

Babel

Version 3.33.1716

2019/07/28

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 , 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, line breaking and so on are being added incrementally.

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

Contents

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 luatex	7
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	24
1.15	Modifying a language	26
1.16	Creating a language	26
1.17	Digits	29
1.18	Getting the current language name	29
1.19	Hyphenation and line breaking	30
1.20	Selecting scripts	31
1.21	Selecting directions	32
1.22	Language attributes	36
1.23	Hooks	36
1.24	Languages supported by babel with ldf files	38
1.25	Unicode character properties in luatex	39
1.26	Tips, workarounds, know issues and notes	39
1.27	Current and future work	40
1.28	Tentative and experimental code	41
2	Loading languages with language.dat	41
2.1	Format	42
3	The interface between the core of babel and the language definition files	42
3.1	Guidelines for contributed languages	44
3.2	Basic macros	44
3.3	Skeleton	45
3.4	Support for active characters	46
3.5	Support for saving macro definitions	47
3.6	Support for extending macros	47
3.7	Macros common to a number of languages	47
3.8	Encoding-dependent strings	48
4	Changes	51
4.1	Changes in babel version 3.9	51
II	Source code	52
5	Identification and loading of required files	52

6	locale directory	52
7	Tools	53
7.1	Multiple languages	57
8	The Package File (\LaTeX, babel.sty)	58
8.1	base	58
8.2	key=value options and other general option	60
8.3	Conditional loading of shorthands	62
8.4	Language options	63
9	The kernel of Babel (babel.def, common)	66
9.1	Tools	66
9.2	Hooks	68
9.3	Setting up language files	70
9.4	Shorthands	72
9.5	Language attributes	81
9.6	Support for saving macro definitions	84
9.7	Short tags	84
9.8	Hyphens	85
9.9	Multiencoding strings	86
9.10	Macros common to a number of languages	92
9.11	Making glyphs available	92
9.11.1	Quotation marks	93
9.11.2	Letters	94
9.11.3	Shorthands for quotation marks	95
9.11.4	Umlauts and tremas	96
9.12	Layout	97
9.13	Load engine specific macros	98
9.14	Creating languages	98
10	The kernel of Babel (babel.def, only \LaTeX)	108
10.1	The redefinition of the style commands	108
10.2	Cross referencing macros	108
10.3	Marks	112
10.4	Preventing clashes with other packages	113
10.4.1	ifthen	113
10.4.2	varioref	114
10.4.3	hhline	114
10.4.4	hyperref	115
10.4.5	fancyhdr	115
10.5	Encoding and fonts	115
10.6	Basic bidi support	117
10.7	Local Language Configuration	120
11	Multiple languages (switch.def)	121
11.1	Selecting the language	122
11.2	Errors	130
12	Loading hyphenation patterns	132
13	Font handling with fontspec	137

14	Hooks for XeTeX and LuaTeX	141
14.1	XeTeX	141
14.2	Layout	143
14.3	LuaTeX	144
14.4	Southeast Asian scripts	150
14.5	CJK line breaking	152
14.6	Layout	154
14.7	Auto bidi with basic and basic-r	156
15	Data for CJK	167
16	The ‘nil’ language	167
17	Support for Plain T_EX (plain.def)	167
17.1	Not renaming hyphen.tex	167
17.2	Emulating some L ^A T _E X features	168
17.3	General tools	169
17.4	Encoding related macros	172
18	Acknowledgements	175

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’	8
Argument of \language@active@arg” has an extra }	11
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	25

Part I

User guide

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

1 The user interface

1.1 Monolingual documents

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

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

```
\documentclass{article}

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

\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 list the required languages as package 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, but it is discouraged except if there a real reason to do so:

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

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

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

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

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

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

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

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

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

```
\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.

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

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

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

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

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

```
\documentclass{article}

\usepackage[russian]{babel}

\babelfont{rm}{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{<language>}`) is deprecated and you will get the error:²

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

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

- Another typical error when using babel is the following:³

```
! 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:

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

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

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

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

`\foreignlanguage` `{\language}{\text}`

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

1.8 Auxiliary language selectors

`\begin{otherlanguage}` `{\language} ... \end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\selectlanguage`, except 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*}` `{\language} ... \end{otherlanguage*}`

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

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

`\begin{hyphenrules}` `{\language} ... \end{hyphenrules}`

The environment `hyphenrules` can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’, provided that in `language.dat` the ‘language’ nohyphenation is defined by loading `zerohyph.tex`. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, `hyphenrules` is discouraged and `otherlanguage*` (the starred version) is preferred, as the former does not take into account possible changes in

encodings of characters like, say, ' done by some languages (eg, italian, french, ukraineb). To set hyphenation exceptions, use `\babelhyphenation` (see below).

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

```
\babelensure{polish}
```

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

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

They are activated when the language is selected (at the `afterextras` event), and it makes some assumptions which could not be fulfilled in some languages. Note also you should include only macros defined by the language, not global macros (eg, `\TeX` 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 discretionaries and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbccode`, 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` $\{\langle shorthands-list \rangle\}$

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

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

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

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

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

\usesshorthands `*{\<char>}`

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

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

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

\defineshorthand `[\<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:

```
\usesshorthands*{"}
\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:

⁵With it encoded string may not work as expected.

```
\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 as user shorthands in the preamble with, for example, `\usesshorthands` or `\usesshorthands*`.)

Very often, this is a more convenient way to deactivate shorthands than `\shorthandoff`, 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

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

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

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

1.11 Package options

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

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.

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

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

activegrave Same for `.

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

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

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

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

safe= none | ref | bib

Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions – 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= $\langle file \rangle$

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

main= $\langle language \rangle$

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

headfoot= $\langle language \rangle$

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

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

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

nocase	New 3.9l Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code>) are ignored. Use only if there are incompatibilities with other packages.
silent	New 3.9l No warnings and no <i>infos</i> are written to the log file. ⁹
strings=	generic unicode encoded $\langle label \rangle$ $\langle font\ encoding \rangle$ Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional \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 <code>\MakeUppercase</code> and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
hyphenmap=	off main select other other* New 3.9g Sets the 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 <code>\begin{document}</code> }, but also the first <code>\selectlanguage</code> in the preamble), and it's the default if a single language option has been stated; ¹¹ select sets it only at <code>\selectlanguage</code> ; other also sets it at <code>otherlanguage</code> ; other* also sets it at <code>otherlanguage*</code> as well as in heads and foots (if the option <code>headfoot</code> is used) and in auxiliary files (ie, at <code>\select@language</code>), and it's the default if several language options have been stated. The option <code>first</code> can be regarded as an optimized version of <code>other*</code> for monolingual documents. ¹²
bidi=	default basic basic-r bidi-l bidi-r New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.21.
layout=	New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.21.

1.12 The base option

With this package option `babel` just loads some basic macros (those in `switch.def`), defines `\AfterBabelLanguage` and exits. It also selects the 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` $\{ \langle option-name \rangle \} \{ \langle code \rangle \}$

⁹You can use alternatively the package `silence`.

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.

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

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

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

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

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

1.13 ini files

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

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

```
\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

East Asia scripts Internal inconsistencies in script and language names must be sorted out, so you may need to set them explicitly in `\babel font`, as well as `CJKShape`. `luatex` does basic line breaking, but currently `xetex` does not (you may load `zhspacing`). Although for a few words and short texts the `ini` files should be fine, CJK texts are best set with a dedicated framework (CJK, `luatexja`, `kotex`, `CTeX`...), . Actually, this is what the `ldf` does in `japanese` with `luatex`, because the following piece of code loads `luatexja`:

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

18

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	zh-Hans-HK	Chinese
se	Northern Sami ^{ul}	zh-Hans-MO	Chinese
seh	Sena	zh-Hans-SG	Chinese
ses	Koyraboro Senni	zh-Hans	Chinese
sg	Sango	zh-Hant-HK	Chinese
shi-Latn	Tachelhit	zh-Hant-MO	Chinese
shi-Tfng	Tachelhit	zh-Hant	Chinese
shi	Tachelhit	zh	Chinese
si	Sinhala	zu	Zulu
sk	Slovak ^{ul}		

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

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

filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda

konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole

nynorsk	serbian-latin-bosniaherzegovina
occitan	serbian-latin-kosovo
oriya	serbian-latin-montenegro
oromo	serbian-latin
ossetic	serbian-latn-ba
pashto	serbian-latn-me
persian	serbian-latn-xk
piedmontese	serbian-latn
polish	serbian
portuguese-br	shambala
portuguese-brazil	shona
portuguese-portugal	sichuanyi
portuguese-pt	sinhala
portuguese	slovak
punjabi-arab	slovene
punjabi-arabic	slovenian
punjabi-gurmukhi	soga
punjabi-guru	somali
punjabi	spanish-mexico
quechua	spanish-mx
romanian	spanish
romansh	standardmoroccantamazight
rombo	swahili
rundi	swedish
russian	swissgerman
rwa	tachelhit-latin
sakha	tachelhit-latn
samburu	tachelhit-tfng
samin	tachelhit-tifinagh
sango	tachelhit
sangu	taita
sanskrit-beng	tamil
sanskrit-bengali	tasawaq
sanskrit-deva	telugu
sanskrit-devanagari	teso
sanskrit-gujarati	thai
sanskrit-gujr	tibetan
sanskrit-kannada	tigrinya
sanskrit-knda	tongan
sanskrit-malayalam	turkish
sanskrit-mlym	turkmen
sanskrit-telu	ukenglish
sanskrit-telugu	ukrainian
sanskrit	upporsorbian
scottishgaelic	urdu
sena	usenglish
serbian-cyrillic-bosniaherzegovina	usorbian
serbian-cyrillic-kosovo	uyghur
serbian-cyrillic-montenegro	uzbek-arab
serbian-cyrillic	uzbek-arabic
serbian-cyrl-ba	uzbek-cyrillic
serbian-cyrl-me	uzbek-cyrl
serbian-cyrl-xk	uzbek-latin
serbian-cyrl	uzbek-latn

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

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]{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}
```

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

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

EXAMPLE Here is how to do it:

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

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

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

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

This makes sure the OpenType script for Devanagari is `deva` and not `dev2` (luatex does not detect automatically the correct script¹⁴). You may also pass some options to `fontspec`: with `silent`, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).

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

NOTE `\fontspec` is not touched at all, only the preset font families (`rm`, `sf`, `tt`, and the like). If a language is switched when an *ad hoc* font is active, or you select the font with this command, neither the script nor the language 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 \TeX conventions to set the basic families – define `\xxdefault`, and activate it with `\xxfamily`. On the other hand, `\setxxxxfont` in `fontspec` takes a different approach, because `\xxfamily` is redefined with the family name hardcoded (so that `\xxdefault` becomes no-op). Of course, both methods are incompatible, and if you use `\setxxxxfont`, font switching with `\babelfont` just does *not* work (nor the standard `\xxdefault`, for that matter). As of **New 3.34** there is an attempt to make them compatible, but the language system will not be set by babel.

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

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.

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

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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.

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

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

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

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

captions= \langle language-tag \rangle

Loads only the strings. For example:

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

hyphenrules= \langle language-list \rangle

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

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

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

A special value is +, which allocates a new language (in the 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= \langle script-name \rangle

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

language= \langle language-name \rangle

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

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

mapfont= direction

Assigns the font for the writing direction of this language (only with bidi=basic).¹⁵ More precisely, what mapfont=direction means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right.¹⁶ So, there should be at most 3 directives of this kind.

¹⁵There will be another value, language, not yet implemented.

