```

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

```

1384 \newcommand\BabelLower[2]{% one to one.
1385   \ifnum\lccode#1=#2\else
1386     \babel@savevariable{\lccode#1}%
1387     \lccode#1=#2\relax
1388   \fi}

```



```

1389 \newcommand\BabelLowerMM[4]{% many-to-many
1390   \@tempcnta=#1\relax
1391   \@tempcntb=#4\relax
1392   \def\bbl@tempa{%
1393     \ifnum\@tempcnta>#2\else
1394       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1395       \advance\@tempcnta#3\relax
1396       \advance\@tempcntb#3\relax
1397       \expandafter\bbl@tempa
1398     \fi}%
1399   \bbl@tempa}
1400 \newcommand\BabelLowerMO[4]{% many-to-one
1401   \@tempcnta=#1\relax
1402   \def\bbl@tempa{%
1403     \ifnum\@tempcnta>#2\else
1404       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1405       \advance\@tempcnta#3
1406       \expandafter\bbl@tempa
1407     \fi}%
1408   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1409 <<{*More package options}>> ≡
1410 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1411 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1412 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1413 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1414 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1415 <</More package options>>

```

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

```

1416 \AtEndOfPackage{%
1417   \ifx\bbl@opt@hyphenmap\undefined
1418     \bbl@xin@{,}{\bbl@language@opts}%
1419     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1420   \fi}

```

9.10 Macros common to a number of languages

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

```

1421 \bbl@trace{Macros related to glyphs}
1422 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1423   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1424   \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.

```

1425 \def\save@sf@q#1{\leavevmode
1426   \begingroup
1427   \edef\SF{\spacefactor\the\spacefactor}#1\SF
1428   \endgroup}

```

9.11 Making glyphs available

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

9.11.1 Quotation marks

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

```
1429 \ProvideTextCommand{\quotedblbase}{OT1}{%
1430 \save@sf@q{\set@low@box{\textquotedblright\}%
1431 \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
1432 \ProvideTextCommandDefault{\quotedblbase}{%
1433 \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1434 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1435 \save@sf@q{\set@low@box{\textquoteright\}%
1436 \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
1437 \ProvideTextCommandDefault{\quotesinglbase}{%
1438 \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
\guillemotright 1439 \ProvideTextCommand{\guillemotleft}{OT1}{%
1440 \ifmmode
1441 \ll
1442 \else
1443 \save@sf@q{\nobreak
1444 \raise.2ex\hbox{\scriptscriptstyle\ll}\bb1@allowhyphens}%
1445 \fi}
1446 \ProvideTextCommand{\guillemotright}{OT1}{%
1447 \ifmmode
1448 \gg
1449 \else
1450 \save@sf@q{\nobreak
1451 \raise.2ex\hbox{\scriptscriptstyle\gg}\bb1@allowhyphens}%
1452 \fi}
```

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

```
1453 \ProvideTextCommandDefault{\guillemotleft}{%
1454 \UseTextSymbol{OT1}{\guillemotleft}}
1455 \ProvideTextCommandDefault{\guillemotright}{%
1456 \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1457 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1458 \ifmmode
1459 <%
1460 \else
1461 \save@sf@q{\nobreak
1462 \raise.2ex\hbox{\scriptscriptstyle<}\bb1@allowhyphens}%
1463 \fi}
1464 \ProvideTextCommand{\guilsinglright}{OT1}{%
1465 \ifmmode
```

```

1466 >%
1467 \else
1468 \save@sfi@q{\nobreak
1469 \raise.2ex\hbox{\scriptscriptstyle>}\bbl@allowhyphens}%
1470 \fi}

```

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

```

1471 \ProvideTextCommandDefault{\guilsinglleft}{%
1472 \UseTextSymbol{OT1}{\guilsinglleft}}
1473 \ProvideTextCommandDefault{\guilsinglright}{%
1474 \UseTextSymbol{OT1}{\guilsinglright}}

```

9.11.2 Letters

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

```

1475 \DeclareTextCommand{\ij}{OT1}{%
1476 i\kern-0.02em\bbl@allowhyphens j}
1477 \DeclareTextCommand{\IJ}{OT1}{%
1478 I\kern-0.02em\bbl@allowhyphens J}
1479 \DeclareTextCommand{\ij}{T1}{\char188}
1480 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

1481 \ProvideTextCommandDefault{\ij}{%
1482 \UseTextSymbol{OT1}{\ij}}
1483 \ProvideTextCommandDefault{\IJ}{%
1484 \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

1485 \def\crrtic@{\hrule height0.1ex width0.3em}
1486 \def\crrtic@{\hrule height0.1ex width0.33em}
1487 \def\ddj@{%
1488 \setbox0\hbox{d}\dimen@=\ht0
1489 \advance\dimen@1ex
1490 \dimen@.45\dimen@
1491 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1492 \advance\dimen@ii.5ex
1493 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1494 \def\DDJ@{%
1495 \setbox0\hbox{D}\dimen@=.55\ht0
1496 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1497 \advance\dimen@ii.15ex % correction for the dash position
1498 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1499 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1500 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1501 %
1502 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1503 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

1504 \ProvideTextCommandDefault{\dj}{%
1505   \UseTextSymbol{OT1}{\dj}}
1506 \ProvideTextCommandDefault{\DJ}{%
1507   \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.

```

1508 \DeclareTextCommand{\SS}{OT1}{SS}
1509 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

9.11.3 Shorthands for quotation marks

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

\glq The ‘german’ single quotes.

```

\grq 1510 \ProvideTextCommandDefault{\glq}{%
1511   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

1512 \ProvideTextCommand{\grq}{T1}{%
1513   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1514 \ProvideTextCommand{\grq}{TU}{%
1515   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1516 \ProvideTextCommand{\grq}{OT1}{%
1517   \save@sf@q{\kern-.0125em
1518     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1519     \kern.07em\relax}}
1520 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 1521 \ProvideTextCommandDefault{\glqq}{%
1522   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1523 \ProvideTextCommand{\grqq}{T1}{%
1524   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1525 \ProvideTextCommand{\grqq}{TU}{%
1526   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1527 \ProvideTextCommand{\grqq}{OT1}{%
1528   \save@sf@q{\kern-.07em
1529     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1530     \kern.07em\relax}}
1531 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

\flq The ‘french’ single guillemets.

```

\frq 1532 \ProvideTextCommandDefault{\flq}{%
1533   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1534 \ProvideTextCommandDefault{\frq}{%
1535   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq The ‘french’ double guillemets.
\frqq

```
1536 \ProvideTextCommandDefault{\flqq}{%
1537 \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1538 \ProvideTextCommandDefault{\frqq}{%
1539 \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

9.11.4 Umlauts and tremas

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

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

```
1540 \def\uumlauthigh{%
1541 \def\bbl@umlauta##1{\leavevmode\bgroup%
1542 \expandafter\accent\csname\f@encoding dqpos\endcsname
1543 ##1\bbl@allowhyphens\egroup}%
1544 \let\bbl@umlaute\bbl@umlauta}
1545 \def\uumlautlow{%
1546 \def\bbl@umlauta{\protect\lower@umlaut}}
1547 \def\uumlautelowa{\protect\lower@umlaut}}
1548 \def\bbl@umlaute{\protect\lower@umlaut}}
1549 \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.

```
1550 \expandafter\ifx\csname U@D\endcsname\relax
1551 \csname newdimen\endcsname\U@D
1552 \fi
```

The following code fools T_EX'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.

```
1553 \def\lower@umlaut#1{%
1554 \leavevmode\bgroup
1555 \U@D 1ex%
1556 {\setbox\z@\hbox{%
1557 \expandafter\char\csname\f@encoding dqpos\endcsname}%
1558 \dimen@ -.45ex\advance\dimen@\ht\z@
1559 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1560 \expandafter\accent\csname\f@encoding dqpos\endcsname
1561 \fontdimen5\font\U@D #1%
1562 \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).

```

1563 \AtBeginDocument{%
1564 \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
1565 \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
1566 \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
1567 \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
1568 \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
1569 \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
1570 \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
1571 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
1572 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
1573 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
1574 \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}%
1575 }

```

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

```

1576 \ifx\l@english\@undefined
1577 \chardef\l@english\z@
1578 \fi
1579 \main@language{english}

```

9.12 Layout

Work in progress.

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

```

1580 \bbl@trace{Bidi layout}
1581 \providecommand\IfBabelLayout[3]{#3}%
1582 \newcommand\BabelPatchSection[1]{%
1583 \@ifundefined{#1}{%
1584 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1585 \@namedef{#1}{%
1586 \@ifstar{\bbl@presec@s{#1}}%
1587 {\@dblarg{\bbl@presec@x{#1}}}}%
1588 \def\bbl@presec@x#1[#2]#3{%
1589 \bbl@exp{%
1590 \\\select@language@x{\bbl@main@language}%
1591 \\\@nameuse{bbl@sspre@#1}%
1592 \\\@nameuse{bbl@ss@#1}%
1593 [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1594 {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
1595 \\\select@language@x{\languagename}}%
1596 \def\bbl@presec@s#1#2{%
1597 \bbl@exp{%
1598 \\\select@language@x{\bbl@main@language}%
1599 \\\@nameuse{bbl@sspre@#1}%
1600 \\\@nameuse{bbl@ss@#1}*%
1601 {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1602 \\\select@language@x{\languagename}}%
1603 \IfBabelLayout{sectioning}%
1604 {\BabelPatchSection{part}%
1605 \BabelPatchSection{chapter}%
1606 \BabelPatchSection{section}%
1607 \BabelPatchSection{subsection}}%

```

```

1608 \BabelPatchSection{subsubsection}%
1609 \BabelPatchSection{paragraph}%
1610 \BabelPatchSection{subparagraph}%
1611 \def\babel@toc#1{%
1612   \select@language@x{\bbl@main@language}}{}
1613 \IfBabelLayout{captions}%
1614 {\BabelPatchSection{caption}}{}

```

Now we load definition files for engines.

```

1615 \bbl@trace{Input engine specific macros}
1616 \ifcase\bbl@engine
1617 \input txtbabel.def
1618 \or
1619 \input luababel.def
1620 \or
1621 \input xebabel.def
1622 \fi

```

9.13 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.

```

1623 \bbl@trace{Creating languages and reading ini files}
1624 \newcommand\babelprovide[2][{}]{%
1625   \let\bbl@savelangname\languagename
1626   \def\languagename{#2}%
1627   \let\bbl@KVP@captions\@nil
1628   \let\bbl@KVP@import\@nil
1629   \let\bbl@KVP@main\@nil
1630   \let\bbl@KVP@script\@nil
1631   \let\bbl@KVP@language\@nil
1632   \let\bbl@KVP@dir\@nil
1633   \let\bbl@KVP@hyphenrules\@nil
1634   \let\bbl@KVP@mapfont\@nil
1635   \let\bbl@KVP@maparabic\@nil
1636   \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}% TODO - error handling
1637   \ifx\bbl@KVP@captions\@nil
1638     \let\bbl@KVP@captions\bbl@KVP@import
1639   \fi
1640   \bbl@ifunset{date#2}%
1641     {\bbl@provide@new{#2}}%
1642     {\bbl@ifblank{#1}%
1643       {\bbl@error
1644         {If you want to modify `#2' you must tell how in\\
1645         the optional argument. Currently there are three\\
1646         options: captions=lang-tag, hyphenrules=lang-list\\
1647         import=lang-tag}%
1648         {Use this macro as documented}}%
1649       {\bbl@provide@renew{#2}}}%
1650   \bbl@exp{\babelensure[exclude=\\today]{#2}}%
1651   \bbl@ifunset{bbl@ensure@\languagename}%
1652     {\bbl@exp{%
1653       \\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1654         \\foreignlanguage{\languagename}%
1655         {###1}}}%
1656     }%

```

```

1657 \ifx\bb1@KVP@script\@nil\else
1658   \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
1659 \fi
1660 \ifx\bb1@KVP@language\@nil\else
1661   \bb1@csarg\edef{lname@#2}{\bb1@KVP@language}%
1662 \fi
1663 \ifx\bb1@KVP@mapfont\@nil\else
1664   \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}{}%
1665     {\bb1@error{Option `bb1@KVP@mapfont' unknown for\%
1666       \mapfont. Use `direction'.%
1667       {See the manual for details.}}}%
1668 \bb1@ifunset{\bb1@lsys@\languagename}{\bb1@provide@lsys{\languagename}}{}%
1669 \bb1@ifunset{\bb1@wdir@\languagename}{\bb1@provide@dirs{\languagename}}{}%
1670 \ifx\bb1@mapselect\@undefined
1671   \AtBeginDocument{%
1672     \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
1673     {\selectfont}}%
1674   \def\bb1@mapselect{%
1675     \let\bb1@mapselect\relax
1676     \edef\bb1@prefontid{\fontid\font}}%
1677   \def\bb1@mapdir##1{%
1678     {\def\languagename{##1}\bb1@switchfont
1679     \directlua{Babel.fontmap
1680       [\the\csname bbl@wdir@##1\endcsname]%
1681       [\bb1@prefontid]=\fontid\font}}}%
1682   \fi
1683   \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\languagename}}}%
1684 \fi
1685 \ifcase\bb1@engine\else
1686   \bb1@ifunset{\bb1@dgnat@\languagename}{}%
1687     {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
1688       \expandafter\expandafter\expandafter
1689       \bb1@setdigits\csname bbl@dgnat@\languagename\endcsname
1690       \ifx\bb1@KVP@maparabic\@nil\else
1691         \expandafter\let\expandafter\@arabic
1692         \csname bbl@counter@\languagename\endcsname
1693       \fi
1694       \fi}%
1695 \fi
1696 \let\languagename\bb1@savelangname}
1697 \def\bb1@setdigits#1#2#3#4#5{%
1698 \bb1@exp{%
1699   \def\<\languagename digits>###1{% ie, \langdigits
1700     \<bbl@digits@\languagename>###1\@nil}%
1701   \def\<\languagename counter>###1{% ie, \langcounter
1702     \expandafter\<bbl@counter@\languagename>%
1703     \csname c@###1\endcsname}%
1704   \def\<bbl@counter@\languagename>###1{% ie, \bbl@counter@lang
1705     \expandafter\<bbl@digits@\languagename>%
1706     \number###1\@nil}}%
1707 \def\bb1@tempa##1##2##3##4##5{%
1708 \bb1@exp{% Wow, quite a lot of hashes! :-(\
1709   \def\<bbl@digits@\languagename>#####1{%
1710     \ifx#####1\@nil % ie, \bbl@digits@lang
1711     \else
1712       \ifx0#####1#1%
1713       \else\ifx1#####1#2%
1714       \else\ifx2#####1#3%
1715       \else\ifx3#####1#4%

```



```

1716     \\else\\ifx4#####1#5%
1717     \\else\\ifx5#####1#1%
1718     \\else\\ifx6#####1#2%
1719     \\else\\ifx7#####1#3%
1720     \\else\\ifx8#####1#4%
1721     \\else\\ifx9#####1#5%
1722     \\else#####1%
1723     \\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi
1724     \\expandafter\<bbl@digits@\language\>%
1725     \\fi}}}%
1726 \bbl@tempa}

```

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

-

```

1727 \def\bbl@provide@new#1{%
1728   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1729   \@namedef{extras#1}{}%
1730   \@namedef{noextras#1}{}%
1731   \StartBabelCommands*{#1}{captions}%
1732   \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
1733   \def\bbl@tempb##1{%              elt for \bbl@captionslist
1734     \ifx##1\@empty\else
1735       \bbl@exp{%
1736         \\SetString\\##1{%
1737           \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
1738       \expandafter\bbl@tempb
1739       \fi}%
1740   \expandafter\bbl@tempb\bbl@captionslist\@empty
1741   \else
1742     \bbl@read@ini{\bbl@KVP@captions}% Here all letters cat = 11
1743     \bbl@after@ini
1744     \bbl@savestrings
1745     \fi
1746   \StartBabelCommands*{#1}{date}%
1747   \ifx\bbl@KVP@import\@nil
1748     \bbl@exp{%
1749       \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
1750   \else
1751     \bbl@savetoday
1752     \bbl@savedate
1753     \fi
1754   \EndBabelCommands
1755   \bbl@exp{%
1756     \def\<#1hyphenmins>%
1757     {\bbl@ifunset{\bbl@lfthm@#1}{2}{\@nameuse{\bbl@lfthm@#1}}}%
1758     {\bbl@ifunset{\bbl@rgthm@#1}{3}{\@nameuse{\bbl@rgthm@#1}}}}}%
1759   \bbl@provide@hyphens{#1}%
1760   \ifx\bbl@KVP@main\@nil\else
1761     \expandafter\main@language\expandafter{#1}%
1762     \fi}
1763 \def\bbl@provide@renew#1{%
1764   \ifx\bbl@KVP@captions\@nil\else
1765     \StartBabelCommands*{#1}{captions}%
1766     \bbl@read@ini{\bbl@KVP@captions}% Here all letters cat = 11
1767     \bbl@after@ini
1768     \bbl@savestrings
1769     \EndBabelCommands
1770   \fi
1771   \ifx\bbl@KVP@import\@nil\else

```