¹⁶In future releases an new value (script) will be added.

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

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

intrapenalty= $\langle penalty \rangle$

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

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

1.17 Digits

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

For example:

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

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

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

1.18 Getting the current language name

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

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

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

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

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

1.19 Hyphenation and line breaking

`\babelhyphen` $\ast\{\langle type \rangle\}$

`\babelhyphen` $\ast\{\langle text \rangle\}$

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, Portugese, 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 provide with a set of basic “hyphens” which can be used by themselves, to define a user shorthand, or even in language files.

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.
- `\babelhyphen{repeat}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portugese and Spanish.
- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).
- `\babelhyphen{empty}` inserts a break oportunity without a hyphen at all.
- `\babelhyphen{\langle text \rangle}` is a hard “hyphen” using $\langle text \rangle$ 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 `\babelexhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

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

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

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

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

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

New 3.9m *In luatex only*,¹⁷ adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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

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

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

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

1.20 Selecting scripts

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

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

¹⁷With 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.

¹⁸The 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.

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.¹⁹

`\ensureascii` $\langle text \rangle$

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

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

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

1.21 Selecting directions

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

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

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

There are some package options controlling bidi writing.

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

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

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

¹⁹But still defined for backwards compatibility.

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

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

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

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

```
\documentclass{article}

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

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[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{fuṣḥā l-ʿaṣr} (MSA) and
    \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

NOTE Boxes are “black boxes”. Numbers inside an `\hbox` (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 `\texthe` must be defined to select the main language):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

Digits in `pdfTeX` must be marked up explicitly (unlike `luatex` with `bidi=basic` or `bidi=basic-r` and, usually, `xetex`). This command is provided to set `{\langle lr-text \rangle}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `rl` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

```
RTL A ltr text \thechapter{} and still ltr RTL B
```

There are *three* R blocks and *two* L blocks, and the order is *RTL B and still ltr 1 ltr text RTL A*. This is by design to provide the proper behavior in the most usual cases — but if you need to use `\ref` in an L text inside R, the L text must be marked up explicitly; for example:

```
RTL A \foreignlanguage{english}{ltr text \thechapter{} and still ltr} RTL B
```

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

```
\BabelFootnote{\enfootnote}{english}\{(\{)\}
```

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

1.22 Language attributes

`\languageattribute`

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

1.23 Hooks

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

`\AddBabelHook`

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

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{<name>}`, `\DisableBabelHook{<name>}`. Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also

applied to an specific language with the optional argument; language specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three T_EX 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<language>`. This event and the next one should not contain language-dependent code (for that, add it to `\extras<language>`).

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

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

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

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

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

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

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

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

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

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

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

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

1.24 Languages supported by babel with ldf files

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

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

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

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

preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension .dn:

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

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

1.25 Unicode character properties in luatex

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

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

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

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

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

1.26 Tips, workarounds, know issues and notes

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

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

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

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


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

(A recent version of inputenc is required.)

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

1.27 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).

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

²¹This explains why L^AT_EX 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.

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.

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

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

1.28 Tentative and experimental code

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

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}
```

2 Loading languages with language.dat

T_EX and most engines based on it (pdfT_EX, xetex, ϵ -T_EX, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, L^AT_EX, XeL^AT_EX, pdfL^AT_EX). 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 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

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

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.

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.

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

Some recommendations:

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

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

²⁷ But not removed, for backward compatibility.

3.1 Guidelines for contributed languages

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

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

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

`\extras<lang>` The macro `\extras<lang>` contains all the extra definitions needed for a specific language.

	This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
<code>\noextras<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@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \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>

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

```

```

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

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

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

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

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

`\declare@shorthand`

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

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

The T_EXbook states: “Plain T_EX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [2, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. L^AT_EX 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, `⟨curname⟩`, the control sequence for which the meaning has to be saved.

`\babel@savevariable`

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

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

3.6 Support for extending macros

`\addto`

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

3.7 Macros common to a number of languages

`\bbl@allowhyphens`

In several languages compound words are used. This means that when T_EX 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.

²⁸This mechanism was introduced by Bernd Raichle.

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

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

²⁹In future releases further categories may be added.

```

\SetString{\chaptername}{utf8-string}

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

\EndBabelCommands

```

A real example is:

```

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

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

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

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

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

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

\EndBabelCommands

```

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

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

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

maintainers of the current languages to decide if using it is appropriate.³⁰

\EndBabelCommands Marks the end of the series of blocks.

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

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

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

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

#1 is replaced by the roman numeral.

\SetCase $[\langle map-list \rangle]\{\langle toupper-code \rangle\}\{\langle tolower-code \rangle\}$
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would 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
```

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

```
\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

New 3.9g Case mapping serves in T_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{\uccode}{\lccode}` is similar to `\lccode` but it's ignored if the char has been set and saves the original `\lccode` to restore it when switching the language (except with `hyphenmap=first`).
- `\BabelLowerMM{\uccode-from}{\uccode-to}{\step}{\lccode-from}` loops through the given uppercase codes, using the step, and assigns them the `\lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{\uccode-from}{\uccode-to}{\step}{\lccode}` loops through the given uppercase codes, using the step, and assigns them the `\lccode`, which is fixed (MO stands for *many-to-one*).

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with `^` (if activated) and also if deactivated.

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

Part II

Source code

`babel` is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use `babel` only as documented (except, of course, if you want to explore and test them – you can post suggestions about multilingual issues to kadingira@tug.org 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 \TeX package, which sets options and loads language styles.

`plain.def` defines some \TeX 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=3.33.1716>>
2 <<date=2019/07/28>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8   {\def#1{#2}}%
9   {\expandafter\def\expandafter#1\expandafter{#1#2}}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@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` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³¹. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

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

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

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

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

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

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

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

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

```

61     \fi
62     \else
63     \expandafter \@firstoftwo
64     \fi}}

```

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

```

65 \def\bbl@ifblank#1{%
66   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
67 \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).

```

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

```

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

```

`\bbl@replace`

```

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

```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace `\elax` by `\ho`, then `\relax` becomes `\rho`). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in `\bbl@TG@@date`). It may change! (I'm not sure ckecking the replacement is really necessary or just paranoia).

```

100 \bbl@exp{\def\@bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
101   \def\bbl@tempa{#1}%

```



```

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

```

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

```

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

```

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

```

148 <<*Make sure ProvidesFile is defined>> ≡
149 \ifx\ProvidesFile\undefined
150 \def\ProvidesFile#1[#2 #3 #4]{%
151 \wlog{File: #1 #4 #3 <#2>}%

```

```

152 \let\ProvidesFile\@undefined}
153 \fi
154 <</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`.

```

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

```

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

```

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

```

167 <<(*Define core switching macros)>> ≡
168 \ifx\language\@undefined
169 \csname newcount\endcsname\language
170 \fi
171 <</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.

```

172 <<(*Define core switching macros)>> ≡
173 \ifx\newlanguage\@undefined
174 \csname newcount\endcsname\last@language
175 \def\addlanguage#1{%
176 \global\advance\last@language\@ne
177 \ifnum\last@language<\@ccclvi
178 \else
179 \errmessage{No room for a new \string\language!}%
180 \fi
181 \global\chardef#1\last@language
182 \wlog{\string#1 = \string\language\the\last@language}}

```

```

183 \else
184   \countdef\last@language=19
185   \def\addlanguage{\alloc@9\language\chardef\@cclvi}
186 \fi
187 <</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.

```

188 (*package)
189 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
190 \ProvidesPackage{babel}[\<<date>> \<<version>> The Babel package]
191 \@ifpackagewith{babel}{debug}
192   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
193    \let\bbl@debug\@firstofone}
194   {\providecommand\bbl@trace[1]{}%
195    \let\bbl@debug\@gobble}
196 \ifx\bbl@switchflag\undefined % Prevent double input
197   \let\bbl@switchflag\relax
198   \input switch.def\relax
199 \fi
200 <<Load patterns in luatex>>
201 <<Basic macros>>
202 \def\AfterBabelLanguage#1{%
203   \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.

```

204 \ifx\bbl@languages\undefined\else
205   \begingroup

```

```

206 \catcode`\^^I=12
207 \@ifpackagewith{babel}{showlanguages}{%
208 \begingroup
209 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
210 \wlog{<*languages>}%
211 \bbl@languages
212 \wlog{</languages>}%
213 \endgroup}{%
214 \endgroup
215 \def\bbl@elt#1#2#3#4{%
216 \ifnum#2=\z@
217 \gdef\bbl@nulllanguage{#1}%
218 \def\bbl@elt##1##2##3##4{}}%
219 \fi}%
220 \bbl@languages
221 \fi
222 \ifodd\bbl@engine
223 % Harftex is evolving, so the callback is not hardcoded, just in case
224 \def\bbl@harfpreline{Harf pre_linebreak_filter callback}%
225 \def\bbl@activate@preotf{%
226 \let\bbl@activate@preotf\relax % only once
227 \directlua{
228 Babel = Babel or {}
229 %
230 function Babel.pre_otfload_v(head)
231 if Babel.numbers and Babel.digits_mapped then
232 head = Babel.numbers(head)
233 end
234 if Babel.bidi_enabled then
235 head = Babel.bidi(head, false, dir)
236 end
237 return head
238 end
239 %
240 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
241 if Babel.numbers and Babel.digits_mapped then
242 head = Babel.numbers(head)
243 end
244 if Babel.fixboxdirs then % Temporary!
245 head = Babel.fixboxdirs(head)
246 end
247 if Babel.bidi_enabled then
248 head = Babel.bidi(head, false, dir)
249 end
250 return head
251 end
252 %
253 luatexbase.add_to_callback('pre_linebreak_filter',
254 Babel.pre_otfload_v,
255 'Babel.pre_otfload_v',
256 luatexbase.priority_in_callback('pre_linebreak_filter',
257 '\bbl@harfpreline')
258 or luatexbase.priority_in_callback('pre_linebreak_filter',
259 'luaotfload.node_processor')
260 or nil)
261 %
262 luatexbase.add_to_callback('hpack_filter',
263 Babel.pre_otfload_h,
264 'Babel.pre_otfload_h',

```

```

265     luatexbase.priority_in_callback('hpack_filter',
266     '\bbl@harfpreline')
267   or luatexbase.priority_in_callback('hpack_filter',
268     'luaotfload.node_processor')
269   or nil)
270 }%
271 \@ifpackageloaded{harfload}%
272   {\directlua{ Babel.mirroring_enabled = false }}%
273   {}}
274 \let\bbl@tempa\relax
275 \@ifpackagewith{babel}{bidi=basic}%
276   {\def\bbl@tempa{basic}}%
277   {\@ifpackagewith{babel}{bidi=basic-r}%
278     {\def\bbl@tempa{basic-r}}%
279     {}}
280 \ifx\bbl@tempa\relax\else
281   \let\bbl@beforeforeign\leavevmode
282   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
283   \RequirePackage{luatexbase}%
284   \directlua{
285     require('babel-data-bidi.lua')
286     require('babel-bidi-\bbl@tempa.lua')
287   }
288   \bbl@activate@preotf
289 \fi
290 \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.

```

291 \bbl@trace{Defining option 'base'}
292 \@ifpackagewith{babel}{base}{%
293   \ifx\directlua\undefined
294     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
295   \else
296     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
297   \fi
298   \DeclareOption{base}{}%
299   \DeclareOption{showlanguages}{}%
300   \ProcessOptions
301   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
302   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
303   \global\let@ifl@ter@@@ifl@ter
304   \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter@ifl@ter@@}%
305   \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.

```

306 \bbl@trace{key=value and another general options}
307 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
308 \def\bbl@tempb#1.#2{%
309   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
310 \def\bbl@tempd#1.#2\@nnil{%
311   \ifx\@empty#2%

```

```

312 \edef\bbl@tempc{\ifx\bbl@tempc@empty\else\bbl@tempc,\fi#1}%
313 \else
314 \in@{=}{#1}\ifin@
315 \edef\bbl@tempc{\ifx\bbl@tempc@empty\else\bbl@tempc,\fi#1.#2}%
316 \else
317 \edef\bbl@tempc{\ifx\bbl@tempc@empty\else\bbl@tempc,\fi#1}%
318 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
319 \fi
320 \fi}
321 \let\bbl@tempc@empty
322 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
323 \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.

```

324 \DeclareOption{KeepShorthandsActive}{}
325 \DeclareOption{activeacute}{}
326 \DeclareOption{activegrave}{}
327 \DeclareOption{debug}{}
328 \DeclareOption{noconfigs}{}
329 \DeclareOption{showlanguages}{}
330 \DeclareOption{silent}{}
331 \DeclareOption{mono}{}
332 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
333 <<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.

```

334 \let\bbl@opt@shorthands\@nnil
335 \let\bbl@opt@config\@nnil
336 \let\bbl@opt@main\@nnil
337 \let\bbl@opt@headfoot\@nnil
338 \let\bbl@opt@layout\@nnil

```

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

```

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

```

349 \let\bbl@language@opts@empty
350 \DeclareOption*{%
351 \bbl@xin@{\string=}{\CurrentOption}%

```

```

352 \ifin@
353 \expandafter\bbbl@tempa\CurrentOption\bbbl@tempa
354 \else
355 \bbbl@add@list\bbbl@language@opts{\CurrentOption}%
356 \fi}

```

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

```

357 \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 `\bbbl@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

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

```

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

```

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

```

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

```

379 \edef\bbbl@opt@shorthands{%
380 \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

381 \bbbl@ifshorthand{'}%
382 {\PassOptionsToPackage{activeacute}{babel}}{}
383 \bbbl@ifshorthand{`}%
384 {\PassOptionsToPackage{activegrave}{babel}}{}
385 \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.

```

386 \ifx\bbbl@opt@headfoot\@nnil\else

```

```

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

```

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

```

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

```

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

```

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

```

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

```

430 \def\bbl@try@load@lang#1#2#3{%

```



```

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

```

449 \ifx\bbl@opt@config\@nnil
450 \@ifpackagewith{babel}{noconfigs}}{}%
451 {\InputIfFileExists{bblopts.cfg}%
452 {\typeout{*****^^J%
453 * Local config file bblopts.cfg used^^J%
454 *}}%
455 {}}%
456 \else
457 \InputIfFileExists{\bbl@opt@config.cfg}%
458 {\typeout{*****^^J%
459 * Local config file \bbl@opt@config.cfg used^^J%
460 *}}%
461 {\bbl@error{%
462 Local config file '\bbl@opt@config.cfg' not found}%
463 Perhaps you misspelled it.}}%
464 \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.

```

465 \bbl@for\bbl@tempa\bbl@language@opts{%
466 \bbl@ifunset{ds@\bbl@tempa}%
467 {\edef\bbl@tempb{%
468 \noexpand\DeclareOption
469 {\bbl@tempa}%
470 {\noexpand\bbl@load@language{\bbl@tempa}}}%
471 \bbl@tempb}%
472 \@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.

```

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

```

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

```

479 \ifx\bbl@opt@main\@nnil\else
480   \expandafter
481   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
482   \DeclareOption{\bbl@opt@main}{}
483 \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):

```

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

```

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

```

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

```

522 \ifx\ldf@quit\@undefined
523 \else
524   \expandafter\endinput
525 \fi
526 <<Make sure ProvidesFile is defined>>
527 \ProvidesFile{babel.def}[\<<date>>] \<<version>> Babel common definitions]
528 <<Load macros for plain if not LaTeX>>

```

The file babel.def expects some definitions made in the \LaTeX 2 ϵ 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 and alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```

529 \ifx\bbl@ifshorthand\@undefined
530   \let\bbl@opt@shorthands\@nnil
531   \def\bbl@ifshorthand#1#2#3{#2}%
532   \let\bbl@language@opts\@empty
533   \ifx\babeloptionstrings\@undefined
534     \let\bbl@opt@strings\@nnil
535   \else
536     \let\bbl@opt@strings\babeloptionstrings
537   \fi
538   \def\BabelStringsDefault{generic}
539   \def\bbl@tempa{normal}
540   \ifx\babeloptionmath\bbl@tempa
541     \def\bbl@mathnormal{\noexpand\textormath}
542   \fi

```

```

543 \def\AfterBabelLanguage#1#2{}
544 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
545 \let\bbl@afterlang\relax
546 \def\bbl@opt@safe{BR}
547 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
548 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
549 \fi

And continue.
550 \ifx\bbl@switchflag\@undefined % Prevent double input
551 \let\bbl@switchflag\relax
552 \input switch.def\relax
553 \fi
554 \bbl@trace{Compatibility with language.def}
555 \ifx\bbl@languages\@undefined
556 \ifx\directlua\@undefined
557 \openin1 = language.def
558 \ifeof1
559 \closein1
560 \message{I couldn't find the file language.def}
561 \else
562 \closein1
563 \begingroup
564 \def\addlanguage#1#2#3#4#5{%
565 \expandafter\ifx\csname lang@#1\endcsname\relax\else
566 \global\expandafter\let\csname l@#1\endcsname
567 \csname lang@#1\endcsname
568 \fi}%
569 \def\uselanguage#1{}\%
570 \input language.def
571 \endgroup
572 \fi
573 \fi
574 \chardef\l@english\z@
575 \fi
576 <<Load patterns in luatex>>
577 <<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.

```

578 \def\addto#1#2{%
579 \ifx#1\@undefined
580 \def#1{#2}%
581 \else
582 \ifx#1\relax
583 \def#1{#2}%
584 \else
585 {\toks@\expandafter{#1#2}%
586 \xdef#1{the\toks@}}%
587 \fi
588 \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.

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

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

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@...`

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

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

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

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

```
598 \def\bbl@redefine@long#1{%
599   \edef\bbl@tempa{\bbl@stripslash#1}%
600   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
601   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
602 \@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`.

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

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

```
610 \@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.

```
611 \bbl@trace{Hooks}
612 \newcommand\AddBabelHook[3][{}]{%
613   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
614   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
```

```

615 \expandafter\bb1@tempa\bb1@evargs,#3=,\@empty
616 \bb1@ifunset{bb1@ev@#2@#3@#1}%
617 {\bb1@csarg\bb1@add{ev@#3@#1}{\bb1@elt{#2}}}%
618 {\bb1@csarg\let{ev@#2@#3@#1}\relax}%
619 \bb1@csarg\newcommand{ev@#2@#3@#1}[\bb1@tempb]}
620 \newcommand\EnableBabelHook[1]{\bb1@csarg\let{hk@#1}\@firstofone}
621 \newcommand\DisableBabelHook[1]{\bb1@csarg\let{hk@#1}\@gobble}
622 \def\bb1@usehooks#1#2{%
623 \def\bb1@elt##1{%
624 \@nameuse{bb1@hk@##1}{\@nameuse{bb1@ev@##1@#1@#2}}}%
625 \@nameuse{bb1@ev@#1@}%
626 \ifx\language\undefined\else % Test required for Plain (?)
627 \def\bb1@elt##1{%
628 \@nameuse{bb1@hk@##1}{\@nameuse{bb1@ev@##1@#1@\language}#2}}%
629 \@nameuse{bb1@ev@#1@\language}%
630 \fi}

```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```

631 \def\bb1@evargs{% <- don't delete this comma
632 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
633 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
634 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
635 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 `\bb1@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 `\bb1@e@<language>` contains `\bb1@ensure{\include}{\exclude}{\fontenc}`, which in turn loops over the macros names in `\bb1@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.

```

636 \bb1@trace{Defining babelensure}
637 \newcommand\babelensure[2][{}% TODO - revise test files
638 \AddBabelHook{babel-ensure}{afterextras}{%
639 \ifcase\bb1@select@type
640 \@nameuse{bb1@e@\language}%
641 \fi}%
642 \begin{group}
643 \let\bb1@ens@include\@empty
644 \let\bb1@ens@exclude\@empty
645 \def\bb1@ens@fontenc{\relax}%
646 \def\bb1@tempb##1{%
647 \ifx\@empty##1\else\noexpand##1\expandafter\bb1@tempb\fi}%
648 \edef\bb1@tempa{\bb1@tempb#1\@empty}%
649 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ens@##1}{##2}}%
650 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
651 \def\bb1@tempc{\bb1@ensure}%
652 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
653 \expandafter{\bb1@ens@include}}%
654 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
655 \expandafter{\bb1@ens@exclude}}%
656 \toks@\expandafter{\bb1@tempc}%

```

```

657 \bbl@exp{%
658 \endgroup
659 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
660 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
661 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
662 \ifx##1\@undefined % 3.32 - Don't assume the macros exists
663 \edef##1{\noexpand\bbl@nocaption
664 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}}%
665 \fi
666 \ifx##1\@empty\else
667 \in@{##1}{#2}%
668 \ifin@\else
669 \bbl@ifunset{\bbl@ensure@\language}%
670 {\bbl@exp{%
671 \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
672 \\\foreignlanguage{\language}%
673 {\ifx\relax#3\else
674 \\\fontencoding{#3}\selectfont
675 \fi
676 #####1}}}%
677 }%
678 \toks@\expandafter{##1}%
679 \edef##1{%
680 \bbl@csarg\noexpand{ensure@\language}%
681 {\the\toks@}}}%
682 \fi
683 \expandafter\bbl@tempb
684 \fi}%
685 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
686 \def\bbl@tempa##1{% elt for include list
687 \ifx##1\@empty\else
688 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
689 \ifin@\else
690 \bbl@tempb##1\@empty
691 \fi
692 \expandafter\bbl@tempa
693 \fi}%
694 \bbl@tempa#1\@empty}
695 \def\bbl@captionslist{%
696 \prefacename\refname\abstractname\bibname\chaptername\appendixname
697 \contentsname\listfigurename\listtablename\indexname\figurename
698 \tablename\partname\enclname\ccname\headtoname\pagename\seename
699 \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`.

```

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

```

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

```

727 \def\ldf@quit#1{%
728   \expandafter\main@language\expandafter{#1}%
729   \catcode`\@=\atcatcode \let\atcatcode\relax
730   \catcode`\==\eqcatcode \let\eqcatcode\relax
731   \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.

```

732 \def\bbl@afterldf#1{%
733   \bbl@afterlang
734   \let\bbl@afterlang\relax
735   \let\BabelModifiers\relax
736   \let\bbl@screset\relax}%
737 \def\ldf@finish#1{%
738   \loadlocalcfg{#1}%
739   \bbl@afterldf{#1}%

```



```

740 \expandafter\main@language\expandafter{#1}%
741 \catcode`\@=\atcatcode \let\atcatcode\relax
742 \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 .

```

743 \@onlypreamble\LdfInit
744 \@onlypreamble\ldf@quit
745 \@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.

```

746 \def\main@language#1{%
747   \def\bbl@main@language{#1}%
748   \let\language\main@language
749   \bbl@id@assign
750   \chardef\localeid\@nameuse{\bbl@id@\language}%
751   \bbl@patterns{\language}}

```

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`.