```

1772 \StartBabelCommands*{#1}{date}%
1773 \bbl@savetoday
1774 \bbl@savedate
1775 \EndBabelCommands
1776 \fi
1777 \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

1778 \def\bbl@provide@hyphens#1{%
1779 \let\bbl@tempa\relax
1780 \ifx\bbl@KVP@hyphenrules\@nil\else
1781 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
1782 \bbl@foreach\bbl@KVP@hyphenrules{%
1783 \ifx\bbl@tempa\relax % if not yet found
1784 \bbl@ifsamestring{##1}{+}%
1785 {{\bbl@exp{\addlanguage\<l@##1>}}}%
1786 {}}%
1787 \bbl@ifunset{l@##1}%
1788 {}%
1789 {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
1790 \fi}%
1791 \fi
1792 \ifx\bbl@tempa\relax % if no opt or no language in opt found
1793 \ifx\bbl@KVP@import\@nil\else % if importing
1794 \bbl@exp{% and hyphenrules is not empty
1795 \bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
1796 {}}%
1797 {\let\bbl@tempa\<l@\@nameuse{bbl@hyphr@\languagename}>}}%
1798 \fi
1799 \fi
1800 \bbl@ifunset{bbl@tempa}% ie, relax or undefined
1801 {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
1802 {\bbl@exp{\adddialect\<l@#1>\language}}%
1803 {}}% so, l@<lang> is ok - nothing to do
1804 {\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.*

```

1805 \def\bbl@read@ini#1{%
1806 \openin1=babel-#1.ini
1807 \ifeof1
1808 \bbl@error
1809 {There is no ini file for the requested language\%
1810 (#1). Perhaps you misspelled it or your installation\%
1811 is not complete.}%
1812 {Fix the name or reinstall babel.}%
1813 \else
1814 \let\bbl@section\@empty
1815 \let\bbl@savestrings\@empty
1816 \let\bbl@savetoday\@empty
1817 \let\bbl@savedate\@empty
1818 \let\bbl@inireader\bbl@iniskip
1819 \bbl@info{Importing data from babel-#1.ini for \languagename}%
1820 \loop
1821 \if T\ifeof1F\fi T\relax % Trick, because inside \loop
1822 \endlinechar\m@ne
1823 \read1 to \bbl@line
1824 \endlinechar``^^M
1825 \ifx\bbl@line\@empty\else

```

```

1826     \expandafter\bb1@inline\bb1@line\bb1@inline
1827     \fi
1828     \repeat
1829     \fi}
1830 \def\bb1@inline#1\bb1@inline{%
1831   \@ifnextchar[\bb1@inisec{\@ifnextchar;\bb1@iniskip\bb1@inireader}#1\@@% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored.

```

1832 \def\bb1@iniskip#1\@@{%      if starts with ;
1833 \def\bb1@inisec[#1]#2\@@{%   if starts with opening bracket
1834   \@nameuse{bb1@secpost@\bb1@section}% ends previous section
1835   \def\bb1@section{#1}%
1836   \@nameuse{bb1@secpre@\bb1@section}% starts current section
1837   \bb1@ifunset{bb1@secline@#1}%
1838     {\let\bb1@inireader\bb1@iniskip}%
1839     {\bb1@exp{\let\bb1@inireader<\bb1@secline@#1>}}}

```

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

```

1840 \def\bb1@inikv#1=#2\@@{%    key=value
1841   \bb1@trim@def\bb1@tempa{#1}%
1842   \bb1@trim\toks@{#2}%
1843   \bb1@csarg\edef{kv@\bb1@section.\bb1@tempa}{\the\toks@}}

```

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

```

1844 \def\bb1@exportkey#1#2#3{%
1845   \bb1@ifunset{bb1@kv@#2}%
1846     {\bb1@csarg\gdef{#1@\languagename}{#3}}%
1847     {\expandafter\ifx\csname bb1@kv@#2\endcsname\@empty
1848       \bb1@csarg\gdef{#1@\languagename}{#3}}%
1849     \else
1850       \bb1@exp{\global\let<\bb1@#1@\languagename>\bb1@kv@#2>}%
1851     \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.

```

1852 \let\bb1@secline@identification\bb1@inikv
1853 \def\bb1@secpost@identification{%
1854   \bb1@exportkey{lname}{identification.name.english}{}%
1855   \bb1@exportkey{lbcpr}{identification.tag.bcp47}{}%
1856   \bb1@exportkey{lotf}{identification.tag.opentype}{dflt}%
1857   \bb1@exportkey{sname}{identification.script.name}{}%
1858   \bb1@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
1859   \bb1@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
1860 \let\bb1@secline@typography\bb1@inikv
1861 \let\bb1@secline@numbers\bb1@inikv
1862 \def\bb1@after@ini{%
1863   \bb1@exportkey{lftm}{typography.lefthyphenmin}{2}%
1864   \bb1@exportkey{rgtm}{typography.righthyphenmin}{3}%
1865   \bb1@exportkey{hyphr}{typography.hyphenrules}{}%
1866   \bb1@exportkey{dgnat}{numbers.digits.native}{}%
1867   \bb1@xin@{0.5}{\@nameuse{bb1@kv@identification.version}}%
1868   \ifin@
1869     \bb1@warning{%
1870       There are neither captions nor date in '\languagename'.\%
1871       It may not be suitable for proper typesetting, and it\%
1872       could change. Reported}%

```

```

1873 \fi
1874 \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
1875 \ifin@
1876 \bbl@warning{%
1877     The '\language' date format may not be suitable\\%
1878     for proper typesetting, and therefore it very likely will\\%
1879     change in a future release. Reported}%
1880 \fi
1881 \bbl@toglobal\bbl@savetoday
1882 \bbl@toglobal\bbl@savestate}

```

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

```

1883 \ifcase\bbl@engine
1884 \bbl@csarg\def{secline@captions.licr}#1=#2\@@{%
1885     \bbl@ini@captions@aux{#1}{#2}}
1886 \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%           for defaults
1887     \bbl@ini@dategreg#1...\relax{#2}}
1888 \bbl@csarg\def{secline@date.gregorian.licr}#1=#2\@@{%      override
1889     \bbl@ini@dategreg#1...\relax{#2}}
1890 \else
1891 \def\bbl@secline@captions#1=#2\@@{%
1892     \bbl@ini@captions@aux{#1}{#2}}
1893 \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%
1894     \bbl@ini@dategreg#1...\relax{#2}}
1895 \fi

```

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

```

1896 \def\bbl@ini@captions@aux#1#2{%
1897     \bbl@trim@def\bbl@tempa{#1}%
1898     \bbl@ifblank{#2}%
1899     {\bbl@exp{%
1900         \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
1901     {\bbl@trim\toks@{#2}}}%
1902     \bbl@exp{%
1903         \bbl@add\bbl@savestrings{%
1904             \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.

```

1905 \bbl@csarg\def{secre@date.gregorian.licr}{%
1906     \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
1907 \def\bbl@ini@dategreg#1.#2.#3.#4\relax#5{% TODO - ignore with 'captions'
1908     \bbl@trim@def\bbl@tempa{#1.#2}%
1909     \bbl@ifsamestring{\bbl@tempa}{months.wide}%
1910     {\bbl@trim@def\bbl@tempa{#3}%
1911     \bbl@trim\toks@{#5}%
1912     \bbl@exp{%
1913         \bbl@add\bbl@savestate{%
1914             \SetString\<month\romannumeral\bbl@tempa name>{\the\toks@}}}}%
1915     {\bbl@ifsamestring{\bbl@tempa}{date.long}%
1916     {\bbl@trim@def\bbl@toreplace{#5}%
1917     \bbl@TG@date
1918     \global\bbl@csarg\let{date@\language}\bbl@toreplace
1919     \bbl@exp{%
1920         \gdef\<\language date>{\protect\<\language date >}%
1921         \gdef\<\language date >####1####2####3{%
1922             \bbl@usedategrouptrue
1923             \<\bbl@ensure@\language>{%

```

```

1924         \<bbl@date@languagename>{####1}{####2}{####3}}}%
1925         \\bbl@add\\bbl@savetoday{%
1926         \\SetString\\today{%
1927         \<languagename date>{\\the\year}{\\the\month}{\\the\day}}}}}%
1928         {}

```

Dates will require some macros for the basic formatting. They may be redefined by language, so “semi-public” names (camel case) are used. Oddly enough, the CLDR places particles like “de” inconsistently in either in the date or in the month name.

```

1929 \newcommand\BabelDateSpace{\nobreakspace}
1930 \newcommand\BabelDateDot{.\@}
1931 \newcommand\BabelDated[1]{\number#1}
1932 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
1933 \newcommand\BabelDateM[1]{\number#1}
1934 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
1935 \newcommand\BabelDateMMMM[1]{%
1936   \csname month\romannumeral#1name\endcsname}}%
1937 \newcommand\BabelDatey[1]{\number#1}%
1938 \newcommand\BabelDateyy[1]{%
1939   \ifnum#1<10 0\number#1 %
1940   \else\ifnum#1<100 \number#1 %
1941   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
1942   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
1943   \else
1944     \bbl@error
1945     {Currently two-digit years are restricted to the\
1946     range 0-9999.}%
1947     {There is little you can do. Sorry.}%
1948     \fi\fi\fi\fi}}
1949 \newcommand\BabelDateyyyy[1]{\number#1}
1950 \def\bbl@replace@finish@iii#1{%
1951   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
1952 \def\bbl@TG@date{%
1953   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
1954   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}}%
1955   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}}%
1956   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}}%
1957   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}}%
1958   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}}%
1959   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}}%
1960   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}}%
1961   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}}%
1962   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}}%
1963 % Note after \bbl@replace \toks@ contains the resulting string.
1964 % TODO - Using this implicit behavior doesn't seem a good idea.
1965   \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.

```

1966 \def\bbl@provide@lsys#1{%
1967   \bbl@ifunset{bbl@lname@#1}%
1968     {\bbl@ini@ids{#1}}%
1969     {}%
1970   \bbl@csarg\let{lsys@#1}\@empty
1971   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
1972   \bbl@ifunset{bbl@sotf#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
1973   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
1974   \bbl@ifunset{bbl@lname@#1}}}%
1975   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%

```

```

1976 \bbl@csarg\bbl@tglobal{lsys@#1}%
1977 % \bbl@exp{% TODO - should be global
1978 % \<keys_if_exist:nnF>{fontspec-opentype/Script}{\bbl@cs{sname@#1}}%
1979 % {\newfontscript{\bbl@cs{sname@#1}}{\bbl@cs{sotf@#1}}}%
1980 % \<keys_if_exist:nnF>{fontspec-opentype/Language}{\bbl@cs{lname@#1}}%
1981 % {\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.

```

1982 \def\bbl@ini@ids#1{%
1983 \def\BabelBeforeIni##1##2{%
1984 \begingroup
1985 \bbl@add\bbl@secpost@identification{\closein1 }%
1986 \catcode`\[=12 \catcode`\]=12 \catcode`\=12 %
1987 \bbl@read@ini{##1}%
1988 \endgroup}% boxed, to avoid extra spaces:
1989 {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}}

```

10 The kernel of Babel (babel.def, only \LaTeX)

10.1 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`.

```

1990 {\def\format{lplain}
1991 \ifx\fmtname\format
1992 \else
1993 \def\format{LaTeX2e}
1994 \ifx\fmtname\format
1995 \else
1996 \aftergroup\endinput
1997 \fi
1998 \fi}

```

10.2 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 \TeX book [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.

```
1999 %\bbl@redefine\newlabel#1#2{%
2000 % \@safe@activestruelorg@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.

```
2001 <<{*More package options}>> ≡
2002 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2003 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2004 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2005 <</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.

```
2006 \bbl@trace{Cross referencing macros}
2007 \ifx\bbl@opt@safe\@empty\else
2008 \def\@newl@bel#1#2#3{%
2009   {\@safe@activestruel
2010     \bbl@ifunset{#1@#2}%
2011     \relax
2012     {\gdef\@multiplelabels{%
2013       \@latex@warning@no@line{There were multiply-defined labels}}%
2014     \@latex@warning@no@line{Label `#2' multiply defined}}%
2015     \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.

```
2016 \CheckCommand*\@testdef[3]{%
2017   \def\reserved@a{#3}%
2018   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2019   \else
2020     \@tempwatruel
2021     \fi}
```

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

```
2022 \def\@testdef#1#2#3{%
2023   \@safe@activestruel
```

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

```
2024 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```
2025 \def\bbl@tempb{#3}%
2026 \@safe@activesfalse
```

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

```
2027 \ifx\bbl@tempa\relax
2028 \else
2029   \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2030 \fi
```

We do the same for `\bbl@tempb`.

```
2031 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```
2032 \ifx\bbl@tempa\bbl@tempb
2033 \else
2034 \@tempswatruue
2035 \fi}
2036 \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.

```
2037 \bbl@xin@{R}\bbl@opt@safe
2038 \ifin@
2039 \bbl@redefineroobust\ref#1{%
2040 \@safe@activestruue\org@ref{#1}\@safe@activesfalse}
2041 \bbl@redefineroobust\pageref#1{%
2042 \@safe@activestruue\org@pageref{#1}\@safe@activesfalse}
2043 \else
2044 \let\org@ref\ref
2045 \let\org@pageref\pageref
2046 \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.

```
2047 \bbl@xin@{B}\bbl@opt@safe
2048 \ifin@
2049 \bbl@redefine\@citex[#1]#2{%
2050 \@safe@activestruue\edef\@tempa{#2}\@safe@activesfalse
2051 \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.

```
2052 \AtBeginDocument{%
2053 \@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.)

```
2054 \def\@citex[#1][#2]#3{%
2055 \@safe@activestruue\edef\@tempa{#3}\@safe@activesfalse
2056 \org@@citex[#1][#2]{\@tempa}}%
2057 }{}}
```

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

```
2058 \AtBeginDocument{%
2059 \@ifpackageloaded{cite}{%
2060 \def\@citex[#1]#2{%
```



```

2061     \@safe@activestruel\org@@citex[#1]{#2}\@safe@activesfalse}%
2062     }{}}

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

2063     \bbl@redefine\nocite#1{%
2064     \@safe@activestruel\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@activestruel 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.

2065     \bbl@redefine\bibcite{%

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

2066     \bbl@cite@choice
2067     \bibcite}

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

2068     \def\bbl@bibcite#1#2{%
2069     \org@bibcite{#1}{\@safe@activesfalse#2}}

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

2070     \def\bbl@cite@choice{%

        First we give \bibcite its default definition.

2071     \global\let\bibcite\bbl@bibcite

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

2072     \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%

        For cite we do the same.

2073     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%

        Make sure this only happens once.

2074     \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.

2075     \AtBeginDocument{\bbl@cite@choice}

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

2076     \bbl@redefine\@bibitem#1{%
2077     \@safe@activestruel\org@@bibitem{#1}\@safe@activesfalse}
2078 \else
2079     \let\org@nocite\nocite
2080     \let\org@@citex\@citex
2081     \let\org@bibcite\bibcite
2082     \let\org@@bibitem\@bibitem
2083 \fi

```

10.3 Marks

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

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

```
2084 \bbl@trace{Marks}
2085 \IfBabelLayout{sectioning}
2086   {\ifx\bbl@opt@headfoot\@nnil
2087     \g@addto@macro\resetactivechars{%
2088       \set@typeset@protect
2089       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2090       \let\protect\noexpand}%
2091   \fi}
2092   {\bbl@redefine\markright#1{%
2093     \bbl@ifblank{#1}%
2094     {\org@markright{}}%
2095     {\toks@{#1}%
2096     \bbl@exp{%
2097       \\org@markright{\\protect\\foreignlanguage{\language}%
2098       {\\protect\\bbl@restore@actives\the\toks@}}}}
```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers. The documentclasses `report` and `book` define and set the headings for the page. While doing so they also store a copy of `\markboth` in `\@mkboth`. Therefore we need to check whether `\@mkboth` has already been set. If so we need to do that again with the new definition of `\markboth`.

```
2099 \ifx\@mkboth\markboth
2100   \def\bbl@tempc{\let\@mkboth\markboth}
2101 \else
2102   \def\bbl@tempc{}
2103 \fi
```

Now we can start the new definition of `\markboth`

```
2104 \bbl@redefine\markboth#1#2{%
2105   \protected@edef\bbl@tempb##1{%
2106     \protect\foreignlanguage
2107     {\language}{\protect\bbl@restore@actives##1}}%
2108   \bbl@ifblank{#1}%
2109   {\toks@{}}%
2110   {\toks@\expandafter{\bbl@tempb{#1}}}%
2111   \bbl@ifblank{#2}%
2112   {\@temptokena{}}%
2113   {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2114   \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}
```

and copy it to `\@mkboth` if necessary.

```
2115 \bbl@tempc} % end \IfBabelLayout
```

10.4 Preventing clashes with other packages

10.4.1 ifthen

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

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

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

```
2116 \bbl@trace{Preventing clashes with other packages}
2117 \bbl@xin@{R}\bbl@opt@safe
2118 \ifin@
2119 \AtBeginDocument{%
2120   \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```
2121   \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.