```

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

```

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

```

755 \def\select@language#1{%
756   \ifcase\bbl@select@type
757     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
758   \else
759     \select@language{#1}%
760   \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.

```

761 \bbl@trace{Shorhands}
762 \def\bbl@add@special#1{% 1:a macro like "\", \?, etc.
763   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
764   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
765   \ifx\nfss@catcodes\undefined\else % TODO - same for above
766     \begingroup
767       \catcode`#1\active
768       \nfss@catcodes
769       \ifnum\catcode`#1=\active
770         \endgroup
771       \bbl@add\nfss@catcodes{\@makeother#1}%
772     \else

```

```

773     \endgroup
774     \fi
775 \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.

```

776 \def\bbl@remove@special#1{%
777   \begingroup
778   \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
779     \else\noexpand##1\noexpand##2\fi}%
780   \def\do{\x\do}%
781   \def\@makeother{\x\@makeother}%
782 \edef\x{\endgroup
783   \def\noexpand\dospecials{\dospecials}%
784   \expandafter\ifx\csname @sanitize\endcsname\relax\else
785     \def\noexpand\@sanitize{\@sanitize}%
786   \fi}%
787 \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).

```

788 \def\bbl@active@def#1#2#3#4{%
789   \@namedef{#3#1}{%
790     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
791       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
792     \else
793       \bbl@afterfi\csname#2@sh@#1\endcsname
794     \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.

```

795   \long\@namedef{#3@arg#1}##1{%
796     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
797       \bbl@afterelse\csname#4#1\endcsname##1%
798     \else
799       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
800     \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.

```
801 \def\initiate@active@char#1{%
802   \bbl@ifunset{active@char\string#1}%
803   {\bbl@withactive
804     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
805   {}}
```

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`).

```
806 \def\@initiate@active@char#1#2#3{%
807   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
808   \ifx#1\@undefined
809     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
810   \else
811     \bbl@csarg\let{oridef@#2}#1%
812     \bbl@csarg\edef{oridef@#2}{%
813       \let\noexpand#1%
814       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
815   \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` $\langle char \rangle$ 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*").

```
816   \ifx#1#3\relax
817     \expandafter\let\csname normal@char#2\endcsname#3%
818   \else
819     \bbl@info{Making #2 an active character}%
820     \ifnum\mathcode`#2="8000
821       \@namedef{normal@char#2}{%
822         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
823     \else
824       \@namedef{normal@char#2}{#3}%
825     \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).

```
826   \bbl@restoreactive{#2}%
827   \AtBeginDocument{%
828     \catcode`#2\active
829     \if@filesw
830       \immediate\write\@mainaux{\catcode`\string#2\active}%
831     \fi}%
832   \expandafter\bbl@add@special\csname#2\endcsname
833   \catcode`#2\active
834   \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⟩`).

```

835 \let\bbl@tempa\@firstoftwo
836 \if\string^#2%
837   \def\bbl@tempa{\noexpand\textormath}%
838 \else
839   \ifx\bbl@mathnormal\@undefined\else
840     \let\bbl@tempa\bbl@mathnormal
841   \fi
842 \fi
843 \expandafter\edef\csname active@char#2\endcsname{%
844   \bbl@tempa
845     {\noexpand\if@safe@actives
846       \noexpand\expandafter
847       \expandafter\noexpand\csname normal@char#2\endcsname
848     \noexpand\else
849       \noexpand\expandafter
850       \expandafter\noexpand\csname bbl@doactive#2\endcsname
851     \noexpand\fi}%
852   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
853 \bbl@csarg\edef{doactive#2}{%
854   \expandafter\noexpand\csname user@active#2\endcsname}%

```

We now define the default values which the shorthand is set to when activated or deactivated. It is set to the deactivated form (globally), so that the character expands to

`\active@prefix⟨char⟩ \normal@char⟨char⟩`

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

```

855 \bbl@csarg\edef{active@#2}{%
856   \noexpand\active@prefix\noexpand#1%
857   \expandafter\noexpand\csname active@char#2\endcsname}%
858 \bbl@csarg\edef{normal@#2}{%
859   \noexpand\active@prefix\noexpand#1%
860   \expandafter\noexpand\csname normal@char#2\endcsname}%
861 \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.

```

862 \bbl@active@def#2\user@group{user@active}{language@active}%
863 \bbl@active@def#2\language@group{language@active}{system@active}%
864 \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.

```

865 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
866   {\expandafter\noexpand\csname normal@char#2\endcsname}%
867 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
868   {\expandafter\noexpand\csname user@active#2\endcsname}%

```

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (‘) active we need to change `\pr@m@s` as well. Also, make sure that a single ‘ in math mode

‘does the right thing’. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

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

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

```
874 <<*More package options>> ≡
875 \DeclareOption{math=active}{}
876 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
877 <</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.

```
878 \@ifpackagewith{babel}{KeepShorthandsActive}%
879 {\let\bbl@restoreactive\@gobble}%
880 {\def\bbl@restoreactive#1{%
881   \bbl@exp{%
882     \\\AfterBabelLanguage\\CurrentOption
883     {\catcode`#1=\the\catcode`#1\relax}%
884     \\\AtEndOfPackage
885     {\catcode`#1=\the\catcode`#1\relax}}}%
886 \AtEndOfPackage{\let\bbl@restoreactive\@gobble}}
```

`\bbl@sh@select` This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of `\hyphenation`. This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either `\bbl@firstcs` or `\bbl@scndcs`. Hence two more arguments need to follow it.

```
887 \def\bbl@sh@select#1#2{%
888   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
889     \bbl@afterelse\bbl@scndcs
890   \else
891     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
892   \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`.

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

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

```
896   \ifx\protect\@unexpandable@protect
897     \noexpand#1%
898   \else
899     \protect#1%
900   \fi
```

```

901 \expandafter\@gobble
902 \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⟩.

903 \newif\if@safe@actives
904 \@safe@activesfalse

\bb1@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.

905 \def\bb1@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

\bb1@activate Both macros take one argument, like \initiate@active@char. The macro is used to
\bb1@deactivate change the definition of an active character to expand to \active@char⟨char⟩ in the case
of \bb1@activate, or \normal@char⟨char⟩ in the case of \bb1@deactivate.

906 \def\bb1@activate#1{%
907 \bb1@withactive{\expandafter\let\expandafter}#1%
908 \csname bbl@active@\string#1\endcsname}
909 \def\bb1@deactivate#1{%
910 \bb1@withactive{\expandafter\let\expandafter}#1%
911 \csname bbl@normal@\string#1\endcsname}

\bb1@firstcs These macros have two arguments. They use one of their arguments to build a control
\bb1@scndcs sequence from.

912 \def\bb1@firstcs#1#2{\csname#1\endcsname}
913 \def\bb1@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.

914 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
915 \def\@decl@short#1#2#3\@nil#4{%
916 \def\bb1@tempa{#3}%
917 \ifx\bb1@tempa\@empty
918 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@scndcs
919 \bb1@ifunset{#1@sh@\string#2@}{}%
920 {\def\bb1@tempa{#4}%
921 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bb1@tempa
922 \else
923 \bb1@info
924 {Redefining #1 shorthand \string#2\\%
925 in language \CurrentOption}%
926 \fi}%
927 \@namedef{#1@sh@\string#2@}{#4}%
928 \else
929 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@firstcs
930 \bb1@ifunset{#1@sh@\string#2@\string#3@}{}%
931 {\def\bb1@tempa{#4}%
932 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bb1@tempa
933 \else

```

```

934      \bbl@info
935      {Redefining #1 shorthand \string#2\string#3\\%
936       in language \CurrentOption}%
937      \fi}%
938      \@namedef{#1@sh@\string#2@\string#3@}{#4}%
939      \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.

940 \def\textormath{%
941   \ifmmode
942     \expandafter\@secondoftwo
943   \else
944     \expandafter\@firstoftwo
945   \fi}

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

946 \def\user@group{user}
947 \def\language@group{english}
948 \def\system@group{system}

\useshorthands This is the user level command to tell  $\TeX$  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.

949 \def\useshorthands{%
950   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
951 \def\bbl@usesh@s#1{%
952   \bbl@usesh@x
953   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
954   {#1}}
955 \def\bbl@usesh@x#1#2{%
956   \bbl@ifshorthand{#2}%
957   {\def\user@group{user}%
958    \initiate@active@char{#2}%
959    #1%
960    \bbl@activate{#2}}%
961   {\bbl@error
962    {Cannot declare a shorthand turned off (\string#2)}
963    {Sorry, but you cannot use shorthands which have been\\%
964     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.

965 \def\user@language@group{user@\language@group}
966 \def\bbl@set@user@generic#1#2{%
967   \bbl@ifunset{user@generic@active#1}%
968   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
969    \bbl@active@def#1\user@group{user@generic@active}{language@active}%
970    \expandafter\edef\csname#2@sh@#1@@\endcsname{

```

```

971 \expandafter\noexpand\csname normal@char#1\endcsname}%
972 \expandafter\edef\csname#2@sh@#1\string\protect@\endcsname{%
973 \expandafter\noexpand\csname user@active#1\endcsname}}%
974 \@empty}
975 \newcommand\defineshorthand[3][user]{%
976 \edef\bbl@tempa{\zap@space#1 \@empty}%
977 \bbl@for\bbl@tempb\bbl@tempa{%
978 \if*\expandafter\@car\bbl@tempb\@nil
979 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
980 \@expandtwoargs
981 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
982 \fi
983 \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.

```

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

```

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

```

985 \def\aliasshorthand#1#2{%
986 \bbl@ifshorthand{#2}%
987 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
988 \ifx\document\@notprerr
989 \@notshorthand{#2}%
990 \else
991 \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`.

```

992 \expandafter\let\csname active@char\string#2\expandafter\endcsname
993 \csname active@char\string#1\endcsname
994 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
995 \csname normal@char\string#1\endcsname
996 \bbl@activate{#2}%
997 \fi
998 \fi}%
999 {\bbl@error
1000 {Cannot declare a shorthand turned off (\string#2)}
1001 {Sorry, but you cannot use shorthands which have been\\
1002 turned off in the package options}}}

```

`\@notshorthand`

```

1003 \def\@notshorthand#1{%
1004 \bbl@error{%
1005 The character '\string #1' should be made a shorthand character;\\%
1006 add the command \string\usesshorthands\string{#1\string} to
1007 the preamble.\\%
1008 I will ignore your instruction}%
1009 {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.

```

1010 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1011 \DeclareRobustCommand*\shorthandoff{%
1012 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1013 \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.