```
2122   \let\bbl@temp@pref\pageref
2123   \let\pageref\org@pageref
2124   \let\bbl@temp@ref\ref
2125   \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.

```
2126   \@safe@activestrue
2127   \org@ifthenelse{#1}%
2128   {\let\pageref\bbl@temp@pref
2129    \let\ref\bbl@temp@ref
2130    \@safe@activesfalse
2131    #2}%
2132   {\let\pageref\bbl@temp@pref
2133    \let\ref\bbl@temp@ref
2134    \@safe@activesfalse
2135    #3}%
2136   }%
2137   }{}%
2138 }
```

10.4.2 varioref

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

```
\Ref 2139 \AtBeginDocument{%
2140   \@ifpackageloaded{varioref}{%
2141     \bbl@redefine\@@vpageref#1[#2]#3{%
```

```

2142     \@safe@activestru
2143     \org@@@vpageref{#1}[#2]{#3}%
2144     \@safe@activesfalse}%

```

The same needs to happen for \vrefpagenum.

```

2145     \bbl@redefine\vrefpagenum#1#2{%
2146     \@safe@activestru
2147     \org@vrefpagenum{#1}{#2}%
2148     \@safe@activesfalse}%

```

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

```

2149     \expandafter\def\csname Ref \endcsname#1{%
2150     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2151     }{}%
2152     }
2153 \fi

```

10.4.3 holine

\holine Delaying the activation of the shorthand characters has introduced a problem with the holine 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 holine is loaded.

```

2154 \AtEndOfPackage{%
2155 \AtBeginDocument{%
2156 \ifpackageloaded{holine}%

```

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

```

2157     {\expandafter\ifx\csname normal@char\string:\endcsname\relax
2158     \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.

```

2159     \makeatletter
2160     \def\@currname{holine}\input{holine.sty}\makeatother
2161     \fi}%
2162     {}}

```

10.4.4 hyperref

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

```

2163 \AtBeginDocument{%
2164 \ifx\pdfstringdefDisableCommands\undefined\else
2165 \pdfstringdefDisableCommands{\languageshortands{system}}%
2166 \fi}

```

10.4.5 fancyhdr

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

```
2167 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2168   \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.

```
2169 \def\substitutefontfamily#1#2#3{%
2170   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2171   \immediate\write15{%
2172     \string\ProvidesFile{#1#2.fd}%
2173     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2174     \space generated font description file]^{}
2175     \string\DeclareFontFamily{#1}{#2}{}^{}
2176     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^{}
2177     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^{}
2178     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^{}
2179     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^{}
2180     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^{}
2181     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^{}
2182     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^{}
2183     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^{}
2184   }%
2185   \closeout15
2186 }
```

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

```
2187 \@onlypreamble\substitutefontfamily
```

10.5 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
```

```
2188 \bbl@trace{Encoding and fonts}
2189 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,}
2190 \let\org@TeX\TeX
2191 \let\org@LaTeX\LaTeX
2192 \let\ensureascii\@firstofone
2193 \AtBeginDocument{%
2194   \in@false
2195   \bbl@foreach\BabelNonASCII{% is there a non-ascii enc?
2196     \ifin@\else
2197       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
2198     \fi}%
2199   \ifin@ % if a non-ascii has been loaded
2200   \def\ensureascii#1{\fontencoding{OT1}\selectfont#1}}%
```

```

2201 \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2202 \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2203 \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
2204 \def\bbl@tempc#1ENC.DEF#2\@@{\%
2205   \ifx\@empty#2\else
2206     \bbl@ifunset{T@#1}%
2207     {}%
2208     {\bbl@xin@{,#1,},{, \BabelNonASCII,}%
2209     \ifin@
2210       \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2211       \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2212     \else
2213       \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2214       \fi}%
2215   \fi}%
2216 \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
2217 \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,}%
2218 \ifin@\else
2219   \edef\ensureascii#1{%
2220     \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2221 \fi
2222 \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.

```

2223 \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.

```

2224 \AtBeginDocument{%
2225   \@ifpackageloaded{fontspec}%
2226   {\xdef\latinencoding{%
2227     \ifx\UTFencname\@undefined
2228       EU\ifcase\bbl@engine\or2\or1\fi
2229     \else
2230       \UTFencname
2231     \fi}}%
2232   {\gdef\latinencoding{OT1}%
2233     \ifx\cf@encoding\bbl@t@one
2234       \xdef\latinencoding{\bbl@t@one}%
2235     \else
2236       \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
2237   \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.

```

2238 \DeclareRobustCommand{\latintext}{%
2239   \fontencoding{\latinencoding}\selectfont
2240   \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.

```
2241 \ifx\@undefined\DeclareTextFontCommand
2242 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2243 \else
2244 \DeclareTextFontCommand{\textlatin}{\latintext}
2245 \fi
```

10.6 Basic bidi support

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

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

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

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `LuaTeX-ja` shows, vertical typesetting is possible, too. Its main drawback is font handling is often considered to be less mature than `xetex`, mainly in Indic scripts (but there are steps to make `HarfBuzz`, the `xetex` font engine, available in `luatex`; see <https://github.com/tatzetwerk/luatex-harfbuzz>).

```
2246 \bbl@trace{Basic (internal) bidi support}
2247 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2248 \def\bbl@rscripts{%
2249   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2250   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
2251   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
2252   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2253   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2254   Old South Arabian,}%
2255 \def\bbl@provide@dirs#1{%
2256   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2257   \ifin@
2258     \global\bbl@csarg\chardef{wdir@#1}\@ne
2259     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2260     \ifin@
2261     \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2262     \fi
2263   \else
2264     \global\bbl@csarg\chardef{wdir@#1}\z@
2265     \fi}
2266 \def\bbl@switchdir{%
```

```

2267 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%
2268 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}}%
2269 \bbl@exp{\bbl@setdirs\bbl@cs{wdir@\languagename}}{}}
2270 \def\bbl@setdirs#1{% TODO - math
2271 \ifcase\bbl@select@type % TODO - strictly, not the right test
2272 \bbl@bodydir{#1}%
2273 \bbl@pardir{#1}%
2274 \fi
2275 \bbl@textdir{#1}}
2276 \ifodd\bbl@engine % luatex=1
2277 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2278 \DisableBabelHook{babel-bidi}
2279 \chardef\bbl@thepardir\z@
2280 \def\bbl@getluadir#1{%
2281 \directlua{
2282 if tex.#1dir == 'TLT' then
2283 tex.sprint('0')
2284 elseif tex.#1dir == 'TRT' then
2285 tex.sprint('1')
2286 end}}
2287 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 r1
2288 \ifcase#3\relax
2289 \ifcase\bbl@getluadir{#1}\relax\else
2290 #2 TLT\relax
2291 \fi
2292 \else
2293 \ifcase\bbl@getluadir{#1}\relax
2294 #2 TRT\relax
2295 \fi
2296 \fi}
2297 \def\bbl@textdir#1{%
2298 \bbl@setluadir{text}\textdir{#1}% TODO - ?\linedir
2299 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2300 \def\bbl@pardir#1{\bbl@setluadir{par}\pardir{#1}%
2301 \chardef\bbl@thepardir#1\relax}
2302 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2303 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2304 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
2305 \else % pdftex=0, xetex=2
2306 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2307 \DisableBabelHook{babel-bidi}
2308 \newcount\bbl@dirlevel
2309 \chardef\bbl@thetextdir\z@
2310 \chardef\bbl@thepardir\z@
2311 \def\bbl@textdir#1{%
2312 \ifcase#1\relax
2313 \chardef\bbl@thetextdir\z@
2314 \bbl@textdir@i\beginL\endL
2315 \else
2316 \chardef\bbl@thetextdir@ne
2317 \bbl@textdir@i\beginR\endR
2318 \fi}
2319 \def\bbl@textdir@i#1#2{%
2320 \ifhmode
2321 \ifnum\currentgrouplevel>\z@
2322 \ifnum\currentgrouplevel=\bbl@dirlevel
2323 \bbl@error{Multiple bidi settings inside a group}%
2324 {I'll insert a new group, but expect wrong results.}%
2325 \bgroup\aftergroup#2\aftergroup\egroup

```



```

2326     \else
2327         \ifcase\currentgrouptype\or % 0 bottom
2328             \aftergroup#2% 1 simple {}
2329         \or
2330             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2331         \or
2332             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2333         \or\or % vbox vtop align
2334         \or
2335             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2336         \or\or\or\or\or\or % output math disc insert vcent mathchoice
2337         \or
2338             \aftergroup#2% 14 \beginngroup
2339         \else
2340             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2341         \fi
2342     \fi
2343     \bbl@dirlevel\currentgrouplevel
2344     \fi
2345     #1%
2346     \fi}
2347 \def\bbl@pdir#1{\chardef\bbl@thepardir#1\relax}
2348 \let\bbl@bodydir\@gobble
2349 \let\bbl@pagedir\@gobble
2350 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

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

```

2351 \def\bbl@xebidipar{%
2352     \let\bbl@xebidipar\relax
2353     \TeXeTstate\@ne
2354     \def\bbl@xeverypar{%
2355         \ifcase\bbl@thepardir
2356             \ifcase\bbl@thetextdir\else\beginR\fi
2357         \else
2358             {\setbox\z@\lastbox\beginR\box\z@}%
2359         \fi}%
2360     \let\bbl@severypar\everypar
2361     \newtoks\everypar
2362     \everypar=\bbl@severypar
2363     \bbl@severypar{\bbl@xeverypar\the\everypar}}
2364 \fi

```

A tool for weak L (mainly digits).

```

2365 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}

```

10.7 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `nor.sk.cfg` will be loaded when the language definition file `nor.sk.ldf` is loaded. For plain-based formats we don't want to override the definition of `\loadlocalcfg` from `plain.def`.

```

2366 \bbl@trace{Local Language Configuration}
2367 \ifx\loadlocalcfg\undefined

```

```

2368 \@ifpackagewith{babel}{noconfigs}%
2369   {\let\loadlocalcfg@gobble}%
2370   {\def\loadlocalcfg#1{%
2371     \InputIfFileExists{#1.cfg}%
2372     {\typeout{*****^J%
2373               * Local config file #1.cfg used^^J%
2374               *}}}%
2375     \@empty}}
2376 \fi

```

Just to be compatible with L^AT_EX 2.09 we add a few more lines of code:

```

2377 \ifx\@unexpandable@protect\@undefined
2378   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2379   \long\def\protected@write#1#2#3{%
2380     \begingroup
2381       \let\thepage\relax
2382       #2%
2383       \let\protect\@unexpandable@protect
2384       \edef\reserved@a{\write#1{#3}}%
2385       \reserved@a
2386     \endgroup
2387     \if@nobreak\ifvmode\nobreak\fi\fi}
2388 \fi
2389 </core>
2390 <*kernel>

```

11 Multiple languages (switch.def)

Plain T_EX 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.

```

2391 <<Make sure ProvidesFile is defined>>
2392 \ProvidesFile{switch.def}[\<<date>>] [\<<version>>] Babel switching mechanism]
2393 <<Load macros for plain if not LaTeX>>
2394 <<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.

```

2395 \def\bbl@version{\<<version>>}
2396 \def\bbl@date{\<<date>>}
2397 \def\adddialect#1#2{%
2398   \global\chardef#1#2\relax
2399   \bbl@usehooks{adddialect}{#1}{#2}}%
2400 \wlog{\string#1 = a dialect from \string\language#2}}

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```

2401 \def\bbl@fixname#1{%
2402   \begingroup
2403     \def\bbl@tempe{l@}%
2404     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%

```

```

2405 \bbl@tempd
2406   {\lowercase\expandafter{\bbl@tempd}%
2407     {\uppercase\expandafter{\bbl@tempd}}%
2408     \@empty
2409     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2410      \uppercase\expandafter{\bbl@tempd}}}%
2411   {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2412    \lowercase\expandafter{\bbl@tempd}}}%
2413   \@empty
2414   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2415 \bbl@tempd}
2416 \def\bbl@iflanguage#1{%
2417   \@ifundefined{l@#1}{\@noanerr{#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.

```

2418 \def\iflanguage#1{%
2419   \bbl@iflanguage{#1}{%
2420     \ifnum\csname l@#1\endcsname=\language
2421       \expandafter\@firstoftwo
2422     \else
2423       \expandafter\@secondoftwo
2424     \fi}}

```

11.1 Selecting the language

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

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

```

2425 \let\bbl@select@type\z@
2426 \edef\selectlanguage{%
2427   \noexpand\protect
2428   \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

2429 \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.

```
2430 \ifx\documentclass\undefined
2431   \def\xstring{\string\string\string}
2432 \else
2433   \let\xstring\string
2434 \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.

```
2435 \def\bbl@language@stack{}
```

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

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

`\bbl@pop@language`

```
2436 \def\bbl@push@language{%
2437   \xdef\bbl@language@stack{\language+\bbl@language@stack}}
```

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

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

```
2438 \def\bbl@pop@lang#1+#2-#3{%
2439   \edef\language{#1}\xdef#3{#2}}
```

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

```
2440 \let\bbl@ifrestoring\@secondoftwo
2441 \def\bbl@pop@language{%
2442   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2443   \let\bbl@ifrestoring\@firstoftwo
2444   \expandafter\bbl@set@language\expandafter{\language}%
2445   \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.

```

2446 \expandafter\def\csname selectlanguage \endcsname#1{%
2447 \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw\fi
2448 \bbl@push@language
2449 \aftergroup\bbl@pop@language
2450 \bbl@set@language{#1}}

```

`\bbl@set@language` The macro `\bbl@set@language` takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of `\language`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\language` are 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.

```

2451 \def\BabelContentsFiles{toc,lof,lot}
2452 \def\bbl@set@language#1{%
2453 \edef\language{%
2454 \ifnum\escapechar=\expandafter`\string#1\@empty
2455 \else\string#1\@empty\fi}%
2456 \select@language{\language}%
2457 \expandafter\ifx\csname date\language\endcsname\relax\else
2458 \if@filesw
2459 \protected@write\@auxout{{\string\babel@aux{\language}}}%
2460 \bbl@usehooks{write}}}%
2461 \fi
2462 \fi}
2463 \def\select@language#1{%
2464 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
2465 \edef\language{#1}%
2466 \bbl@fixname\language
2467 \bbl@iflanguage\language{%
2468 \expandafter\ifx\csname date\language\endcsname\relax
2469 \bbl@error
2470 {Unknown language `#1'. Either you have\\%
2471 misspelled its name, it has not been installed,\\%
2472 or you requested it in a previous run. Fix its name,\\%
2473 install it or just rerun the file, respectively. In\\%
2474 some cases, you may need to remove the aux file}%
2475 {You may proceed, but expect wrong results}%
2476 \else
2477 \let\bbl@select@type\z@
2478 \expandafter\bbl@switch\expandafter{\language}%
2479 \fi}}
2480 \def\babel@aux#1#2{%
2481 \expandafter\ifx\csname date#1\endcsname\relax
2482 \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
2483 \@namedef{bbl@auxwarn@#1}{}%
2484 \bbl@warning
2485 {Unknown language `#1'. Very likely you\\%
2486 requested it in a previous run. Expect some\\%
2487 wrong results in this run, which should vanish\\%
2488 in the next one. Reported}%
2489 \fi
2490 \else
2491 \select@language{#1}%
2492 \bbl@foreach\BabelContentsFiles{%
2493 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
2494 \fi}

```

```

2495 \def\babel@toc#1#2{%
2496   \select@language{#1}}

```

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

```

2497 \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 *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

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

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

```

2498 \newif\ifbbl@usedategroup
2499 \def\bbl@switch#1{%
2500   \originalTeX
2501   \expandafter\def\expandafter\originalTeX\expandafter{%
2502     \csname noextras#1\endcsname
2503     \let\originalTeX\@empty
2504     \babel@beginsave}%
2505   \bbl@usehooks{afterreset}{}%
2506   \languageshorthands{none}%
2507   \ifcase\bbl@select@type
2508     \ifhmode
2509       \hskip\z@skip % trick to ignore spaces
2510       \csname captions#1\endcsname\relax
2511       \csname date#1\endcsname\relax
2512       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2513     \else
2514       \csname captions#1\endcsname\relax
2515       \csname date#1\endcsname\relax
2516     \fi
2517   \else\ifbbl@usedategroup
2518     \bbl@usedategroupfalse
2519     \ifhmode
2520       \hskip\z@skip % trick to ignore spaces
2521       \csname date#1\endcsname\relax
2522       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2523     \else
2524       \csname date#1\endcsname\relax
2525     \fi
2526   \fi\fi
2527   \bbl@usehooks{beforeextras}{}%
2528   \csname extras#1\endcsname\relax
2529   \bbl@usehooks{afterextras}{}%
2530   \ifcase\bbl@opt@hyphenmap\or
2531     \def\BabelLower##1##2{\lccode##1=##2\relax}%
2532   \ifnum\bbl@hymapsel>4\else
2533     \csname\languagename @bbl@hyphenmap\endcsname

```

```

2534 \fi
2535 \chardef\bbl@opt@hyphenmap\z@
2536 \else
2537 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2538 \csname\languagenam @bbl@hyphenmap\endcsname
2539 \fi
2540 \fi
2541 \global\let\bbl@hymapsel\@cclv
2542 \bbl@patterns{#1}%
2543 \babel@savevariable\lefthyphenmin
2544 \babel@savevariable\righthyphenmin
2545 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2546 \set@hyphenmins\tw@\thr@\relax
2547 \else
2548 \expandafter\expandafter\expandafter\set@hyphenmins
2549 \csname #1hyphenmins\endcsname\relax
2550 \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.

```

2551 \long\def\otherlanguage#1{%
2552 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
2553 \csname selectlanguage \endcsname{#1}%
2554 \ignorespaces}

```

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

```

2555 \long\def\endotherlanguage{%
2556 \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`.

```

2557 \expandafter\def\csname otherlanguage*\endcsname#1{%
2558 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2559 \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”.

```

2560 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras⟨lang⟩` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`. `\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

(3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in vmode and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into hmode with the surrounding lang, and with `\foreignlanguage*` with the new lang.

```

2561 \providecommand\bb1@beforeforeign{}
2562 \edef\foreignlanguage{%
2563   \noexpand\protect
2564   \expandafter\noexpand\csname foreignlanguage \endcsname}
2565 \expandafter\def\csname foreignlanguage \endcsname{%
2566   \@ifstar\bb1@foreign@s\bb1@foreign@x}
2567 \def\bb1@foreign@x#1#2{%
2568   \begingroup
2569     \let\BabelText\@firstofone
2570     \bb1@beforeforeign
2571     \foreign@language{#1}%
2572     \bb1@usehooks{foreign}{}%
2573     \BabelText{#2}% Now in horizontal mode!
2574   \endgroup}
2575 \def\bb1@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
2576   \begingroup
2577     {\par}%
2578     \let\BabelText\@firstofone
2579     \foreign@language{#1}%
2580     \bb1@usehooks{foreign*}{}%
2581     \bb1@dirparastext
2582     \BabelText{#2}% Still in vertical mode!
2583     {\par}%
2584   \endgroup}

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the other `language*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bb1@switch`.

```

2585 \def\foreign@language#1{%
2586   \edef\languagename{#1}%
2587   \bb1@fixname\languagename
2588   \bb1@iflanguage\languagename{%
2589     \expandafter\ifx\csname date\languagename\endcsname\relax
2590       \bb1@warning
2591         {Unknown language `#1'. Either you have\\%
2592           misspelled its name, it has not been installed,\\%
2593           or you requested it in a previous run. Fix its name,\\%
2594           install it or just rerun the file, respectively.\\%
2595           I'll proceed, but expect wrong results.\\%
2596           Reported}%
2597     \fi
2598     \let\bb1@select@type\@ne
2599     \expandafter\bb1@switch\expandafter{\languagename}}

```

`\bb1@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.

```

2600 \let\bbl@hyphlist\@empty
2601 \let\bbl@hyphenation@\relax
2602 \let\bbl@pttnlist\@empty
2603 \let\bbl@patterns@\relax
2604 \let\bbl@hymapsel=\@ccclv
2605 \def\bbl@patterns#1{%
2606   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2607     \csname l@#1\endcsname
2608     \edef\bbl@tempa{#1}%
2609   \else
2610     \csname l@#1:\f@encoding\endcsname
2611     \edef\bbl@tempa{#1:\f@encoding}%
2612   \fi
2613   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
2614   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
2615     \begingroup
2616       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
2617       \ifin@ \else
2618         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
2619         \hyphenation{%
2620           \bbl@hyphenation@
2621           \@ifundefined{bbl@hyphenation@#1}%
2622             \@empty
2623             {\space\csname bbl@hyphenation@#1\endcsname}}%
2624         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2625       \fi
2626     \endgroup}}

```

hyphenrules The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change `\languagenam` and when the hyphenation rules specified were not loaded it has no effect. Note however, `\lccode`'s and font encodings are not set at all, so in most cases you should use other language*.

```

2627 \def\hyphenrules#1{%
2628   \edef\bbl@tempf{#1}%
2629   \bbl@fixname\bbl@tempf
2630   \bbl@iflanguage\bbl@tempf{%
2631     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2632     \languageshortands{none}%
2633     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2634       \set@hyphenmins\tw@\thr@@\relax
2635   \else
2636     \expandafter\expandafter\expandafter\set@hyphenmins
2637     \csname\bbl@tempf hyphenmins\endcsname\relax
2638   \fi}}
2639 \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.

```

2640 \def\providehyphenmins#1#2{%
2641   \expandafter\ifx\csname #1hyphenmins\endcsname\relax

```

```
2642 \namedef{#1hyphenmins}{#2}%
2643 \fi}
```

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

```
2644 \def\set@hyphenmins#1#2{%
2645 \lefthyphenmin#1\relax
2646 \righthyphenmin#2\relax}
```

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

```
2647 \ifx\ProvidesFile\undefined
2648 \def\ProvidesLanguage#1[#2 #3 #4]{%
2649 \wlog{Language: #1 #4 #3 <#2>}%
2650 }
2651 \else
2652 \def\ProvidesLanguage#1{%
2653 \begingroup
2654 \catcode`\ 10 %
2655 \@makeother\%
2656 \@ifnextchar[%]
2657 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2658 \def\@provideslanguage#1[#2]{%
2659 \wlog{Language: #1 #2}%
2660 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2661 \endgroup}
2662 \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`

```
2663 \def\LdfInit{%
2664 \chardef\atcatcode=\catcode`\@
2665 \catcode`\@=11\relax
2666 \input babel.def\relax
2667 \catcode`\@=\atcatcode \let\atcatcode\relax
2668 \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`.

```
2669 \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.

```
2670 \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’:

```
2671 \providecommand\setlocale{%
2672 \bb1@error
```

```

2673 {Not yet available}%
2674 {Find an armchair, sit down and wait}}
2675 \let\uselocale\setlocale
2676 \let\locale\setlocale
2677 \let\selectlocale\setlocale
2678 \let\textlocale\setlocale
2679 \let\textlanguage\setlocale
2680 \let\languagetext\setlocale

```

11.2 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 $\LaTeX 2\epsilon$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

```

2681 \edef\bbl@nulllanguage{\string\language=0}
2682 \ifx\PackageError\@undefined
2683   \def\bbl@error#1#2{%
2684     \begingroup
2685       \newlinechar=`^^J
2686       \def\{^^J(babel) }%
2687       \errhelp{#2}\errmessage{\{#1}%
2688     \endgroup}
2689   \def\bbl@warning#1{%
2690     \begingroup
2691       \newlinechar=`^^J
2692       \def\{^^J(babel) }%
2693       \message{\{#1}%
2694     \endgroup}
2695   \def\bbl@info#1{%
2696     \begingroup
2697       \newlinechar=`^^J
2698       \def\{^^J}%
2699       \wlog{#1}%
2700     \endgroup}
2701 \else
2702   \def\bbl@error#1#2{%
2703     \begingroup
2704       \def\{\MessageBreak}%
2705       \PackageError{babel}{#1}{#2}%
2706     \endgroup}
2707   \def\bbl@warning#1{%
2708     \begingroup
2709       \def\{\MessageBreak}%
2710       \PackageWarning{babel}{#1}%
2711     \endgroup}
2712   \def\bbl@info#1{%
2713     \begingroup
2714       \def\{\MessageBreak}%
2715       \PackageInfo{babel}{#1}%
2716     \endgroup}
2717 \fi

```

```

2718 \@ifpackagewith{babel}{silent}
2719 {\let\bbl@info@gobble
2720 \let\bbl@warning@gobble}
2721 {}
2722 \def\bbl@nocaption{\protect\bbl@nocaption@i}
2723 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
2724 \global\@namedef{#2}{\textbf{?#1?}}%
2725 \@nameuse{#2}%
2726 \bbl@warning{%
2727 \@backslashchar#2 not set. Please, define\\%
2728 it in the preamble with something like:\\%
2729 \string\renewcommand\@backslashchar#2{..}\\%
2730 Reported}}
2731 \def\@nolanerr#1{%
2732 \bbl@error
2733 {You haven't defined the language #1\space yet}%
2734 {Your command will be ignored, type <return> to proceed}}
2735 \def\@nopatterns#1{%
2736 \bbl@warning
2737 {No hyphenation patterns were preloaded for\\%
2738 the language `#1' into the format.\\%
2739 Please, configure your TeX system to add them and\\%
2740 rebuild the format. Now I will use the patterns\\%
2741 preloaded for \bbl@nulllanguage\space instead}}
2742 \let\bbl@usehooks@gobbletwo
2743 </kernel>
2744 <*patterns>

```

12 Loading hyphenation patterns

The following code is meant to be read by \LaTeX 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.

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 \LaTeX the above scheme won't work. The reason is that \LaTeX 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 L^AT_EX 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.

```

2745 <<Make sure ProvidesFile is defined>>
2746 \ProvidesFile{hyphen.cfg}[\<<date>>] \<<version>> Babel hyphens]
2747 \xdef\bbbl@format{\jobname}
2748 \ifx\AtBeginDocument\@undefined
2749   \def\@empty{}
2750   \let\orig@dump\dump
2751   \def\dump{%
2752     \ifx\@ztryfc\@undefined
2753       \else
2754         \toks0=\expandafter{\@preamblecmds}%
2755         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
2756         \def\@begindocumenthook{}%
2757       \fi
2758       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
2759 \fi
2760 <<Define core switching macros>>

```

`\process@line` Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with `=`. When the first token of a line is an `=`, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

```

2761 \def\process@line#1#2 #3 #4 {%
2762   \ifx=#1%
2763     \process@synonym{#2}%
2764   \else
2765     \process@language{#1#2}{#3}{#4}%
2766   \fi
2767   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bbbl@languages` is also set to empty.

```

2768 \toks@{}
2769 \def\bbbl@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.

```

2770 \def\process@synonym#1{%
2771   \ifnum\last@language=\m@ne
2772     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
2773   \else
2774     \expandafter\chardef\csname l@#1\endcsname\last@language
2775     \wlog{\string\l@#1=\string\language\the\last@language}%
2776     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
2777       \csname\languagenamename hyphenmins\endcsname
2778     \let\bbbl@elt\relax

```

```

2779 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
2780 \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

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

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

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

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

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form `\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

2781 \def\process@language#1#2#3{%
2782 \expandafter\addlanguage\csname l@#1\endcsname
2783 \expandafter\language\csname l@#1\endcsname
2784 \edef\language#1}%
2785 \bbl@hook@everylanguage{#1}%
2786 \bbl@get@enc#1::\@@@
2787 \begingroup
2788 \lefthyphenmin\m@ne
2789 \bbl@hook@loadpatterns{#2}%
2790 \ifnum\lefthyphenmin=\m@ne
2791 \else
2792 \expandafter\xdef\csname #1hyphenmins\endcsname{%
2793 \the\lefthyphenmin\the\righthyphenmin}%
2794 \fi
2795 \endgroup
2796 \def\bbl@tempa{#3}%
2797 \ifx\bbl@tempa\@empty\else
2798 \bbl@hook@loadexceptions{#3}%
2799 \fi
2800 \let\bbl@elt\relax
2801 \edef\bbl@languages{%
2802 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
2803 \ifnum\the\language=\z@

```

```

2804 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2805 \set@hyphenmins\tw@\thr@@\relax
2806 \else
2807 \expandafter\expandafter\expandafter\set@hyphenmins
2808 \csname #1hyphenmins\endcsname
2809 \fi
2810 \the\toks@
2811 \toks@{}}%
2812 \fi}

```

`\bbl@get@enc` The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. It uses delimited arguments to achieve this.

```

2813 \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.

```

2814 \def\bbl@hook@everylanguage#1{}
2815 \def\bbl@hook@loadpatterns#1{\input #1\relax}
2816 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
2817 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
2818 \begingroup
2819 \def\AddBabelHook#1#2{%
2820 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
2821 \def\next{\toks1}%
2822 \else
2823 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
2824 \fi
2825 \next}
2826 \ifx\directlua\undefined
2827 \ifx\XeTeXinputencoding\undefined\else
2828 \input xebabel.def
2829 \fi
2830 \else
2831 \input luababel.def
2832 \fi
2833 \openin1 = babel-\bbl@format.cfg
2834 \ifeof1
2835 \else
2836 \input babel-\bbl@format.cfg\relax
2837 \fi
2838 \closein1
2839 \endgroup
2840 \bbl@hook@loadkernel{switch.def}

```

`\readconfigfile` The configuration file can now be opened for reading.

```

2841 \openin1 = language.dat

```

See if the file exists, if not, use the default hyphenation file `hyphen.tex`. The user will be informed about this.

```

2842 \def\languagename{english}%
2843 \ifeof1
2844 \message{I couldn't find the file language.dat,\space
2845 I will try the file hyphen.tex}
2846 \input hyphen.tex\relax
2847 \chardef\l@english\z@
2848 \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`.

```
2849 \last@language@m@ne
```

We now read lines from the file until the end is found

```
2850 \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.

```
2851 \endlinechar@m@ne
```

```
2852 \read1 to \bbl@line
```

```
2853 \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.

```
2854 \if T\ifeof1F\fi T\relax
```

```
2855 \ifx\bbl@line\@empty\else
```

```
2856 \edef\bbl@line{\bbl@line\space\space\space}%
```

```
2857 \expandafter\process@line\bbl@line\relax
```

```
2858 \fi
```

```
2859 \repeat
```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns.

```
2860 \begingroup
```

```
2861 \def\bbl@elt#1#2#3#4{%
```

```
2862 \global\language=#2\relax
```

```
2863 \gdef\languagename{#1}%
```

```
2864 \def\bbl@elt##1##2##3##4{}}%
```

```
2865 \bbl@languages
```

```
2866 \endgroup
```

```
2867 \fi
```

and close the configuration file.

```
2868 \closein1
```

We add a message about the fact that babel is loaded in the format and with which language patterns to the `\everyjob` register.

```
2869 \if/\the\toks@\else
```

```
2870 \errhelp{language.dat loads no language, only synonyms}
```

```
2871 \errmessage{Orphan language synonym}
```

```
2872 \fi
```

Also remove some macros from memory and raise an error if `\toks@` is not empty. Finally load `switch.def`, but the latter is not required and the line inputting it may be commented out.

```
2873 \let\bbl@line\@undefined
```

```
2874 \let\process@line\@undefined
```

```
2875 \let\process@synonym\@undefined
```

```
2876 \let\process@language\@undefined
```

```
2877 \let\bbl@get@enc\@undefined
```

```
2878 \let\bbl@hyph@enc\@undefined
```

```
2879 \let\bbl@tempa\@undefined
```

```
2880 \let\bbl@hook@loadkernel\@undefined
```

```
2881 \let\bbl@hook@everylanguage\@undefined
```



```

2882 \let\bbl@hook@loadpatterns\@undefined
2883 \let\bbl@hook@loadexceptions\@undefined
2884 \end{patterns}

```

Here the code for `iniTEX` ends.