```

1014 \def\bbl@switch@sh#1#2{%
1015   \ifx#2\@nnil\else
1016     \bbl@ifunset{\bbl@active@\string#2}%
1017     {\bbl@error
1018       {I cannot switch '\string#2' on or off--not a shorthand}%
1019       {This character is not a shorthand. Maybe you made\\%
1020         a typing mistake? I will ignore your instruction}}%
1021     {\ifcase#1%
1022       \catcode'#212\relax
1023       \or
1024       \catcode'#2\active
1025       \or
1026       \csname bbl@oricat@\string#2\endcsname
1027       \csname bbl@oridef@\string#2\endcsname
1028       \fi}%
1029     \bbl@afterfi\bbl@switch@sh#1%
1030   \fi}

```

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

```

1031 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1032 \def\bbl@putsh#1{%
1033   \bbl@ifunset{\bbl@active@\string#1}%
1034   {\bbl@putsh@i#1\@empty\@nnil}%
1035   {\csname bbl@active@\string#1\endcsname}}
1036 \def\bbl@putsh@i#1#2\@nnil{%
1037   \csname\language\sh@\string#1@%
1038     \ifx\@empty#2\else\string#2@\fi\endcsname}
1039 \ifx\bbl@opt@shorthands\@nnil\else
1040   \let\bbl@s@initiate@active@char\initiate@active@char
1041   \def\initiate@active@char#1{%
1042     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1043   \let\bbl@s@switch@sh\bbl@switch@sh
1044   \def\bbl@switch@sh#1#2{%
1045     \ifx#2\@nnil\else
1046       \bbl@afterfi
1047       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1048     \fi}
1049   \let\bbl@s@activate\bbl@activate
1050   \def\bbl@activate#1{%
1051     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1052   \let\bbl@s@deactivate\bbl@deactivate
1053   \def\bbl@deactivate#1{%
1054     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1055 \fi

```

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

```

1056 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

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

```

1057 \def\bbl@prim@s{%
1058   \prime\futurelet\@let@token\bbl@pr@m@s}
1059 \def\bbl@if@primes#1#2{%
1060   \ifx#1\@let@token
1061     \expandafter\@firstoftwo
1062   \else\ifx#2\@let@token
1063     \bbl@afterelse\expandafter\@firstoftwo
1064   \else
1065     \bbl@afterfi\expandafter\@secondoftwo
1066   \fi\fi}
1067 \begingroup
1068   \catcode`\^=7 \catcode`\*=\active \lccode`\*='^
1069   \catcode`\'=12 \catcode`\"=\active \lccode`\"='\'
1070   \lowercase{%
1071     \gdef\bbl@pr@m@s{%
1072       \bbl@if@primes""%
1073       \pr@@@s
1074       {\bbl@if@primes*^ \pr@@@t\egroup}}
1075 \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).

```

1076 \initiate@active@char{~}
1077 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1078 \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.

```

1079 \expandafter\def\csname OT1dqpos\endcsname{127}
1080 \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

```

1081 \ifx\f@encoding\@undefined
1082   \def\f@encoding{OT1}
1083 \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.

```

1084 \bbl@trace{Language attributes}
1085 \newcommand\languageattribute[2]{%

```

```

1086 \def\bbl@tempc{#1}%
1087 \bbl@fixname\bbl@tempc
1088 \bbl@iflanguage\bbl@tempc{%
1089   \bbl@vforeach{#2}{%

```

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

```

1090   \ifx\bbl@known@attribs\undefined
1091     \in@false
1092   \else

```

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

```

1093     \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1094   \fi

```

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

```

1095   \ifin@
1096     \bbl@warning{%
1097       You have more than once selected the attribute '##1'\%
1098       for language #1. Reported}%
1099   \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.

```

1100     \bbl@exp{%
1101       \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1102     \edef\bbl@tempa{\bbl@tempc-##1}%
1103     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1104     {\csname\bbl@tempc @attr##1\endcsname}%
1105     {\@attrerrr{\bbl@tempc}{##1}}%
1106   \fi}}

```

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

```

1107 \@onlypreamble\languageattribute

```

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

```

1108 \newcommand*{\@attrerrr}[2]{%
1109   \bbl@error
1110   {The attribute #2 is unknown for language #1.}%
1111   {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}`.

```

1112 \def\bbl@declare@ttribute#1#2#3{%
1113   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1114   \ifin@
1115     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1116   \fi
1117   \bbl@add@list\bbl@attributes{#1-#2}%
1118   \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.

```
1119 \def\bbl@ifattributeset#1#2#3#4{%
```

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

```
1120 \ifx\bbl@known@attribs\@undefined
1121 \in@false
1122 \else
```

The we need to check the list of known attributes.

```
1123 \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1124 \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`'.

```
1125 \ifin@
1126 \bbl@afterelse#3%
1127 \else
1128 \bbl@afterfi#4%
1129 \fi
1130 }
```

`\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.

```
1131 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1132 \let\bbl@tempa\@secondoftwo
```

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

```
1133 \bbl@loopx\bbl@tempb{#2}{%
1134 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1135 \ifin@
```

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

```
1136 \let\bbl@tempa\@firstoftwo
1137 \else
1138 \fi}%
```

Finally we execute `\bbl@tempa`.

```
1139 \bbl@tempa
1140 }
```

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

```
1141 \def\bbl@clear@ttribs{%
1142 \ifx\bbl@attributes\@undefined\else
1143 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1144 \expandafter\bbl@clear@ttrib\bbl@tempa.
1145 }%
1146 \let\bbl@attributes\@undefined
1147 \fi}
1148 \def\bbl@clear@ttrib#1-#2.{%
1149 \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1150 \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` 1151 `\bbl@trace{Macros for saving definitions}`
1152 `\def\babel@beginsave{\babel@savecnt\z@}`

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

1153 `\newcount\babel@savecnt`
1154 `\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.

1155 `\def\babel@save#1{%`
1156 `\expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax`
1157 `\toks@\expandafter{\originalTeX\let#1=}%`
1158 `\bbl@exp{%`
1159 `\def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%`
1160 `\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.

1161 `\def\babel@savevariable#1{%`
1162 `\toks@\expandafter{\originalTeX #1=}%`
1163 `\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.

1164 `\def\bbl@frenchspacing{%`
1165 `\ifnum\the\sffcode`\.=@m`
1166 `\let\bbl@nonfrenchspacing\relax`
1167 `\else`
1168 `\frenchspacing`
1169 `\let\bbl@nonfrenchspacing\nonfrenchspacing`
1170 `\fi}`
1171 `\let\bbl@nonfrenchspacing\nonfrenchspacing`

9.7 Short tags

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

1172 `\bbl@trace{Short tags}`
1173 `\def\babeltags#1{%`

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

```

1174 \edef\bbl@tempa{\zap@space#1 \@empty}%
1175 \def\bbl@tempb##1=##2\@@{%
1176   \edef\bbl@tempc{%
1177     \noexpand\newcommand
1178     \expandafter\noexpand\csname ##1\endcsname{%
1179       \noexpand\protect
1180       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1181     \noexpand\newcommand
1182     \expandafter\noexpand\csname text##1\endcsname{%
1183       \noexpand\foreignlanguage{##2}}
1184   \bbl@tempc}%
1185 \bbl@for\bbl@tempa\bbl@tempa{%
1186   \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.

```

1187 \bbl@trace{Hyphens}
1188 \@onlypreamble\babelhyphenation
1189 \AtEndOfPackage{%
1190   \newcommand\babelhyphenation[2][\@empty]{%
1191     \ifx\bbl@hyphenation@\relax
1192       \let\bbl@hyphenation@\@empty
1193     \fi
1194     \ifx\bbl@hyphlist\@empty\else
1195       \bbl@warning{%
1196         You must not intermingle \string\selectlanguage\space and\\%
1197         \string\babelhyphenation\space or some exceptions will not\\%
1198         be taken into account. Reported}%
1199     \fi
1200     \ifx\@empty#1%
1201       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1202     \else
1203       \bbl@vforeach{#1}{%
1204         \def\bbl@tempa{##1}%
1205         \bbl@fixname\bbl@tempa
1206         \bbl@iflanguage\bbl@tempa{%
1207           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1208             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1209             \@empty
1210             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1211             #2}}}%
1212       \fi}}

```

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

```

1213 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
1214 \def\bbl@t@one{T1}
1215 \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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1216 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1217 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
1218 \def\bb1@hyphen{%
1219   \@ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i \@empty}}
1220 \def\bb1@hyphen@i#1#2{%
1221   \bb1@ifunset{\bb1@hy@#1#2\@empty}%
1222   {\csname bb1@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1223   {\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.

```

1224 \def\bb1@usehyphen#1{%
1225   \leavevmode
1226   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1227   \nobreak\hskip\z@skip}
1228 \def\bb1@@usehyphen#1{%
1229   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1230 \def\bb1@hyphenchar{%
1231   \ifnum\hyphenchar\font=\m@ne
1232     \babelnullhyphen
1233   \else
1234     \char\hyphenchar\font
1235   \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.

```

1236 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1237 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1238 \def\bb1@hy@hard{\bb1@usehyphen\bb1@hyphenchar}
1239 \def\bb1@hy@@hard{\bb1@usehyphen\bb1@hyphenchar}
1240 \def\bb1@hy@nobreak{\bb1@usehyphen{\mbox{\bb1@hyphenchar}}}
1241 \def\bb1@hy@nobreak{\mbox{\bb1@hyphenchar}}
1242 \def\bb1@hy@repeat{%
1243   \bb1@usehyphen{%
1244     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
1245 \def\bb1@hy@@repeat{%
1246   \bb1@usehyphen{%
1247     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
1248 \def\bb1@hy@empty{\hskip\z@skip}
1249 \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.

```

1250 \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.

```

1251 \bbl@trace{Multiencoding strings}
1252 \def\bbl@tglobal#1{\global\let#1#1}
1253 \def\bbl@recatcode#1{%
1254   \@tempcnta="7F
1255   \def\bbl@tempa{%
1256     \ifnum\@tempcnta>"FF\else
1257       \catcode\@tempcnta=#1\relax
1258       \advance\@tempcnta\@ne
1259       \expandafter\bbl@tempa
1260     \fi}%
1261   \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).

```

1262 \@ifpackagewith{babel}{nocase}%
1263   {\let\bbl@patchuclc\relax}%
1264   {\def\bbl@patchuclc{%
1265     \global\let\bbl@patchuclc\relax
1266     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1267     \gdef\bbl@uclc##1{%
1268       \let\bbl@encoded\bbl@encoded@uclc
1269       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1270       {##1}%
1271       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1272         \csname\language @bbl@uclc\endcsname}%
1273       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1274     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1275     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1276 <<(*More package options)>> ≡
1277 \DeclareOption{nocase}{}
1278 <</More package options>>

```

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

```

1279 <<(*More package options)>> ≡
1280 \let\bbl@opt@strings\@nnil % accept strings=value
1281 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1282 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1283 \def\BabelStringsDefault{generic}
1284 <</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.

```
1285 \@onlypreamble\StartBabelCommands
```



```

1286 \def\StartBabelCommands{%
1287   \begingroup
1288   \bbl@recatcode{11}%
1289   <<Macros local to BabelCommands>>
1290   \def\bbl@provstring##1##2{%
1291     \providecommand##1{##2}%
1292     \bbl@toglobal##1}%
1293   \global\let\bbl@scafter\@empty
1294   \let\StartBabelCommands\bbl@startcmds
1295   \ifx\BabelLanguages\relax
1296     \let\BabelLanguages\CurrentOption
1297   \fi
1298   \begingroup
1299   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1300   \StartBabelCommands}
1301 \def\bbl@startcmds{%
1302   \ifx\bbl@screset\@nnil\else
1303     \bbl@usehooks{stopcommands}{}%
1304   \fi
1305   \endgroup
1306   \begingroup
1307   \@ifstar
1308     {\ifx\bbl@opt@strings\@nnil
1309       \let\bbl@opt@strings\BabelStringsDefault
1310     \fi
1311     \bbl@startcmds@i}%
1312   \bbl@startcmds@i}
1313 \def\bbl@startcmds@i#1#2{%
1314   \edef\bbl@L{\zap@space#1 \@empty}%
1315   \edef\bbl@G{\zap@space#2 \@empty}%
1316   \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.