13 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].

```

2885 <<{*More package options}>> ≡
2886 \ifodd\bbl@engine
2887   \DeclareOption{bidi=basic-r}%
2888     {\ExecuteOptions{bidi=basic}}
2889   \DeclareOption{bidi=basic}%
2890     {\let\bbl@beforeforeign\leavevmode
2891       \newattribute\bbl@attr@dir
2892       \bbl@exp{\output{\bodydir\pagedir\the\output}}%
2893       \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
2894 \else
2895   \DeclareOption{bidi=basic-r}%
2896     {\ExecuteOptions{bidi=basic}}
2897   \DeclareOption{bidi=basic}%
2898     {\bbl@error
2899       {The bidi method `basic' is available only in\\%
2900         luatex. I'll continue with `bidi=default', so\\%
2901         expect wrong results}%
2902       {See the manual for further details.}%
2903       \let\bbl@beforeforeign\leavevmode
2904       \AtEndOfPackage{%
2905         \EnableBabelHook{babel-bidi}%
2906         \bbl@xebidipar}}
2907 \fi
2908 \DeclareOption{bidi=default}%
2909   {\let\bbl@beforeforeign\leavevmode
2910     \ifodd\bbl@engine
2911       \newattribute\bbl@attr@dir
2912       \bbl@exp{\output{\bodydir\pagedir\the\output}}%
2913     \fi
2914     \AtEndOfPackage{%
2915       \EnableBabelHook{babel-bidi}%
2916       \ifodd\bbl@engine\else
2917         \bbl@xebidipar
2918       \fi}}
2919 <</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.

```

2920 <<{*Font selection}>> ≡
2921 \bbl@trace{Font handling with fontspec}
2922 \@onlypreamble\babelfont
2923 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
2924   \edef\bbl@tempa{#1}%
2925   \def\bbl@tempb{#2}%
2926   \ifx\fontspec\@undefined
2927     \usepackage{fontspec}%

```

```

2928 \fi
2929 \EnableBabelHook{babel-fontspec}%
2930 \bbl@bblfont}
2931 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname
2932 \bbl@ifunset{\bbl@tempb family}{\bbl@providfam{\bbl@tempb}}{}}%
2933 \bbl@ifunset{\bbl@lsys@languagenam}{\bbl@provid@lsys{languagenam}}{}}%
2934 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
2935 {\bbl@csarg\edef{\bbl@tempb dflt@}<{#1}{#2}}% save bbl@rmdflt@
2936 \bbl@exp{%
2937 \let\bbl@bbl@tempb dflt@\languagenam>\<bbl@bbl@tempb dflt@>%
2938 \\\bbl@font@set\bbl@bbl@tempb dflt@\languagenam>%
2939 \<bbl@tempb default>\<bbl@tempb family>}}%
2940 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
2941 \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:

```

2942 \def\bbl@providfam#1{%
2943 \bbl@exp{%
2944 \\\newcommand\<#1default>{}% Just define it
2945 \\\bbl@add@list\\bbl@font@fams{#1}%
2946 \\\DeclareRobustCommand\<#1family>{%
2947 \\\not@math@alphabet\<#1family>\relax
2948 \\\fontfamily\<#1default>\\selectfont}%
2949 \\\DeclareTextFontCommand{\<text#1>}\<#1family>}}

```

The following macro is activated when the hook babel-fontspec is enabled.

```

2950 \def\bbl@switchfont{%
2951 \bbl@ifunset{\bbl@lsys@languagenam}{\bbl@provid@lsys{languagenam}}{}}%
2952 \bbl@exp{% eg Arabic -> arabic
2953 \lowercase{\edef\\bbl@tempa{\bbl@cs{sname@languagenam}}}}%
2954 \bbl@foreach\bbl@font@fams{%
2955 \bbl@ifunset{\bbl@##1dflt@languagenam}% (1) language?
2956 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
2957 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
2958 {}}% 123=F - nothing!
2959 {\bbl@exp{% 3=T - from generic
2960 \global\let\bbl@##1dflt@languagenam>%
2961 \<bbl@##1dflt@>}}}%
2962 {\bbl@exp{% 2=T - from script
2963 \global\let\bbl@##1dflt@languagenam>%
2964 \<bbl@##1dflt@*\bbl@tempa>}}}%
2965 {}}% 1=T - language, already defined
2966 \def\bbl@tempa{%
2967 \bbl@warning{The current font is not a standard family:\\%
2968 \fontname\font\\%
2969 Script and Language are not applied. Consider defining a\\%
2970 new family with \string\babelfont. Reported}}%
2971 \bbl@foreach\bbl@font@fams{% don't gather with prev for
2972 \bbl@ifunset{\bbl@##1dflt@languagenam}%
2973 {\bbl@cs{famrst@##1}%
2974 \global\bbl@csarg\let{famrst@##1}\relax}%
2975 {\bbl@exp{% order is relevant
2976 \\\bbl@add\\originalTeX{%
2977 \\\bbl@font@rst{\bbl@cs{##1dflt@languagenam}}%
2978 \<##1default>\<##1family>{##1}}%
2979 \\\bbl@font@set\bbl@##1dflt@languagenam>% the main part!
2980 \<##1default>\<##1family>}}}%
2981 \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. We must deactivate temporarily `\bbl@mapselect` because `\selectfont` is called internally when a font is defined.

```

2982 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
2983 \bbl@xin@{<>}{#1}%
2984 \fin@
2985 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1}%
2986 \fi
2987 \bbl@exp{%
2988 \def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
2989 \bbl@ifsamestring{#2}{\f@family}{\#3\let\bbl@tempa\relax}{}}
2990 \def\bbl@fontspec@set#1#2#3{% eg \bbl@rmdflt@lang fnt-opt fnt-nme
2991 \let\bbl@tempe\bbl@mapselect
2992 \let\bbl@mapselect\relax
2993 \bbl@exp{\<fontspec_set_family:Nnn>\#1%
2994 {\bbl@cs{lsys@\languagename},#2}{#3}%
2995 \let\bbl@mapselect\bbl@tempe
2996 \bbl@toggle#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.

```

2997 \def\bbl@font@rst#1#2#3#4{%
2998 \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`.

```

2999 \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 :-).

```

3000 \newcommand\babelFSstore[2][]{%
3001 \bbl@ifblank{#1}%
3002 {\bbl@csarg\def{sname@#2}{Latin}}%
3003 {\bbl@csarg\def{sname@#2}{#1}}%
3004 \bbl@provide@dirs{#2}%
3005 \bbl@csarg\ifnum{wdir@#2}>\z@
3006 \let\bbl@beforeforeign\leavevmode
3007 \EnableBabelHook{babel-bidi}%
3008 \fi
3009 \bbl@foreach{#2}{%
3010 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
3011 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
3012 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
3013 \def\bbl@FSstore#1#2#3#4{%
3014 \bbl@csarg\edef{#2default#1}{#3}%
3015 \expandafter\addto\csname extras#1\endcsname{%
3016 \let#4#3%
3017 \ifx#3\f@family
3018 \edef#3{\csname bbl@#2default#1\endcsname}%
3019 \fontfamily{#3}\selectfont
3020 \else
3021 \edef#3{\csname bbl@#2default#1\endcsname}%
3022 \fi}%
3023 \expandafter\addto\csname noextras#1\endcsname{%
3024 \ifx#3\f@family
3025 \fontfamily{#4}\selectfont

```

```

3026   \fi
3027   \let#3#4}}
3028 \let\bbbl@langfeatures\@empty
3029 \def\babelFSfeatures{% make sure \fontspec is redefined once
3030   \let\bbbl@ori@fontspec\fontspec
3031   \renewcommand\fontspec[1][]{%
3032     \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
3033   \let\babelFSfeatures\bbbl@FSfeatures
3034   \babelFSfeatures}
3035 \def\bbbl@FSfeatures#1#2{%
3036   \expandafter\addto\csname extras#1\endcsname{%
3037     \babel@save\bbbl@langfeatures
3038     \edef\bbbl@langfeatures{#2,}}
3039 <</Font selection>>

```

14 Hooks for XeTeX and LuaTeX

14.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

L^AT_EX 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 L^AT_EX. Anyway, for consistency LuaL^AT_EX also resets the catcodes.

```

3040 <<{*Restore Unicode catcodes before loading patterns}>> ≡
3041   \begingroup
3042     % Reset chars "80-"C0 to category "other", no case mapping:
3043     \catcode\@=11 \count@=128
3044     \loop\ifnum\count@<192
3045       \global\uccode\count@=0 \global\lccode\count@=0
3046       \global\catcode\count@=12 \global\sfcode\count@=1000
3047       \advance\count@ by 1 \repeat
3048     % Other:
3049     \def\O ##1 {%
3050       \global\uccode"##1=0 \global\lccode"##1=0
3051       \global\catcode"##1=12 \global\sfcode"##1=1000 }%
3052     % Letter:
3053     \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3054       \global\uccode"##1="##2
3055       \global\lccode"##1="##3
3056       % Uppercase letters have sfcode=999:
3057       \ifnum"##1="##3 \else \global\sfcode"##1=999 \fi }%
3058     % Letter without case mappings:
3059     \def\l ##1 {\L ##1 ##1 ##1 }%
3060     \l 00AA
3061     \L 00B5 039C 00B5
3062     \l 00BA
3063     \O 00D7
3064     \l 00DF
3065     \O 00F7
3066     \L 00FF 0178 00FF
3067   \endgroup
3068   \input #1\relax
3069 <</Restore Unicode catcodes before loading patterns>>

```

Some more common code.

```

3070 <<(*Footnote changes)>> ≡
3071 \bbl@trace{Bidi footnotes}
3072 \ifx\bbl@beforeforeign\leavevmode
3073   \def\bbl@footnote#1#2#3{%
3074     \@ifnextchar[%
3075       {\bbl@footnote@o{#1}{#2}{#3}}%
3076       {\bbl@footnote@x{#1}{#2}{#3}}}]
3077   \def\bbl@footnote@x#1#2#3#4{%
3078     \bgroup
3079     \select@language@x{\bbl@main@language}%
3080     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3081     \egroup}
3082   \def\bbl@footnote@o#1#2#3[#4]#5{%
3083     \bgroup
3084     \select@language@x{\bbl@main@language}%
3085     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3086     \egroup}
3087   \def\bbl@footnotetext#1#2#3{%
3088     \@ifnextchar[%
3089       {\bbl@footnotetext@o{#1}{#2}{#3}}%
3090       {\bbl@footnotetext@x{#1}{#2}{#3}}}]
3091   \def\bbl@footnotetext@x#1#2#3#4{%
3092     \bgroup
3093     \select@language@x{\bbl@main@language}%
3094     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3095     \egroup}
3096   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3097     \bgroup
3098     \select@language@x{\bbl@main@language}%
3099     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3100     \egroup}
3101   \def\BabelFootnote#1#2#3#4{%
3102     \ifx\bbl@fn@footnote\@undefined
3103       \let\bbl@fn@footnote\footnote
3104     \fi
3105     \ifx\bbl@fn@footnotetext\@undefined
3106       \let\bbl@fn@footnotetext\footnotetext
3107     \fi
3108     \bbl@ifblank{#2}%
3109     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3110      \@namedef{\bbl@stripslash#1text}%
3111       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3112     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3113      \@namedef{\bbl@stripslash#1text}%
3114       {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}]
3115 \fi
3116 <</Footnote changes>>

```

Now, the code.

```

3117 (*xetex)
3118 \def\BabelStringsDefault{unicode}
3119 \let\xebbl@stop\relax
3120 \AddBabelHook{xetex}{encodedcommands}{%
3121   \def\bbl@tempa{#1}%
3122   \ifx\bbl@tempa\@empty
3123     \XeTeXinputencoding"bytes"%
3124   \else
3125     \XeTeXinputencoding"#1"%
3126   \fi

```

```

3127 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3128 \AddBabelHook{xetex}{stopcommands}{%
3129 \xebbl@stop
3130 \let\xebbl@stop\relax}
3131 \AddBabelHook{xetex}{loadkernel}{%
3132 <<Restore Unicode catcodes before loading patterns>>}}
3133 \ifx\DisableBabelHook\@undefined\endinput\fi
3134 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3135 \DisableBabelHook{babel-fontspec}
3136 <<Font selection>>
3137 \input txtbabel.def
3138 </xetex>

```

14.2 Layout

In progress.

Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks). At least at this stage, babel will not do it and therefore a package like bidi (by Vafa Khalighi) would be necessary to overcome the limitations of xetex. Any help in making babel and bidi collaborate will be welcome, although the underlying concepts in both packages seem very different. Note also elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titles, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T_EX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdftex and xetex.

```

3139 (*texxet)
3140 \bbl@trace{Redefinitions for bidi layout}
3141 \def\bbl@sspre@caption{%
3142 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
3143 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
3144 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3145 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3146 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3147 \def\@hangfrom#1{%
3148 \setbox\@tempboxa\hbox{#1}}%
3149 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3150 \noindent\box\@tempboxa}
3151 \def\raggedright{%
3152 \let\@centercr
3153 \bbl@startskip\z@skip
3154 \@rightskip\@flushglue
3155 \bbl@endskip\@rightskip
3156 \parindent\z@
3157 \parfillskip\bbl@startskip}
3158 \def\raggedleft{%
3159 \let\@centercr
3160 \bbl@startskip\@flushglue
3161 \bbl@endskip\z@skip
3162 \parindent\z@
3163 \parfillskip\bbl@endskip}
3164 \fi
3165 \IfBabelLayout{lists}
3166 {\def\list#1#2{%
3167 \ifnum \@listdepth >5\relax

```

```

3168     \@toodeep
3169 \else
3170     \global\advance\@listdepth\@ne
3171 \fi
3172 \rightmargin\z@
3173 \listparindent\z@
3174 \itemindent\z@
3175 \csname @list\romannumeral\the\@listdepth\endcsname
3176 \def\@itemlabel{#1}%
3177 \let\makelabel\@mklab
3178 \@nmbrrlistfalse
3179 #2\relax
3180 \@trivlist
3181 \parskip\parsep
3182 \parindent\listparindent
3183 \advance\linewidth-\rightmargin
3184 \advance\linewidth-\leftmargin
3185 \advance\@totalleftmargin
3186     \ifcase\@bbl@thepardir\leftmargin\else\rightmargin\fi
3187 \parshape\@ne\@totalleftmargin\linewidth
3188 \ignorespaces}%
3189 \ifcase\@bbl@engine
3190     \def\labelenumii{\theenumii}%
3191     \def\p@enumii{\p@enumii}\theenumii}%
3192 \fi
3193 \def\@verbatim{%
3194     \trivlist \item\relax
3195     \if@minipage\else\vskip\parskip\fi
3196     \bbl@startskip\textwidth
3197     \advance\bbl@startskip-\linewidth
3198     \bbl@endskip\z@skip
3199     \parindent\z@
3200     \parfillskip\@flushglue
3201     \parskip\z@skip
3202     \@@par
3203     \language\@nohyphenation
3204     \@tempswafalse
3205     \def\par{%
3206         \if@tempswa
3207             \leavevmode\null
3208             \@@par\penalty\interlinepenalty
3209         \else
3210             \@tempwatrue
3211             \ifhmode\@@par\penalty\interlinepenalty\fi
3212         \fi}%
3213     \let\do\@makeother \dospecials
3214     \obeylines \verbatim@font \@noligs
3215     \everypar\expandafter{\the\everypar\unpenalty}}
3216 {}
3217 \IfBabelLayout{contents}
3218 {\def\@dottedtocline#1#2#3#4#5{%
3219     \ifnum#1>\c@tocdepth\else
3220     \vskip \z@ \@plus.2\p@
3221     {\bbl@startskip#2\relax
3222     \bbl@endskip\@tocrmarg
3223     \parfillskip-\bbl@endskip
3224     \parindent#2\relax
3225     \@afterindenttrue
3226     \interlinepenalty\M

```

```

3227     \leavevmode
3228     \@tempdima#3\relax
3229     \advance\bbl@startskip\@tempdima
3230     \null\nobreak\hskip-\bbl@startskip
3231     {#4}\nobreak
3232     \leaders\hbox{%
3233         $\m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
3234     \hfill\nobreak
3235     \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
3236     \par}%
3237     \fi}}
3238 {}
3239 \IfBabelLayout{columns}
3240 {\def\@outputdblcol{%
3241     \if@firstcolumn
3242         \global\@firstcolumnfalse
3243         \global\setbox\@leftcolumn\copy\@outputbox
3244         \splitmaxdepth\maxdimen
3245         \vbadness\maxdimen
3246         \setbox\@outputbox\box{\unvbox\@outputbox\unskip}%
3247         \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3248         \toks\@expandafter{\topmark}%
3249         \xdef\@firstcoltopmark{\the\toks@}%
3250         \toks\@expandafter{\splitfirstmark}%
3251         \xdef\@firstcolfirstmark{\the\toks@}%
3252         \ifx\@firstcolfirstmark\@empty
3253             \global\let\@setmarks\relax
3254         \else
3255             \gdef\@setmarks{%
3256                 \let\firstmark\@firstcolfirstmark
3257                 \let\topmark\@firstcoltopmark}%
3258         \fi
3259     \else
3260         \global\@firstcolumntrue
3261         \setbox\@outputbox\box{%
3262             \hb@xt@\textwidth{%
3263                 \hskip\columnwidth
3264                 \hfil
3265                 {\normalcolor\vrule \@width\columnseprule}%
3266                 \hfil
3267                 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3268                 \hskip-\textwidth
3269                 \hb@xt@\columnwidth{\box\@outputbox \hss}%
3270                 \hskip\columnsep
3271                 \hskip\columnwidth}}%
3272         \@combinedblfloats
3273         \@setmarks
3274         \@outputpage
3275         \begingroup
3276             \dblfloatplacement
3277             \startdblcolumn
3278             \whilesw\if@colmade \fi{\@outputpage
3279                 \startdblcolumn}%
3280         \endgroup
3281     \fi}}%
3282 {}
3283 <<Footnote changes>>
3284 \IfBabelLayout{footnotes}%
3285 {\BabelFootnote\footnote\languagename}{}%

```



```

3286 \BabelFootnote\localfootnote\language\language{}{}%
3287 \BabelFootnote\mainfootnote{}{}{}
3288 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

3289 \IfBabelLayout{counters}%
3290 {\let\bbl@latinarabic=@arabic
3291 \def@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3292 \let\bbl@asciroman=@roman
3293 \def@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3294 \let\bbl@asciiRoman=@Roman
3295 \def@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
3296 </texxet>

```

14.3 LuaTeX

The new loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). For the moment, a dangerous approach is used – just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

```

3297 (*luatex)
3298 \ifx\AddBabelHook\undefined
3299 \bbl@trace{Read language.dat}
3300 \begingroup
3301 \toks@{}
3302 \count@% 0=start, 1=0th, 2=normal
3303 \def\bbl@process@line#1#2 #3 #4 {%
3304 \ifx=#1%

```

```

3305     \bbl@process@synonym{#2}%
3306     \else
3307     \bbl@process@language{#1#2}{#3}{#4}%
3308     \fi
3309     \lignorespaces}
3310 \def\bbl@manylang{%
3311     \ifnum\bbl@last>\@ne
3312     \bbl@info{Non-standard hyphenation setup}%
3313     \fi
3314     \let\bbl@manylang\relax}
3315 \def\bbl@process@language#1#2#3{%
3316     \ifcase\count@
3317     \@ifundefined{zth#1}{\count@tw@}{\count@\@ne}%
3318     \or
3319     \count@tw@
3320     \fi
3321     \ifnum\count@=\tw@
3322     \expandafter\addlanguage\csname l@#1\endcsname
3323     \language\allocationnumber
3324     \chardef\bbl@last\allocationnumber
3325     \bbl@manylang
3326     \let\bbl@elt\relax
3327     \xdef\bbl@languages{%
3328     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3329     \fi
3330     \the\toks@
3331     \toks@{}}
3332 \def\bbl@process@synonym@aux#1#2{%
3333     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3334     \let\bbl@elt\relax
3335     \xdef\bbl@languages{%
3336     \bbl@languages\bbl@elt{#1}{#2}{}}}%
3337 \def\bbl@process@synonym#1{%
3338     \ifcase\count@
3339     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3340     \or
3341     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3342     \else
3343     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3344     \fi}
3345 \ifx\bbl@languages@\undefined % Just a (sensible?) guess
3346     \chardef\l@english\z@
3347     \chardef\l@USenglish\z@
3348     \chardef\bbl@last\z@
3349     \global\@namedef{\bbl@hyphendata@0}{\hyphen.tex}{}
3350     \gdef\bbl@languages{%
3351     \bbl@elt{english}{0}{\hyphen.tex}{}%
3352     \bbl@elt{USenglish}{0}{}}
3353     \else
3354     \global\let\bbl@languages@format\bbl@languages
3355     \def\bbl@elt#1#2#3#4{% Remove all except language 0
3356     \ifnum#2>\z@\else
3357     \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3358     \fi}%
3359     \xdef\bbl@languages{\bbl@languages}%
3360     \fi
3361     \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
3362     \bbl@languages
3363     \openin1=language.dat

```

```

3364 \ifeof1
3365 \bbl@warning{I couldn't find language.dat. No additional\%
3366           patterns loaded. Reported}%
3367 \else
3368 \loop
3369 \endlinechar\m@ne
3370 \read1 to \bbl@line
3371 \endlinechar`\^^M
3372 \if T\ifeof1F\fi T\relax
3373 \ifx\bbl@line\empty\else
3374 \edef\bbl@line{\bbl@line\space\space\space}%
3375 \expandafter\bbl@process@line\bbl@line\relax
3376 \fi
3377 \repeat
3378 \fi
3379 \endgroup
3380 \bbl@trace{Macros for reading patterns files}
3381 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
3382 \ifx\babelcatcodetablenum\undefined
3383 \def\babelcatcodetablenum{5211}
3384 \fi
3385 \def\bbl@luapatterns#1#2{%
3386 \bbl@get@enc#1::\@@@
3387 \setbox\z@\hbox\bgroup
3388 \begingroup
3389 \ifx\catcodetable\undefined
3390 \let\savecatcodetable\luatexsavecatcodetable
3391 \let\initcatcodetable\luatexinitcatcodetable
3392 \let\catcodetable\luatexcatcodetable
3393 \fi
3394 \savecatcodetable\babelcatcodetablenum\relax
3395 \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3396 \catcodetable\numexpr\babelcatcodetablenum+1\relax
3397 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3398 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
3399 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3400 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3401 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
3402 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
3403 \input #1\relax
3404 \catcodetable\babelcatcodetablenum\relax
3405 \endgroup
3406 \def\bbl@tempa{#2}%
3407 \ifx\bbl@tempa\empty\else
3408 \input #2\relax
3409 \fi
3410 \egroup}%
3411 \def\bbl@patterns@lua#1{%
3412 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3413 \csname l@#1\endcsname
3414 \edef\bbl@tempa{#1}%
3415 \else
3416 \csname l@#1:\f@encoding\endcsname
3417 \edef\bbl@tempa{#1:\f@encoding}%
3418 \fi\relax
3419 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3420 \@ifundefined{bbl@hyphendata@the\language}%
3421 {\def\bbl@elt##1##2##3##4{%
3422 \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...

```

```

3423     \def\bbl@tempb{##3}%
3424     \ifx\bbl@tempb\@empty\else % if not a synonymous
3425         \def\bbl@tempc{##3}{##4}}%
3426     \fi
3427     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3428     \fi}%
3429 \bbl@languages
3430 \@ifundefined{bbl@hyphendata@the\language}%
3431     {\bbl@info{No hyphenation patterns were set for\%
3432         language '\bbl@tempa'. Reported}}%
3433     {\expandafter\expandafter\expandafter\bbl@luapatterns
3434         \csname bbl@hyphendata@the\language\endcsname}}}}
3435 \endinput\fi
3436 \begingroup
3437 \catcode`\%=12
3438 \catcode`\'=12
3439 \catcode`\ "=12
3440 \catcode`\:=12
3441 \directlua{
3442 Babel = Babel or {}
3443 function Babel.bytes(line)
3444     return line:gsub(".",
3445         function (chr) return unicode.utf8.char(string.byte(chr)) end)
3446 end
3447 function Babel.begin_process_input()
3448     if luatexbase and luatexbase.add_to_callback then
3449         luatexbase.add_to_callback('process_input_buffer',
3450             Babel.bytes, 'Babel.bytes')
3451     else
3452         Babel.callback = callback.find('process_input_buffer')
3453         callback.register('process_input_buffer', Babel.bytes)
3454     end
3455 end
3456 function Babel.end_process_input ()
3457     if luatexbase and luatexbase.remove_from_callback then
3458         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3459     else
3460         callback.register('process_input_buffer', Babel.callback)
3461     end
3462 end
3463 function Babel.addpatterns(pp, lg)
3464     local lg = lang.new(lg)
3465     local pats = lang.patterns(lg) or ''
3466     lang.clear_patterns(lg)
3467     for p in pp:gmatch('[^%s]+') do
3468         ss = ''
3469         for i in string.utfcharacters(p:gsub('%d', '')) do
3470             ss = ss .. '%d?' .. i
3471         end
3472         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
3473         ss = ss:gsub('%.%d%?$', '%%.')
3474         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3475         if n == 0 then
3476             tex.sprint(
3477                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
3478                 .. p .. [[{}]])
3479             pats = pats .. ' ' .. p
3480         else
3481             tex.sprint(

```

```

3482         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
3483         .. p .. [{}]])
3484     end
3485 end
3486 lang.patterns(lg, pats)
3487 end
3488 }
3489 \endgroup
3490 \def\BabelStringsDefault{unicode}
3491 \let\luabbl@stop\relax
3492 \AddBabelHook{luatex}{encodedcommands}{%
3493 \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3494 \ifx\bbl@tempa\bbl@tempb\else
3495 \directlua{Babel.begin_process_input()}%
3496 \def\luabbl@stop{%
3497 \directlua{Babel.end_process_input()}}%
3498 \fi}%
3499 \AddBabelHook{luatex}{stopcommands}{%
3500 \luabbl@stop
3501 \let\luabbl@stop\relax}
3502 \AddBabelHook{luatex}{patterns}{%
3503 \@ifundefined{bbl@hyphendata@the\language}%
3504 {\def\bbl@elt##1##2##3##4{%
3505 \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3506 \def\bbl@tempb{##3}%
3507 \ifx\bbl@tempb@empty\else % if not a synonymous
3508 \def\bbl@tempc{##3}{##4}%
3509 \fi
3510 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3511 \fi}%
3512 \bbl@languages
3513 \@ifundefined{bbl@hyphendata@the\language}%
3514 {\bbl@info{No hyphenation patterns were set for\%
3515 language '#2'. Reported}}%
3516 {\expandafter\expandafter\expandafter\bbl@luapatterns
3517 \csname bbl@hyphendata@the\language\endcsname}}}%
3518 \@ifundefined{bbl@patterns@}{}%
3519 \begingroup
3520 \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
3521 \ifin@else
3522 \ifx\bbl@patterns@empty\else
3523 \directlua{ Babel.addpatterns(
3524 [[\bbl@patterns@]], \number\language) }%
3525 \fi
3526 \@ifundefined{bbl@patterns@#1}%
3527 @empty
3528 {\directlua{ Babel.addpatterns(
3529 [[\space\csname bbl@patterns@#1\endcsname]],
3530 \number\language) }}%
3531 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3532 \fi
3533 \endgroup}}
3534 \AddBabelHook{luatex}{everylanguage}{%
3535 \def\process@language##1##2##3{%
3536 \def\process@line####1####2 ####3 ####4 {}}
3537 \AddBabelHook{luatex}{loadpatterns}{%
3538 \input #1\relax
3539 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3540 {#1}{}}

```

```

3541 \AddBabelHook{luatex}{loadexceptions}{%
3542   \input #1\relax
3543   \def\bbl@tempb##1##2{{##1}{##2}}%
3544   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3545     {\expandafter\expandafter\expandafter\bbl@tempb
3546       \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.

```

3547 \@onlypreamble\babelpatterns
3548 \AtEndOfPackage{%
3549   \newcommand\babelpatterns[2][\@empty]{%
3550     \ifx\bbl@patterns@relax
3551       \let\bbl@patterns@\@empty
3552     \fi
3553     \ifx\bbl@pntnlist\@empty\else
3554       \bbl@warning{%
3555         You must not intermingle \string\selectlanguage\space and\%
3556         \string\babelpatterns\space or some patterns will not\%
3557         be taken into account. Reported}%
3558       \fi
3559     \ifx\@empty#1%
3560       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
3561     \else
3562       \edef\bbl@tempb{\zap@space#1 \@empty}%
3563       \bbl@for\bbl@tempa\bbl@tempb{%
3564         \bbl@fixname\bbl@tempa
3565         \bbl@iflanguage\bbl@tempa{%
3566           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3567             \@ifundefined{bbl@patterns@\bbl@tempa}%
3568               \@empty
3569               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3570             #2}}}%
3571     \fi}}

```

Common stuff.

```

3572 \AddBabelHook{luatex}{loadkernel}{%
3573   <<Restore Unicode catcodes before loading patterns>>}
3574 \ifx\DisableBabelHook\undefined\endinput\fi
3575 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3576 \DisableBabelHook{babel-fontspec}
3577 <<Font selection>>

```

14.4 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`.

```

3578 \bbl@trace{Redefinitions for bidi layout}

```

```

3579 \ifx\@eqnnum\undefined\else
3580 \ifx\bbl@attr@dir\undefined\else
3581 \edef\@eqnnum{%
3582 \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
3583 \unexpanded\expandafter{\@eqnnum}}
3584 \fi
3585 \fi
3586 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
3587 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3588 \def\bbl@nextfake#1{%
3589 \mathdir\bodydir % non-local, use always inside a group!
3590 \bbl@exp{%
3591 #1% Once entered in math, set boxes to restore values
3592 \everyvbox{%
3593 \the\everyvbox
3594 \bodydir\the\bodydir
3595 \mathdir\the\mathdir
3596 \everyhbox{\the\everyhbox}%
3597 \everyvbox{\the\everyvbox}}%
3598 \everyhbox{%
3599 \the\everyhbox
3600 \bodydir\the\bodydir
3601 \mathdir\the\mathdir
3602 \everyhbox{\the\everyhbox}%
3603 \everyvbox{\the\everyvbox}}}%
3604 \def\@hangfrom#1{%
3605 \setbox\@tempboxa\hbox{#{#1}}%
3606 \hangindent\wd\@tempboxa
3607 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3608 \shapemode\@ne
3609 \fi
3610 \noindent\box\@tempboxa}
3611 \fi
3612 \IfBabelLayout{tabular}
3613 {\def\@tabular{%
3614 \leavevmode\hbox\bgroup\bbl@nextfake$% %$
3615 \let\@acol\@tabacol \let\@classz\@tabclassz
3616 \let\@classiv\@tabclassiv \let\@tabularcr\@tabarray}}
3617 {}
3618 \IfBabelLayout{lists}
3619 {\def\list#1#2{%
3620 \ifnum \@listdepth >5\relax
3621 \@toodeep
3622 \else
3623 \global\advance\@listdepth\@ne
3624 \fi
3625 \rightmargin\z@
3626 \listparindent\z@
3627 \itemindent\z@
3628 \csname @list\romannumeral\the\@listdepth\endcsname
3629 \def\itemlabel{#1}%
3630 \let\makelabel\@mklab
3631 \@nbrlistfalse
3632 #2\relax
3633 \@trivlist
3634 \parskip\parsep
3635 \parindent\listparindent
3636 \advance\linewidth -\rightmargin
3637 \advance\linewidth -\leftmargin

```

```

3638 \advance\@totalleftmargin \leftmargin
3639 \parshape \@ne
3640 \@totalleftmargin \linewidth
3641 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
3642 \shapemode\tw@
3643 \fi
3644 \ignorespaces}}
3645 {}

```

Implicitly reverses sectioning labels in `bidi=basic-r`, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes `bidi=basic-r`, but there are some additional readjustments for `bidi=default`.