```

1317 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1318   \let\SetString\@gobbletwo
1319   \let\bbl@stringdef\@gobbletwo
1320   \let\AfterBabelCommands\@gobble
1321   \ifx\@empty#1%
1322     \def\bbl@sc@label{generic}%
1323     \def\bbl@encstring##1##2{%
1324       \ProvideTextCommandDefault##1{##2}%
1325       \bbl@toglobal##1%
1326       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1327     \let\bbl@sctest\in@true
1328   \else
1329     \let\bbl@sc@charset\space % <- zapped below
1330     \let\bbl@sc@fontenc\space % <- " "
1331     \def\bbl@tempa##1=##2\@nil{%
1332       \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%

```

```

1333 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1334 \def\bbl@tempa##1 ##2{% space -> comma
1335   ##1%
1336   \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1337 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1338 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1339 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1340 \def\bbl@encstring##1##2{%
1341   \bbl@foreach\bbl@sc@fontenc{%
1342     \bbl@ifunset{T####1}%
1343     {}%
1344     {\ProvideTextCommand##1{####1}{##2}%
1345     \bbl@tglobal##1%
1346     \expandafter
1347     \bbl@tglobal\csname####1\string##1\endcsname}}}%
1348 \def\bbl@sctest{%
1349   \bbl@xin@{\,\bbl@opt@strings,}{\,\bbl@sc@label,\bbl@sc@fontenc,}%
1350 \fi
1351 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1352 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1353   \let\AfterBabelCommands\bbl@aftercmds
1354   \let\SetString\bbl@setstring
1355   \let\bbl@stringdef\bbl@encstring
1356 \else % ie, strings=value
1357 \bbl@sctest
1358 \ifin@
1359   \let\AfterBabelCommands\bbl@aftercmds
1360   \let\SetString\bbl@setstring
1361   \let\bbl@stringdef\bbl@provstring
1362 \fi\fi\fi
1363 \bbl@scswitch
1364 \ifx\bbl@G\@empty
1365   \def\SetString##1##2{%
1366     \bbl@error{Missing group for string \string##1}%
1367     {You must assign strings to some category, typically\\%
1368     captions or extras, but you set none}}%
1369 \fi
1370 \ifx\@empty#1%
1371   \bbl@usehooks{defaultcommands}{}%
1372 \else
1373   \@expandtwoargs
1374   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1375 \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).

```

1376 \def\bbl@forlang#1##2{%
1377   \bbl@for#1\bbl@L{%
1378     \bbl@xin@{, #1,}{\,\BabelLanguages,}%
1379     \ifin@#2\relax\fi}}
1380 \def\bbl@scswitch{%
1381   \bbl@forlang\bbl@tempa{%

```

```

1382 \ifx\bb1@G\@empty\else
1383 \ifx\SetString\@gobbletwo\else
1384 \edef\bb1@GL{\bb1@G\bb1@tempa}%
1385 \bb1@xin@{\bb1@GL,}{\bb1@screset,}%
1386 \ifin@else
1387 \global\expandafter\let\csname\bb1@GL\endcsname\@undefined
1388 \xdef\bb1@screset{\bb1@screset,\bb1@GL}%
1389 \fi
1390 \fi
1391 \fi}}
1392 \AtEndOfPackage{%
1393 \def\bb1@forlang#1#2{\bb1@for#1\bb1@L{\bb1@ifunset{date#1}{\#2}}}%
1394 \let\bb1@scswitch\relax}
1395 \@onlypreamble\EndBabelCommands
1396 \def\EndBabelCommands{%
1397 \bb1@usehooks{stopcommands}{}%
1398 \endgroup
1399 \endgroup
1400 \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.

```

1401 \def\bb1@setstring#1#2{%
1402 \bb1@forlang\bb1@tempa{%
1403 \edef\bb1@LC{\bb1@tempa\bb1@stripslash#1}%
1404 \bb1@ifunset{\bb1@LC}% eg, \germanchaptername
1405 {\global\expandafter % TODO - con \bb1@exp ?
1406 \bb1@add\csname\bb1@G\bb1@tempa\expandafter\endcsname\expandafter
1407 {\expandafter\bb1@scset\expandafter#1\csname\bb1@LC\endcsname}}}%
1408 {}}%
1409 \def\BabelString{\#2}%
1410 \bb1@usehooks{stringprocess}{}%
1411 \expandafter\bb1@stringdef
1412 \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.

```

1413 \ifx\bb1@opt@strings\relax
1414 \def\bb1@scset#1#2{\def#1{\bb1@encoded#2}}
1415 \bb1@patchuclc
1416 \let\bb1@encoded\relax
1417 \def\bb1@encoded@uclc#1{%
1418 \@inmathwarn#1%
1419 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1420 \expandafter\ifx\csname ?\string#1\endcsname\relax
1421 \TextSymbolUnavailable#1%
1422 \else
1423 \csname ?\string#1\endcsname
1424 \fi
1425 \else
1426 \csname\cf@encoding\string#1\endcsname

```

```

1427 \fi}
1428 \else
1429 \def\bbl@scset#1#2{\def#1{#2}}
1430 \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.

```

1431 <<(*Macros local to BabelCommands)>> ≡
1432 \def\SetStringLoop##1##2{%
1433   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1434   \count@\z@
1435   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1436     \advance\count@\@ne
1437     \toks@\expandafter{\bbl@tempa}%
1438     \bbl@exp{%
1439       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1440       \count@=\the\count@\relax}}}%
1441 <</Macros local to BabelCommands)>>

```

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

```

1442 \def\bbl@aftercmds#1{%
1443   \toks@\expandafter{\bbl@scafter#1}%
1444   \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.

```

1445 <<(*Macros local to BabelCommands)>> ≡
1446 \newcommand\SetCase[3][1]{%
1447   \bbl@patchuclc
1448   \bbl@forlang\bbl@tempa{%
1449     \expandafter\bbl@encstring
1450     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1451     \expandafter\bbl@encstring
1452     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1453     \expandafter\bbl@encstring
1454     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1455 <</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.

```

1456 <<(*Macros local to BabelCommands)>> ≡
1457 \newcommand\SetHyphenMap[1]{%
1458   \bbl@forlang\bbl@tempa{%
1459     \expandafter\bbl@stringdef
1460     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1461 <</Macros local to BabelCommands)>>

```

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

```

1462 \newcommand\BabelLower[2]{% one to one.
1463   \ifnum\lccode#1=#2\else
1464     \babel@savevariable{\lccode#1}%
1465     \lccode#1=#2\relax
1466   \fi}

```

```

1467 \newcommand\BabelLowerMM[4]{% many-to-many
1468   \@tempcnta=#1\relax
1469   \@tempcntb=#4\relax
1470   \def\bbl@tempa{%
1471     \ifnum\@tempcnta>#2\else
1472       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1473       \advance\@tempcnta#3\relax
1474       \advance\@tempcntb#3\relax
1475       \expandafter\bbl@tempa
1476     \fi}%
1477   \bbl@tempa}
1478 \newcommand\BabelLowerMO[4]{% many-to-one
1479   \@tempcnta=#1\relax
1480   \def\bbl@tempa{%
1481     \ifnum\@tempcnta>#2\else
1482       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1483       \advance\@tempcnta#3
1484       \expandafter\bbl@tempa
1485     \fi}%
1486   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1487 <<{*More package options}> ≡
1488 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1489 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1490 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1491 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1492 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1493 <</More package options>>

```

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

```

1494 \AtEndOfPackage{%
1495   \ifx\bbl@opt@hyphenmap\undefined
1496     \bbl@xin@{,}{\bbl@language@opts}%
1497     \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
1498   \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.

```

1499 \bbl@trace{Macros related to glyphs}
1500 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1501   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1502   \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.

```

1503 \def\save@sf@q#1{\leavevmode
1504   \begingroup
1505     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1506   \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.

```
1507 \ProvideTextCommand{\quotedblbase}{OT1}{%  
1508   \save@sf@q{\set@low@box{\textquotedblright\}%  
1509     \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.

```
1510 \ProvideTextCommandDefault{\quotedblbase}{%  
1511   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1512 \ProvideTextCommand{\quotesinglbase}{OT1}{%  
1513   \save@sf@q{\set@low@box{\textquoteright\}%  
1514     \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.

```
1515 \ProvideTextCommandDefault{\quotesinglbase}{%  
1516   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
\guillemotright 1517 \ProvideTextCommand{\guillemotleft}{OT1}{%  
1518   \ifmmode  
1519     \ll  
1520   \else  
1521     \save@sf@q{\nobreak  
1522       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%  
1523   \fi}  
1524 \ProvideTextCommand{\guillemotright}{OT1}{%  
1525   \ifmmode  
1526     \gg  
1527   \else  
1528     \save@sf@q{\nobreak  
1529       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%  
1530   \fi}
```

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

```
1531 \ProvideTextCommandDefault{\guillemotleft}{%  
1532   \UseTextSymbol{OT1}{\guillemotleft}}  
1533 \ProvideTextCommandDefault{\guillemotright}{%  
1534   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1535 \ProvideTextCommand{\guilsinglleft}{OT1}{%  
1536   \ifmmode  
1537     <%  
1538   \else  
1539     \save@sf@q{\nobreak  
1540       \raise.2ex\hbox{$\scriptscriptstyle<$}\bb1@allowhyphens}%  
1541   \fi}  
1542 \ProvideTextCommand{\guilsinglright}{OT1}{%  
1543   \ifmmode
```

```

1544 >%
1545 \else
1546 \save@sf@q{\nobreak
1547 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1548 \fi}

```

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

```

1549 \ProvideTextCommandDefault{\guilsinglleft}{%
1550 \UseTextSymbol{OT1}{\guilsinglleft}}
1551 \ProvideTextCommandDefault{\guilsinglright}{%
1552 \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.

```

1553 \DeclareTextCommand{\ij}{OT1}{%
1554 i\kern-0.02em\bbl@allowhyphens j}
1555 \DeclareTextCommand{\IJ}{OT1}{%
1556 I\kern-0.02em\bbl@allowhyphens J}
1557 \DeclareTextCommand{\ij}{T1}{\char188}
1558 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

1559 \ProvideTextCommandDefault{\ij}{%
1560 \UseTextSymbol{OT1}{\ij}}
1561 \ProvideTextCommandDefault{\IJ}{%
1562 \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

1563 \def\crrtic@{\hrule height0.1ex width0.3em}
1564 \def\crttic@{\hrule height0.1ex width0.33em}
1565 \def\ddj@{%
1566 \setbox0\hbox{d}\dimen@=\ht0
1567 \advance\dimen@1ex
1568 \dimen@.45\dimen@
1569 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1570 \advance\dimen@ii.5ex
1571 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1572 \def\DDJ@{%
1573 \setbox0\hbox{D}\dimen@=.55\ht0
1574 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1575 \advance\dimen@ii.15ex % correction for the dash position
1576 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1577 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1578 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1579 %
1580 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1581 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

1582 \ProvideTextCommandDefault{\dj}{%
1583   \UseTextSymbol{OT1}{\dj}}
1584 \ProvideTextCommandDefault{\DJ}{%
1585   \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.

```

1586 \DeclareTextCommand{\SS}{OT1}{SS}
1587 \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 1588 \ProvideTextCommandDefault{\glq}{%
1589   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

1590 \ProvideTextCommand{\grq}{T1}{%
1591   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
1592 \ProvideTextCommand{\grq}{TU}{%
1593   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
1594 \ProvideTextCommand{\grq}{OT1}{%
1595   \save@sf@q{\kern-.0125em
1596     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1597     \kern.07em\relax}}
1598 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

\glqq The ‘german’ double quotes.