```

3646 \IfBabelLayout{counters}%
3647 {\def\@textsuperscript#1{% lua has separate settings for math
3648 \m@th
3649 \mathdir\pagedir % required with basic-r; ok with default, too
3650 \ensuremath{^{\mbox {\fontsize \sf@size \z@ #1}}}}%
3651 \let\bbbl@latinarabic=\@arabic
3652 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
3653 \@ifpackagewith{babel}{bidi=default}%
3654 {\let\bbbl@asciroman=\@roman
3655 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
3656 \let\bbbl@asciiRoman=\@Roman
3657 \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
3658 \def\labelenumii{\theenumii}%
3659 \def\p@enumiii{\p@enumii}\theenumii{}}{}}{}
3660 <<Footnote changes>>
3661 \IfBabelLayout{footnotes}%
3662 {\BabelFootnote\footnote\languagename{}}{}%
3663 \BabelFootnote\localfootnote\languagename{}}{}%
3664 \BabelFootnote\mainfootnote{}}{}}{}
3665 {}

```

Some \LaTeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

3666 \IfBabelLayout{extras}%
3667 {\def\underline#1{%
3668 \relax
3669 \ifmode\@underline{#1}%
3670 \else\bbbl@nextfake$\@underline{\hbox{#1}}\m@th$\relax\fi}%
3671 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
3672 \if b\expandafter\car\@series\@nil\boldmath\fi
3673 \babelsublr{%
3674 \LaTeX\kern.15em2\bbbl@nextfake$_{\textstyle\varepsilon}$}}}}%
3675 {}
3676 </luatex>

```

14.5 Auto bidi with basic-r

The file `babel-bidi.lua` currently only contains data. It is 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.

TODO: math mode (as weak L?)

```
3677 (*basic-r)
3678 Babel = Babel or {}
3679
3680 require('babel-bidi.lua')
3681
3682 local characters = Babel.characters
3683 local ranges = Babel.ranges
3684
3685 local DIR = node.id("dir")
3686
3687 local function dir_mark(head, from, to, outer)
3688   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
3689   local d = node.new(DIR)
3690   d.dir = '+' .. dir
3691   node.insert_before(head, from, d)
3692   d = node.new(DIR)
3693   d.dir = '-' .. dir
3694   node.insert_after(head, to, d)
3695 end
3696
3697 function Babel.pre_otfload_v(head)
3698   -- head = Babel.numbers(head)
3699   head = Babel.bidi(head, true)
3700   return head
3701 end
3702
3703 function Babel.pre_otfload_h(head)
3704   -- head = Babel.numbers(head)
3705   head = Babel.bidi(head, false)
3706   return head
3707 end
3708
3709 function Babel.bidi(head, ispar)
3710   local first_n, last_n           -- first and last char with nums
3711   local last_es                   -- an auxiliary 'last' used with nums
```

```

3712 local first_d, last_d          -- first and last char in L/R block
3713 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):

```

3714 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
3715 local strong_lr = (strong == 'l') and 'l' or 'r'
3716 local outer = strong
3717
3718 local new_dir = false
3719 local first_dir = false
3720
3721 local last_lr
3722
3723 local type_n = ''
3724
3725 for item in node.traverse(head) do
3726
3727   -- three cases: glyph, dir, otherwise
3728   if item.id == node.id'glyph'
3729     or (item.id == 7 and item.subtype == 2) then
3730
3731     local itemchar
3732     if item.id == 7 and item.subtype == 2 then
3733       itemchar = item.replace.char
3734     else
3735       itemchar = item.char
3736     end
3737     local chardata = characters[itemchar]
3738     dir = chardata and chardata.d or nil
3739     if not dir then
3740       for nn, et in ipairs(ranges) do
3741         if itemchar < et[1] then
3742           break
3743         elseif itemchar <= et[2] then
3744           dir = et[3]
3745           break
3746         end
3747       end
3748     end
3749     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.

```

3750   if new_dir then
3751     attr_dir = 0
3752     for at in node.traverse(item.attr) do
3753       if at.number == luatexbase.registernumber'bbl@attr@dir' then
3754         attr_dir = at.value % 3
3755       end
3756     end
3757     if attr_dir == 1 then
3758       strong = 'r'
3759     elseif attr_dir == 2 then
3760       strong = 'al'
3761     else

```

```

3762     strong = 'l'
3763     end
3764     strong_lr = (strong == 'l') and 'l' or 'r'
3765     outer = strong_lr
3766     new_dir = false
3767 end
3768
3769     if dir == 'nsm' then dir = strong end           -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

3770     dir_real = dir           -- We need dir_real to set strong below
3771     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:

```

3772     if strong == 'al' then
3773         if dir == 'en' then dir = 'an' end           -- W2
3774         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
3775         strong_lr = 'r'                               -- W3
3776     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

3777     elseif item.id == node.id'dir' then
3778         new_dir = true
3779         dir = nil
3780     else
3781         dir = nil           -- Not a char
3782     end

```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```

3783     if dir == 'en' or dir == 'an' or dir == 'et' then
3784         if dir ~= 'et' then
3785             type_n = dir
3786         end
3787         first_n = first_n or item
3788         last_n = last_es or item
3789         last_es = nil
3790     elseif dir == 'es' and last_n then -- W3+W6
3791         last_es = item
3792     elseif dir == 'cs' then           -- it's right - do nothing
3793     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
3794         if strong_lr == 'r' and type_n ~= '' then
3795             dir_mark(head, first_n, last_n, 'r')
3796         elseif strong_lr == 'l' and first_d and type_n == 'an' then
3797             dir_mark(head, first_n, last_n, 'r')
3798             dir_mark(head, first_d, last_d, outer)
3799             first_d, last_d = nil, nil
3800         elseif strong_lr == 'l' and type_n ~= '' then
3801             last_d = last_n
3802         end
3803         type_n = ''
3804         first_n, last_n = nil, nil
3805     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:

```

3806   if dir == 'l' or dir == 'r' then
3807     if dir ~= outer then
3808       first_d = first_d or item
3809       last_d = item
3810     elseif first_d and dir ~= strong_lr then
3811       dir_mark(head, first_d, last_d, outer)
3812       first_d, last_d = nil, nil
3813     end
3814   end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

3815   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
3816     item.char = characters[item.char] and
3817               characters[item.char].m or item.char
3818   elseif (dir or new_dir) and last_lr ~= item then
3819     local mir = outer .. strong_lr .. (dir or outer)
3820     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
3821       for ch in node.traverse(node.next(last_lr)) do
3822         if ch == item then break end
3823         if ch.id == node.id'glyph' then
3824           ch.char = characters[ch.char].m or ch.char
3825         end
3826       end
3827     end
3828   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).

```

3829   if dir == 'l' or dir == 'r' then
3830     last_lr = item
3831     strong = dir_real           -- Don't search back - best save now
3832     strong_lr = (strong == 'l') and 'l' or 'r'
3833   elseif new_dir then
3834     last_lr = nil
3835   end
3836 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

3837   if last_lr and outer == 'r' then
3838     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
3839       ch.char = characters[ch.char].m or ch.char
3840     end
3841   end
3842   if first_n then
3843     dir_mark(head, first_n, last_n, outer)
3844   end
3845   if first_d then
3846     dir_mark(head, first_d, last_d, outer)
3847   end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```
3848 return node.prev(head) or head
3849 end
3850 (/basic-r)
```

And here the Lua code for bidi=basic:

```
3851 (*basic)
3852 Babel = Babel or {}
3853
3854 Babel.fontmap = Babel.fontmap or {}
3855 Babel.fontmap[0] = {} -- l
3856 Babel.fontmap[1] = {} -- r
3857 Babel.fontmap[2] = {} -- al/an
3858
3859 function Babel.pre_otfload_v(head)
3860 -- head = Babel.numbers(head)
3861 head = Babel.bidi(head, true)
3862 return head
3863 end
3864
3865 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
3866 -- head = Babel.numbers(head)
3867 head = Babel.bidi(head, false, dir)
3868 return head
3869 end
3870
3871 require('babel-bidi.lua')
3872
3873 local characters = Babel.characters
3874 local ranges = Babel.ranges
3875
3876 local DIR = node.id('dir')
3877 local GLYPH = node.id('glyph')
3878
3879 local function insert_implicit(head, state, outer)
3880 local new_state = state
3881 if state.sim and state.eim and state.sim ~= state.eim then
3882 dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
3883 local d = node.new(DIR)
3884 d.dir = '+' .. dir
3885 node.insert_before(head, state.sim, d)
3886 local d = node.new(DIR)
3887 d.dir = '-' .. dir
3888 node.insert_after(head, state.eim, d)
3889 end
3890 new_state.sim, new_state.eim = nil, nil
3891 return head, new_state
3892 end
3893
3894 local function insert_numeric(head, state)
3895 local new
3896 local new_state = state
3897 if state.san and state.ean and state.san ~= state.ean then
3898 local d = node.new(DIR)
3899 d.dir = '+TLT'
3900 _, new = node.insert_before(head, state.san, d)
3901 if state.san == state.sim then state.sim = new end
```

```

3902     local d = node.new(DIR)
3903     d.dir = '-TLT'
3904     _, new = node.insert_after(head, state.ean, d)
3905     if state.ean == state.eim then state.eim = new end
3906 end
3907 new_state.san, new_state.ean = nil, nil
3908 return head, new_state
3909 end
3910
3911 -- \hbox with an explicit dir can lead to wrong results
3912 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>
3913
3914 function Babel.bidi(head, ispar, hdir)
3915     local d -- d is used mainly for computations in a loop
3916     local prev_d = ''
3917     local new_d = false
3918
3919     local nodes = {}
3920     local outer_first = nil
3921
3922     local has_en = false
3923     local first_et = nil
3924
3925     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
3926
3927     local save_outer
3928     local temp = node.get_attribute(head, ATDIR)
3929     if temp then
3930         temp = temp % 3
3931         save_outer = (temp == 0 and 'l') or
3932                     (temp == 1 and 'r') or
3933                     (temp == 2 and 'al')
3934     elseif ispar then -- Or error? Shouldn't happen
3935         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
3936     else
3937         save_outer = ('TRT' == hdir) and 'r' or 'l'
3938     end
3939     local outer = save_outer
3940     local last = outer
3941     -- 'al' is only taken into account in the first, current loop
3942     if save_outer == 'al' then save_outer = 'r' end
3943
3944     local fontmap = Babel.fontmap
3945
3946     for item in node.traverse(head) do
3947
3948         -- In what follows, #node is the last (previous) node, because the
3949         -- current one is not added until we start processing the neutrals.
3950
3951         -- three cases: glyph, dir, otherwise
3952         if item.id == GLYPH
3953             or (item.id == 7 and item.subtype == 2) then
3954
3955             local d_font = nil
3956             local item_r
3957             if item.id == 7 and item.subtype == 2 then
3958                 item_r = item.replace -- automatic discs have just 1 glyph
3959             else
3960                 item_r = item

```

```

3961     end
3962     local chardata = characters[item_r.char]
3963     d = chardata and chardata.d or nil
3964     if not d or d == 'nsm' then
3965         for nn, et in ipairs(ranges) do
3966             if item_r.char < et[1] then
3967                 break
3968             elseif item_r.char <= et[2] then
3969                 if not d then d = et[3]
3970                 elseif d == 'nsm' then d_font = et[3]
3971                 end
3972             break
3973         end
3974     end
3975     end
3976     d = d or 'l'
3977     d_font = d_font or d
3978
3979     d_font = (d_font == 'l' and 0) or
3980             (d_font == 'nsm' and 0) or
3981             (d_font == 'r' and 1) or
3982             (d_font == 'al' and 2) or
3983             (d_font == 'an' and 2) or nil
3984     if d_font and fontmap and fontmap[d_font][item_r.font] then
3985         item_r.font = fontmap[d_font][item_r.font]
3986     end
3987
3988     if new_d then
3989         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
3990         attr_d = node.get_attribute(item, ATDIR)
3991         attr_d = attr_d % 3
3992         if attr_d == 1 then
3993             outer_first = 'r'
3994             last = 'r'
3995         elseif attr_d == 2 then
3996             outer_first = 'r'
3997             last = 'al'
3998         else
3999             outer_first = 'l'
4000             last = 'l'
4001         end
4002         outer = last
4003         has_en = false
4004         first_et = nil
4005         new_d = false
4006     end
4007
4008     elseif item.id == DIR then
4009         d = nil
4010         new_d = true
4011
4012     else
4013         d = nil
4014     end
4015
4016     -- AL <= EN/ET/ES      -- W2 + W3 + W6
4017     if last == 'al' and d == 'en' then
4018         d = 'an'          -- W3
4019     elseif last == 'al' and (d == 'et' or d == 'es') then

```

```

4020     d = 'on'           -- W6
4021 end
4022
4023 -- EN + CS/ES + EN    -- W4
4024 if d == 'en' and #nodes >= 2 then
4025     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4026         and nodes[#nodes-1][2] == 'en' then
4027         nodes[#nodes][2] = 'en'
4028     end
4029 end
4030
4031 -- AN + CS + AN      -- W4 too, because uax9 mixes both cases
4032 if d == 'an' and #nodes >= 2 then
4033     if (nodes[#nodes][2] == 'cs')
4034         and nodes[#nodes-1][2] == 'an' then
4035         nodes[#nodes][2] = 'an'
4036     end
4037 end
4038
4039 -- ET/EN             -- W5 + W7->l / W6->on
4040 if d == 'et' then
4041     first_et = first_et or (#nodes + 1)
4042 elseif d == 'en' then
4043     has_en = true
4044     first_et = first_et or (#nodes + 1)
4045 elseif first_et then -- d may be nil here !
4046     if has_en then
4047         if last == 'l' then
4048             temp = 'l' -- W7
4049         else
4050             temp = 'en' -- W5
4051         end
4052     else
4053         temp = 'on' -- W6
4054     end
4055     for e = first_et, #nodes do
4056         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4057     end
4058     first_et = nil
4059     has_en = false
4060 end
4061
4062 if d then
4063     if d == 'al' then
4064         d = 'r'
4065         last = 'al'
4066     elseif d == 'l' or d == 'r' then
4067         last = d
4068     end
4069     prev_d = d
4070     table.insert(nodes, {item, d, outer_first})
4071 else
4072     -- Not sure about the following. Looks too 'ad hoc', but it's
4073     -- required for numbers, so that 89 19 becomes 19 89. It also
4074     -- affects n+cs/es+n.
4075     if prev_d == 'an' or prev_d == 'en' then
4076         table.insert(nodes, {item, 'on', nil})
4077     end
4078 end

```



```

4079
4080     outer_first = nil
4081
4082 end
4083
4084 -- TODO -- repeated here in case EN/ET is the last node. Find a
4085 -- better way of doing things:
4086 if first_et then      -- dir may be nil here !
4087     if has_en then
4088         if last == 'l' then
4089             temp = 'l'    -- W7
4090         else
4091             temp = 'en'  -- W5
4092         end
4093     else
4094         temp = 'on'     -- W6
4095     end
4096     for e = first_et, #nodes do
4097         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4098     end
4099 end
4100
4101 -- dummy node, to close things
4102 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4103
4104 ----- NEUTRAL -----
4105
4106 outer = save_outer
4107 last = outer
4108
4109 local first_on = nil
4110
4111 for q = 1, #nodes do
4112     local item
4113
4114     local outer_first = nodes[q][3]
4115     outer = outer_first or outer
4116     last = outer_first or last
4117
4118     local d = nodes[q][2]
4119     if d == 'an' or d == 'en' then d = 'r' end
4120     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4121
4122     if d == 'on' then
4123         first_on = first_on or q
4124     elseif first_on then
4125         if last == d then
4126             temp = d
4127         else
4128             temp = outer
4129         end
4130         for r = first_on, q - 1 do
4131             nodes[r][2] = temp
4132             item = nodes[r][1]    -- MIRRORING
4133             if item.id == GLYPH and temp == 'r' then
4134                 item.char = characters[item.char].m or item.char
4135             end
4136         end
4137         first_on = nil

```

```

4138     end
4139
4140     if d == 'r' or d == 'l' then last = d end
4141 end
4142
4143 ----- IMPLICIT, REORDER -----
4144
4145 outer = save_outer
4146 last = outer
4147
4148 local state = {}
4149 state.has_r = false
4150
4151 for q = 1, #nodes do
4152
4153     local item = nodes[q][1]
4154
4155     outer = nodes[q][3] or outer
4156
4157     local d = nodes[q][2]
4158
4159     if d == 'nsm' then d = last end           -- W1
4160     if d == 'en' then d = 'an' end
4161     local isdir = (d == 'r' or d == 'l')
4162
4163     if outer == 'l' and d == 'an' then
4164         state.san = state.san or item
4165         state.ean = item
4166     elseif state.san then
4167         head, state = insert_numeric(head, state)
4168     end
4169
4170     if outer == 'l' then
4171         if d == 'an' or d == 'r' then      -- im -> implicit
4172             if d == 'r' then state.has_r = true end
4173             state.sim = state.sim or item
4174             state.eim = item
4175         elseif d == 'l' and state.sim and state.has_r then
4176             head, state = insert_implicit(head, state, outer)
4177         elseif d == 'l' then
4178             state.sim, state.eim, state.has_r = nil, nil, false
4179         end
4180     else
4181         if d == 'an' or d == 'l' then
4182             state.sim = state.sim or item
4183             state.eim = item
4184         elseif d == 'r' and state.sim then
4185             head, state = insert_implicit(head, state, outer)
4186         elseif d == 'r' then
4187             state.sim, state.eim = nil, nil
4188         end
4189     end
4190
4191     if isdir then
4192         last = d           -- Don't search back - best save now
4193     elseif d == 'on' and state.san then
4194         state.san = state.san or item
4195         state.ean = item
4196     end

```

```

4197
4198 end
4199
4200 return node.prev(head) or head
4201 end
4202 </basic>

```

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.

```

4203 <*nil>
4204 \ProvidesLanguage{nil}[<<date>> <<version>> Nil language]
4205 \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.