```

\grqq 1599 \ProvideTextCommandDefault{\glqq}{%
1600   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1601 \ProvideTextCommand{\grqq}{T1}{%
1602   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}}
1603 \ProvideTextCommand{\grqq}{TU}{%
1604   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}}
1605 \ProvideTextCommand{\grqq}{OT1}{%
1606   \save@sf@q{\kern-.07em
1607     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1608     \kern.07em\relax}}
1609 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}

```

\flq The ‘french’ single guillemets.

```

\frq 1610 \ProvideTextCommandDefault{\flq}{%
1611   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1612 \ProvideTextCommandDefault{\frq}{%
1613   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```


`\flqq` The ‘french’ double guillemets.

```
\frqq 1614 \ProvideTextCommandDefault{\flqq}{%
1615   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1616 \ProvideTextCommandDefault{\frqq}{%
1617   \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 positioning, the default will be `\umlauthigh` (the normal positioning).

```
\umlautlow 1618 \def\uumlauthigh{%
1619   \def\bbl@umlauta##1{\leavevmode\bgroup%
1620     \expandafter\accent\csname\fontencoding dqpos\endcsname
1621     ##1\bbl@allowhyphens\egroup}%
1622   \let\bbl@umlaute\bbl@umlauta}
1623 \def\uumlautlow{%
1624   \def\bbl@umlauta{\protect\lower@umlaut}}
1625 \def\umlautelower{%
1626   \def\bbl@umlaute{\protect\lower@umlaut}}
1627 \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.

```
1628 \expandafter\ifx\csname U@D\endcsname\relax
1629   \csname newdimen\endcsname\U@D
1630 \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.

```
1631 \def\lower@umlaut#1{%
1632   \leavevmode\bgroup
1633   \U@D 1ex%
1634   {\setbox\z@\hbox{%
1635     \expandafter\char\csname\fontencoding dqpos\endcsname}%
1636     \dimen@ -.45ex\advance\dimen@\ht\z@
1637     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1638   \expandafter\accent\csname\fontencoding dqpos\endcsname
1639   \fontdimen5\font\U@D #1%
1640   \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).

```

1641 \AtBeginDocument{%
1642   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
1643   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
1644   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
1645   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
1646   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
1647   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
1648   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
1649   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
1650   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
1651   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
1652   \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}%
1653 }

```

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

```

1654 \ifx\l@english\@undefined
1655   \chardef\l@english\z@
1656 \fi
1657 \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.

```

1658 \bbl@trace{Bidi layout}
1659 \providecommand\IfBabelLayout[3]{#3}%
1660 \newcommand\BabelPatchSection[1]{%
1661   \@ifundefined{#1}{%
1662     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1663     \@namedef{#1}{%
1664       \ifstar\bbl@presec{s{#1}}%
1665       {\@dblarg\bbl@presec{x{#1}}}}%
1666   \def\bbl@presec{x#1[#2]#3}%
1667   \bbl@exp{%
1668     \\select@language{x\bbl@main@language}%
1669     \\@nameuse{bbl@sspre@#1}%
1670     \\@nameuse{bbl@ss@#1}%
1671     [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1672     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
1673     \\select@language{x\languagename}}%
1674   \def\bbl@presec{s#1#2{%
1675     \bbl@exp{%
1676       \\select@language{x\bbl@main@language}%
1677       \\@nameuse{bbl@sspre@#1}%
1678       \\@nameuse{bbl@ss@#1}%
1679       {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1680       \\select@language{x\languagename}}%
1681   \IfBabelLayout{sectioning}%
1682   {\BabelPatchSection{part}%
1683    \BabelPatchSection{chapter}%
1684    \BabelPatchSection{section}%
1685    \BabelPatchSection{subsection}%

```

```

1686 \BabelPatchSection{subsubsection}%
1687 \BabelPatchSection{paragraph}%
1688 \BabelPatchSection{subparagraph}%
1689 \def\babel@toc#1{%
1690 \select@language@x{\bbl@main@language}}{}
1691 \IfBabelLayout{captions}%
1692 {\BabelPatchSection{caption}}{}

```

9.13 Load engine specific macros

```

1693 \bbl@trace{Input engine specific macros}
1694 \ifcase\bbl@engine
1695 \input txtbabel.def
1696 \or
1697 \input luababel.def
1698 \or
1699 \input xebabel.def
1700 \fi

```

9.14 Creating languages

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

```

1701 \bbl@trace{Creating languages and reading ini files}
1702 \newcommand\babelprovide[2][{}]{%
1703 \let\bbl@savelangname\language@name
1704 \edef\bbl@savelocaleid{\the\localeid}%
1705 % Set name and locale id
1706 \def\language@name{#2}%
1707 \bbl@id@assign
1708 \chardef\localeid\@nameuse{\bbl@id@\language@name}%
1709 \let\bbl@KVP@captions\@nil
1710 \let\bbl@KVP@import\@nil
1711 \let\bbl@KVP@main\@nil
1712 \let\bbl@KVP@script\@nil
1713 \let\bbl@KVP@language\@nil
1714 \let\bbl@KVP@dir\@nil
1715 \let\bbl@KVP@hyphenrules\@nil
1716 \let\bbl@KVP@mapfont\@nil
1717 \let\bbl@KVP@maparabic\@nil
1718 \let\bbl@KVP@mapdigits\@nil
1719 \let\bbl@KVP@intraspace\@nil
1720 \let\bbl@KVP@intrapenalty\@nil
1721 \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}% TODO - error handling
1722 \ifx\bbl@KVP@import\@nil\else
1723 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1724 {\begin@group
1725 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1726 \InputIfFileExists{babel-#2.tex}{}{}}%
1727 \end@group}%
1728 {}%
1729 \fi
1730 \ifx\bbl@KVP@captions\@nil
1731 \let\bbl@KVP@captions\bbl@KVP@import
1732 \fi
1733 % Load ini
1734 \bbl@ifunset{date#2}%

```

```

1735 {\bbl@provide@new{#2}}%
1736 {\bbl@ifblank{#1}}%
1737 {\bbl@error
1738   {If you want to modify `#2' you must tell how in\\%
1739     the optional argument. See the manual for the\\%
1740     available options.}%
1741   {Use this macro as documented}}%
1742 {\bbl@provide@renew{#2}}}%
1743 % Post tasks
1744 \bbl@exp{\\babelensure[exclude=\\today]{#2}}%
1745 \bbl@ifunset{bbl@ensure@\\languagename}%
1746   {\bbl@exp{%
1747     \\DeclareRobustCommand\<bbl@ensure@\\languagename>[1]{%
1748       \\foreignlanguage{\\languagename}%
1749       {###1}}}%
1750   }%
1751 % At this point all parameters are defined if 'import'. Now we
1752 % execute some code depending on them. But what about if nothing was
1753 % imported? We just load the very basic parameters: ids and a few
1754 % more.
1755 \bbl@ifunset{bbl@lname@#2}%
1756   {\def\BabelBeforeIni##1##2{%
1757     \begingroup
1758       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
1759       \let\bbl@ini@captions@aux\@gobbletwo
1760       \def\bbl@inidate ###1.###2.###3.###4\relax ###5###6{%
1761         \bbl@read@ini{##1}%
1762         \bbl@exportkey{chrng}{characters.ranges}{}%
1763         \bbl@exportkey{dgnat}{numbers.digits.native}{}%
1764         \endgroup%
1765         {\setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
1766       }%
1767   % -
1768 % Override script and language names with script= and language=
1769 \ifx\bbl@KVP@script\@nil\else
1770   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1771 \fi
1772 \ifx\bbl@KVP@language\@nil\else
1773   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1774 \fi
1775 % For bidi texts, to switch the language based on direction
1776 \ifx\bbl@KVP@mapfont\@nil\else
1777   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
1778   {\bbl@error{Option `bbl@KVP@mapfont' unknown for\\%
1779     mapfont. Use `direction'.%
1780     {See the manual for details.}}}%
1781 \bbl@ifunset{bbl@lsys@\\languagename}{\bbl@provide@lsys{\\languagename}}}%
1782 \bbl@ifunset{bbl@wdir@\\languagename}{\bbl@provide@dirs{\\languagename}}}%
1783 \ifx\bbl@mapselect\@undefined
1784   \AtBeginDocument{%
1785     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1786   {\selectfont}}%
1787 \def\bbl@mapselect{%
1788   \let\bbl@mapselect\relax
1789   \edef\bbl@prefontid{\fontid\font}%
1790   \def\bbl@mapdir##1{%
1791     {\def\\languagename{##1}%
1792       \let\bbl@ifrestoring\@firstoftwo % avoid font warning
1793       \bbl@switchfont

```

```

1794 \directlua{Babel.fontmap
1795 [\the\csname bbl@wdir@##1\endcsname]%
1796 [\bbl@prefontid]=\fontid\font}}}%
1797 \fi
1798 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language\language}}}%
1799 \fi
1800 % For East Asian, Southeast Asian, if interspace in ini - TODO: as hook?
1801 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
1802 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
1803 \fi
1804 \ifcase\bbl@engine\or
1805 \bbl@ifunset{bbl@intsp@\language\language}}{%
1806 {\expandafter\ifx\csname bbl@intsp@\language\language\endcsname\@empty\else
1807 \bbl@xin@{\bbl@cs{sbcp@\language\language}}{Hant,Hans,Jpan,Kore,Kana}%
1808 \ifin@
1809 \bbl@cjkintraspace
1810 \directlua{
1811 Babel = Babel or {}
1812 Babel.locale_props = Babel.locale_props or {}
1813 Babel.locale_props[\the\localeid].linebreak = 'c'
1814 }%
1815 \bbl@exp{\bbl@intraspace\bbl@cs{intsp@\language\language}}\@{}%
1816 \ifx\bbl@KVP@intrapenalty\@nil
1817 \bbl@intrapenalty0\@@
1818 \fi
1819 \else
1820 \bbl@seaintraspace
1821 \bbl@exp{\bbl@intraspace\bbl@cs{intsp@\language\language}}\@{}%
1822 \directlua{
1823 Babel = Babel or {}
1824 Babel.sea_ranges = Babel.sea_ranges or {}
1825 Babel.set_chranges('\bbl@cs{sbcp@\language\language}',
1826 '\bbl@cs{chrng@\language\language}')
1827 }%
1828 \ifx\bbl@KVP@intrapenalty\@nil
1829 \bbl@intrapenalty0\@@
1830 \fi
1831 \fi
1832 \fi
1833 \ifx\bbl@KVP@intrapenalty\@nil\else
1834 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
1835 \fi}%
1836 \or
1837 \bbl@xin@{\bbl@cs{sbcp@\language\language}}{Thai,Lao,Khmr}%
1838 \ifin@
1839 \bbl@ifunset{bbl@intsp@\language\language}}{%
1840 {\expandafter\ifx\csname bbl@intsp@\language\language\endcsname\@empty\else
1841 \ifx\bbl@KVP@intraspace\@nil
1842 \bbl@exp{%
1843 \bbl@intraspace\bbl@cs{intsp@\language\language}}\@{}%
1844 \fi
1845 \ifx\bbl@KVP@intrapenalty\@nil
1846 \bbl@intrapenalty0\@@
1847 \fi
1848 \fi
1849 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
1850 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
1851 \fi
1852 \ifx\bbl@KVP@intrapenalty\@nil\else

```