```

4206 \ifx\l@nohyphenation\@undefined
4207   \@nopatterns{nil}
4208   \adddialect\l@nil0
4209 \else
4210   \let\l@nil\l@nohyphenation
4211 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

4212 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```

\captionnil
\datenil 4213 \let\captionnil\@empty
4214 \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.

```

4215 \ldf@finish{nil}
4216 </nil>

```

16 Support for Plain T_EX (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 T_EX-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTeX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing `iniTeX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
4217 (*bplain | blplain)
4218 \catcode`\{=1 % left brace is begin-group character
4219 \catcode`\}=2 % right brace is end-group character
4220 \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...

```
4221 \openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
4222 \ifeof0
4223 \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).

```
4224 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
4225 \def\input #1 {%
4226   \let\input\input\input\input
4227   \input #1 hyphen.cfg
```

Once that's done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
4228   \let\input\undefined
4229 }
4230 \fi
4231 </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`.

```
4232 (bplain)\input plain.tex
4233 (blplain)\input 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.

```
4234 (bplain)\def\fmtname{babel-plain}
4235 (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 $\LaTeX 2_{\epsilon}$ that are needed for `babel`.

```
4236 (*plain)
4237 \def\@empty{}
4238 \def\loadlocalcfg#1{%
```

```

4239 \openin0#1.cfg
4240 \ifeof0
4241 \closein0
4242 \else
4243 \closein0
4244 {\immediate\write16{*****}%}
4245 \immediate\write16{* Local config file #1.cfg used}%
4246 \immediate\write16{*}%
4247 }
4248 \input #1.cfg\relax
4249 \fi
4250 \@endoflfd}

```

16.3 General tools

A number of \LaTeX macro's that are needed later on.

```

4251 \long\def\@firstofone#1{#1}
4252 \long\def\@firstoftwo#1#2{#1}
4253 \long\def\@secondoftwo#1#2{#2}
4254 \def\@nnil{\@nil}
4255 \def\@gobbletwo#1#2{}
4256 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4257 \def\@star@or@long#1{%
4258 \@ifstar
4259 {\let\l@ngrel@x\relax#1}%
4260 {\let\l@ngrel@x\long#1}}
4261 \let\l@ngrel@x\relax
4262 \def\@car#1#2\@nil{#1}
4263 \def\@cdr#1#2\@nil{#2}
4264 \let\@typeset@protect\relax
4265 \let\protected@edef\edef
4266 \long\def\@gobble#1{}
4267 \edef\@backslashchar{\expandafter\@gobble\string\}
4268 \def\strip@prefix#1>{}
4269 \def\g@addto@macro#1#2{%
4270 \toks@\expandafter{#1#2}%
4271 \xdef#1{\the\toks@}}
4272 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4273 \def\@nameuse#1{\csname #1\endcsname}
4274 \def\@ifundefined#1{%
4275 \expandafter\ifx\csname#1\endcsname\relax
4276 \expandafter\@firstoftwo
4277 \else
4278 \expandafter\@secondoftwo
4279 \fi}
4280 \def\@expandtwoargs#1#2#3{%
4281 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4282 \def\zap@space#1 #2{%
4283 #1%
4284 \ifx#2\@empty\else\expandafter\zap@space\fi
4285 #2}

```

$\LaTeX 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

4286 \ifx\@preamblecmds\undefined
4287 \def\@preamblecmds{}
4288 \fi
4289 \def\@onlypreamble#1{%

```

```

4290 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4291   \@preamblecmds\do#1}}
4292 \@onlypreamble\@onlypreamble

```

Mimick L^AT_EX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```

4293 \def\begindocument{%
4294   \@begindocumenthook
4295   \global\let\@begindocumenthook\@undefined
4296   \def\do#1{\global\let##1\@undefined}%
4297   \@preamblecmds
4298   \global\let\do\noexpand}

4299 \ifx\@begindocumenthook\@undefined
4300   \def\@begindocumenthook{}
4301 \fi
4302 \@onlypreamble\@begindocumenthook
4303 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick L^AT_EX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```

4304 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
4305 \@onlypreamble\AtEndOfPackage
4306 \def\@endofldf{}
4307 \@onlypreamble\@endofldf
4308 \let\bbl@afterlang\@empty
4309 \chardef\bbl@opt@hyphenmap\z@

```

L^AT_EX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

4310 \ifx\if@filesw\@undefined
4311   \expandafter\let\csname if@filesw\expandafter\endcsname
4312     \csname iffalse\endcsname
4313 \fi

```

Mimick L^AT_EX's commands to define control sequences.

```

4314 \def\newcommand{\@star@or@long\new@command}
4315 \def\new@command#1{%
4316   \@testopt{\@newcommand#1}0}
4317 \def\@newcommand#1[#2]{%
4318   \@ifnextchar [{\@xargdef#1[#2]}%
4319     {\@argdef#1[#2]}}
4320 \long\def\@argdef#1[#2]#3{%
4321   \@yargdef#1\@ne{#2}{#3}}
4322 \long\def\@xargdef#1[#2][#3]#4{%
4323   \expandafter\def\expandafter#1\expandafter{%
4324     \expandafter\@protected@testopt\expandafter #1%
4325     \csname\string#1\expandafter\endcsname{#3}}%
4326   \expandafter\@yargdef \csname\string#1\endcsname
4327     \tw@{#2}{#4}}
4328 \long\def\@yargdef#1#2#3{%
4329   \@tempcnta#3\relax
4330   \advance \@tempcnta \@ne
4331   \let\@hash@\relax
4332   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4333   \@tempcntb #2%
4334   \@whilenum\@tempcntb <\@tempcnta
4335   \do{%
4336     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%

```

```

4337 \advance\@tempcntb \@ne}%
4338 \let\@hash@##%
4339 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
4340 \def\providecommand{\@star@or@long\provide@command}
4341 \def\provide@command#1{%
4342 \begingroup
4343 \escapechar@m@ne\xdef\@gtempa{\string#1}%
4344 \endgroup
4345 \expandafter\@ifundefined\@gtempa
4346 {\def\reserved@a{\new@command#1}}%
4347 {\let\reserved@a\relax
4348 \def\reserved@a{\new@command\reserved@a}}%
4349 \reserved@a}%

4350 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4351 \def\declare@robustcommand#1{%
4352 \edef\reserved@a{\string#1}%
4353 \def\reserved@b{#1}%
4354 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4355 \edef#1{%
4356 \ifx\reserved@a\reserved@b
4357 \noexpand\x@protect
4358 \noexpand#1%
4359 \fi
4360 \noexpand\protect
4361 \expandafter\new@command\csname\bbl@stripslash#1 \endcsname
4362 }%
4363 \expandafter\new@command\csname\bbl@stripslash#1 \endcsname
4364 }
4365 \def\x@protect#1{%
4366 \ifx\protect\@typeset@protect\else
4367 \@x@protect#1%
4368 \fi
4369 }
4370 \def\@x@protect#1\fi#2#3{%
4371 \fi\protect#1%
4372 }

```

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`.

```

4373 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4374 \ifx\in@\undefined
4375 \def\in@#1#2{%
4376 \def\in@##1#1##2##3\in@{%
4377 \ifx\in@##2\in@false\else\in@true\fi}%
4378 \in@#2#1\in@\in@}
4379 \else
4380 \let\bbl@tempa\empty
4381 \fi
4382 \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).

```
4383 \def\ifpackagewith#1#2#3#4{#3}
```

The L^AT_EX macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain T_EX but we need the macro to be defined as a no-op.

```
4384 \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 L^AT_EX 2_ε versions; just enough to make things work in plain T_EX environments.

```
4385 \ifx\@tempcnta\@undefined
4386   \csname newcount\endcsname\@tempcnta\relax
4387 \fi
4388 \ifx\@tempcntb\@undefined
4389   \csname newcount\endcsname\@tempcntb\relax
4390 \fi
```

To prevent wasting two counters in L^AT_EX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
4391 \ifx\bye\@undefined
4392   \advance\count10 by -2\relax
4393 \fi
4394 \ifx\@ifnextchar\@undefined
4395   \def\@ifnextchar#1#2#3{%
4396     \let\reserved@d=#1%
4397     \def\reserved@a{#2}\def\reserved@b{#3}%
4398     \futurelet\@let@token\@ifnch}
4399   \def\@ifnch{%
4400     \ifx\@let@token\@sptoken
4401       \let\reserved@c\@xifnch
4402     \else
4403       \ifx\@let@token\reserved@d
4404         \let\reserved@c\reserved@a
4405       \else
4406         \let\reserved@c\reserved@b
4407     \fi
4408   \fi
4409   \reserved@c}
4410   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
4411   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
4412 \fi
4413 \def\@testopt#1#2{%
4414   \@ifnextchar[#{#1}{#1[#2]}}
4415 \def\@protected@testopt#1{%
4416   \ifx\protect\@typeset@protect
4417     \expandafter\@testopt
4418   \else
4419     \@x@protect#1%
4420   \fi}
4421 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4422   #2\relax}\fi}
4423 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4424   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain T_EX environment.

```
4425 \def\DeclareTextCommand{%
4426   \@dec@text@cmd\providecommand
```



```

4427 }
4428 \def\ProvideTextCommand{%
4429   \@dec@text@cmd\providecommand
4430 }
4431 \def\DeclareTextSymbol#1#2#3{%
4432   \@dec@text@cmd\chardef#1{#2}#3\relax
4433 }
4434 \def\@dec@text@cmd#1#2#3{%
4435   \expandafter\def\expandafter#2%
4436     \expandafter{%
4437       \csname#3-cmd\expandafter\endcsname
4438       \expandafter#2%
4439       \csname#3\string#2\endcsname
4440     }%
4441 %   \let\@ifdefinable\rc@ifdefinable
4442   \expandafter#1\csname#3\string#2\endcsname
4443 }
4444 \def\@current@cmd#1{%
4445   \ifx\protect\@typeset@protect\else
4446     \noexpand#1\expandafter\@gobble
4447   \fi
4448 }
4449 \def\@changed@cmd#1#2{%
4450   \ifx\protect\@typeset@protect
4451     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4452       \expandafter\ifx\csname ?\string#1\endcsname\relax
4453         \expandafter\def\csname ?\string#1\endcsname{%
4454           \@changed@x@err{#1}%
4455         }%
4456       \fi
4457     \global\expandafter\let
4458       \csname\cf@encoding \string#1\expandafter\endcsname
4459       \csname ?\string#1\endcsname
4460     \fi
4461     \csname\cf@encoding\string#1%
4462     \expandafter\endcsname
4463   \else
4464     \noexpand#1%
4465   \fi
4466 }
4467 \def\@changed@x@err#1{%
4468   \errhelp{Your command will be ignored, type <return> to proceed}%
4469   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4470 \def\DeclareTextCommandDefault#1{%
4471   \DeclareTextCommand#1?%
4472 }
4473 \def\ProvideTextCommandDefault#1{%
4474   \ProvideTextCommand#1?%
4475 }
4476 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
4477 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4478 \def\DeclareTextAccent#1#2#3{%
4479   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
4480 }
4481 \def\DeclareTextCompositeCommand#1#2#3#4{%
4482   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
4483   \edef\reserved@b{\string##1}%
4484   \edef\reserved@c{%
4485     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%

```

```

4486 \ifx\reserved@b\reserved@c
4487   \expandafter\expandafter\expandafter\ifx
4488     \expandafter\@car\reserved@a\relax\relax\@nil
4489     \@text@composite
4490   \else
4491     \edef\reserved@b##1{%
4492       \def\expandafter\noexpand
4493         \csname#2\string#1\endcsname###1{%
4494         \noexpand\@text@composite
4495         \expandafter\noexpand\csname#2\string#1\endcsname
4496         ###1\@empty\@empty\@text@composite
4497         {##1}%
4498       }%
4499     }%
4500     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
4501   \fi
4502   \expandafter\def\csname\expandafter\string\csname
4503     #2\endcsname\string#1-\string#3\endcsname{#4}
4504 \else
4505   \errhelp{Your command will be ignored, type <return> to proceed}%
4506   \errmessage{\string\DeclareTextCompositeCommand\space used on
4507     inappropriate command \protect#1}
4508 \fi
4509 }
4510 \def\@text@composite#1#2#3\@text@composite{%
4511   \expandafter\@text@composite@x
4512     \csname\string#1-\string#2\endcsname
4513 }
4514 \def\@text@composite@x#1#2{%
4515   \ifx#1\relax
4516     #2%
4517   \else
4518     #1%
4519   \fi
4520 }
4521 %
4522 \def\@strip@args#1:#2-#3\@strip@args{#2}
4523 \def\DeclareTextComposite#1#2#3#4{%
4524   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
4525   \bgroup
4526     \lcode` \@=#4%
4527     \lowercase{%
4528   \egroup
4529     \reserved@a @%
4530   }%
4531 }
4532 %
4533 \def\UseTextSymbol#1#2{%
4534 %   \let\@curr@enc\cf@encoding
4535 %   \@use@text@encoding{#1}%
4536 %   #2%
4537 %   \@use@text@encoding\@curr@enc
4538 }
4539 \def\UseTextAccent#1#2#3{%
4540 %   \let\@curr@enc\cf@encoding
4541 %   \@use@text@encoding{#1}%
4542 %   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
4543 %   \@use@text@encoding\@curr@enc
4544 }

```

```

4545 \def\@use@text@encoding#1{%
4546 % \edef\font@encoding{#1}%
4547 % \xdef\font@name{%
4548 % \csname\curr@fontshape/\font@size\endcsname
4549 % }%
4550 % \pickup@font
4551 % \font@name
4552 % \@@enc@update
4553 }
4554 \def\DeclareTextSymbolDefault#1#2{%
4555 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
4556 }
4557 \def\DeclareTextAccentDefault#1#2{%
4558 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
4559 }
4560 \def\cf@encoding{OT1}

```

Currently we only use the $\LaTeX 2_\epsilon$ method for accents for those that are known to be made active in *some* language definition file.

```

4561 \DeclareTextAccent{"}{OT1}{127}
4562 \DeclareTextAccent{'}{OT1}{19}
4563 \DeclareTextAccent{^}{OT1}{94}
4564 \DeclareTextAccent{`}{OT1}{18}
4565 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```

4566 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
4567 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
4568 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
4569 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
4570 \DeclareTextSymbol{\i}{OT1}{16}
4571 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```

4572 \ifx\scriptsize\@undefined
4573 \let\scriptsize\sevenrm
4574 \fi
4575 </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 \TeX book*, 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.