```

1853     \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
1854   \fi
1855   \ifx\bbbl@ispace\@undefined
1856     \AtBeginDocument{%
1857       \expandafter\bbbl@add
1858       \csname selectfont \endcsname{\bbbl@ispace}%
1859       \def\bbbl@ispace{\bbbl@cs{xeisp@\bbbl@cs{sbcp@language}}}%
1860     \fi}%
1861   \fi
1862 \fi
1863 % Native digits, if provided in ini (TeX level, xe and lua)
1864 \ifcase\bbbl@engine\else
1865   \bbbl@ifunset{\bbbl@dgnat@language}{}%
1866   {\expandafter\ifx\csname \bbbl@dgnat@language\endcsname\@empty\else
1867     \expandafter\expandafter\expandafter
1868     \bbbl@setdigits\csname \bbbl@dgnat@language\endcsname
1869     \ifx\bbbl@KVP@maparabic\@nil\else
1870       \ifx\bbbl@latinarabic\@undefined
1871         \expandafter\let\expandafter\@arabic
1872         \csname \bbbl@counter@language\endcsname
1873       \else % ie, if layout=counters, which redefines \@arabic
1874         \expandafter\let\expandafter\bbbl@latinarabic
1875         \csname \bbbl@counter@language\endcsname
1876       \fi
1877     \fi
1878   \fi}%
1879 \fi
1880 % Native digits (lua level).
1881 \ifodd\bbbl@engine
1882   \ifx\bbbl@KVP@mapdigits\@nil\else
1883     \bbbl@ifunset{\bbbl@dgnat@language}{}%
1884     {\RequirePackage{luatexbase}%
1885       \bbbl@activate@preotf
1886       \directlua{
1887         Babel = Babel or {} %%% -> presets in luababel
1888         Babel.digits_mapped = true
1889         Babel.digits = Babel.digits or {}
1890         Babel.digits[\the\localeid] =
1891           table.pack(string.utfvalue('\bbbl@cs{dgnat@language}'))
1892         if not Babel.numbers then
1893           function Babel.numbers(head)
1894             local LOCALE = luatexbase.registernumber'\bbbl@attr@locale'
1895             local GLYPH = node.id'glyph'
1896             local inmath = false
1897             for item in node.traverse(head) do
1898               if not inmath and item.id == GLYPH then
1899                 local temp = node.get_attribute(item, LOCALE)
1900                 if Babel.digits[temp] then
1901                   local chr = item.char
1902                   if chr > 47 and chr < 58 then
1903                     item.char = Babel.digits[temp][chr-47]
1904                   end
1905                 end
1906               elseif item.id == node.id'math' then
1907                 inmath = (item.subtype == 0)
1908               end
1909             end
1910             return head
1911           end

```



```

1965     \ifx##1\@empty\else
1966     \bbl@exp{%
1967         \\\SetString\\##1{%
1968             \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
1969     \expandafter\bbl@tempb
1970     \fi}%
1971     \expandafter\bbl@tempb\bbl@captionslist\@empty
1972 \else
1973     \bbl@read@ini{\bbl@KVP@captions}% Here all letters cat = 11
1974     \bbl@after@ini
1975     \bbl@savestrings
1976     \fi
1977 \StartBabelCommands*{#1}{date}%
1978 \ifx\bbl@KVP@import\@nil
1979     \bbl@exp{%
1980         \\\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
1981     \else
1982         \bbl@savetoday
1983         \bbl@savedate
1984     \fi
1985 \EndBabelCommands
1986 \bbl@exp{%
1987     \def\<#1hyphenmins>{%
1988         {\bbl@ifunset{\bbl@ifthm@#1}{2}{\@nameuse{\bbl@ifthm@#1}}}%
1989         {\bbl@ifunset{\bbl@rgthm@#1}{3}{\@nameuse{\bbl@rgthm@#1}}}}}%
1990 \bbl@provide@hyphens{#1}%
1991 \ifx\bbl@KVP@main\@nil\else
1992     \expandafter\main@language\expandafter{#1}%
1993 \fi}
1994 \def\bbl@provide@renew#1{%
1995     \ifx\bbl@KVP@captions\@nil\else
1996         \StartBabelCommands*{#1}{captions}%
1997         \bbl@read@ini{\bbl@KVP@captions}% Here all letters cat = 11
1998         \bbl@after@ini
1999         \bbl@savestrings
2000         \EndBabelCommands
2001     \fi
2002     \ifx\bbl@KVP@import\@nil\else
2003         \StartBabelCommands*{#1}{date}%
2004         \bbl@savetoday
2005         \bbl@savedate
2006         \EndBabelCommands
2007     \fi
2008     \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2009 \def\bbl@provide@hyphens#1{%
2010     \let\bbl@tempa\relax
2011     \ifx\bbl@KVP@hyphenrules\@nil\else
2012         \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2013         \bbl@foreach\bbl@KVP@hyphenrules{%
2014             \ifx\bbl@tempa\relax % if not yet found
2015                 \bbl@ifsamestring{##1}{+}%
2016                 {\bbl@exp{\\addlanguage\<l@##1>}}}%
2017                 {}%
2018             \bbl@ifunset{l@##1}%
2019             {}%
2020             {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
2021         \fi}%

```



```

2022 \fi
2023 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
2024 \ifx\bbbl@KVP@import\@nil\else % if importing
2025 \bbbl@exp{% and hyphenrules is not empty
2026 \\\bbbl@ifblank{\@nameuse{bbbl@hyphr@#1}}}%
2027 }%
2028 {\let\\bbbl@tempa<l@\\@nameuse{bbbl@hyphr@\\language}%}}%
2029 \fi
2030 \fi
2031 \bbbl@ifunset{bbbl@tempa}% ie, relax or undefined
2032 {\bbbl@ifunset{l@#1}% no hyphenrules found - fallback
2033 {\bbbl@exp{\\adddialect<l@#1>\\language}}}%
2034 }% so, l@<lang> is ok - nothing to do
2035 {\bbbl@exp{\\adddialect<l@#1>\bbbl@tempa}}% found in opt list or ini
2036 \bbbl@ifunset{bbbl@prehc@\\language}%
2037 }% TODO - XeTeX, based on \babelfont and HyphenChar?
2038 {\ifodd\bbbl@engine\bbbl@exp{%
2039 \\\bbbl@ifblank{\@nameuse{bbbl@prehc@#1}}}%
2040 }%
2041 {\\\AddBabelHook[\\language]{babel-prehc-\\language}{patterns}%
2042 {\prehyphenchar=\@nameuse{bbbl@prehc@\\language}\relax}}%
2043 \fi}}

```

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

```

2044 \def\bbbl@read@ini#1{%
2045 \openin1=babel-#1.ini % FIXME - number must not be hardcoded
2046 \ifeof1
2047 \bbbl@error
2048 {There is no ini file for the requested language\\
2049 (#1). Perhaps you misspelled it or your installation\\
2050 is not complete.}%
2051 {Fix the name or reinstall babel.}%
2052 \else
2053 \let\bbbl@section\@empty
2054 \let\bbbl@savestrings\@empty
2055 \let\bbbl@savetoday\@empty
2056 \let\bbbl@savestate\@empty
2057 \def\bbbl@inipreread##1=##2\@{%
2058 \bbbl@trim\def\bbbl@tempa{##1}% Redundant below !!
2059 % Move trims here ??
2060 \bbbl@ifunset{bbbl@KVP@\bbbl@section.\bbbl@tempa}%
2061 {\expandafter\bbbl@inireader\bbbl@tempa=##2\@}%
2062 {\def\bbbl@tempc{\expandafter\bbbl@inireader\bbbl@tempa}%
2063 \expandafter\expandafter\expandafter
2064 \bbbl@tempc\csname bbbl@KVP@\bbbl@section.\bbbl@tempa\endcsname\@}%
2065 \global\bbbl@csarg\let{KVP@\bbbl@section.\bbbl@tempa}\relax}%
2066 \let\bbbl@inireader\bbbl@iniskip
2067 \bbbl@info{Importing data from babel-#1.ini for \\language}%
2068 \loop
2069 \if T\ifeof1F\fi T\relax % Trick, because inside \loop
2070 \endlinechar\m@ne
2071 \read1 to \bbbl@line
2072 \endlinechar\^^M
2073 \ifx\bbbl@line\@empty\else
2074 \expandafter\bbbl@iniline\bbbl@line\bbbl@iniline
2075 \fi
2076 \repeat
2077 \fi}

```

```

2078 \def\bbl@inline#1\bbl@inline{%
2079   \ifnextchar[\bbl@inisec{\ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@@}% ]

```

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

```

2080 \def\bbl@iniskip#1\@@{%      if starts with ;
2081 \def\bbl@inisec[#1]#2\@@{%   if starts with opening bracket
2082   \@nameuse{bbl@secpost@\bbl@section}% ends previous section
2083   \def\bbl@section{#1}%
2084   \@nameuse{bbl@secpre@\bbl@section}% starts current section
2085   \bbl@ifunset{bbl@inikv@#1}%
2086   {\let\bbl@inireader\bbl@iniskip}%
2087   {\bbl@exp{\let\\bbl@inireader<\bbl@inikv@#1>}}}

```

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

```

2088 \def\bbl@inikv#1=#2\@@{%    key=value
2089   \bbl@trim@def\bbl@tempa{#1}%
2090   \bbl@trim\toks@{#2}%
2091   \bbl@csarg\edef{@kv@\bbl@section.\bbl@tempa}{\the\toks@}}

```

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

```

2092 \def\bbl@exportkey#1#2#3{%
2093   \bbl@ifunset{bbl@kv@#2}%
2094   {\bbl@csarg\gdef{#1@\language}{#3}}%
2095   {\expandafter\ifx\csname bbl@kv@#2\endcsname\empty
2096     \bbl@csarg\gdef{#1@\language}{#3}}%
2097   \else
2098     \bbl@exp{\global\let<\bbl@#1@\language><\bbl@kv@#2>}%
2099     \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.

```

2100 \let\bbl@inikv@identification\bbl@inikv
2101 \def\bbl@secpost@identification{%
2102   \bbl@exportkey{lname}{identification.name.english}{}%
2103   \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
2104   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2105   \bbl@exportkey{sname}{identification.script.name}{}%
2106   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2107   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
2108 \let\bbl@inikv@typography\bbl@inikv
2109 \let\bbl@inikv@characters\bbl@inikv
2110 \let\bbl@inikv@numbers\bbl@inikv
2111 \def\bbl@after@ini{%
2112   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2113   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2114   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2115   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2116   \bbl@exportkey{intsp}{typography.intraspace}{}%
2117   \bbl@exportkey{jstfy}{typography.justify}{w}%
2118   \bbl@exportkey{chrng}{characters.ranges}{}%
2119   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2120   \bbl@exportkey{rtex}{identification.require.babel}{}%
2121   \bbl@xin@{0.5}{\@nameuse{bbl@kv@identification.version}}%
2122   \ifin@
2123     \bbl@warning{%
2124       There are neither captions nor date in `language'.\%

```

```

2125     It may not be suitable for proper typesetting, and it\\%
2126     could change. Reported}%
2127 \fi
2128 \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
2129 \ifin@
2130     \bbl@warning{%
2131         The ``\language' date format may not be suitable\\%
2132         for proper typesetting, and therefore it very likely will\\%
2133         change in a future release. Reported}%
2134 \fi
2135 \bbl@toglobal\bbl@savetoday
2136 \bbl@toglobal\bbl@savestate}

```

Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

```

2137 \ifcase\bbl@engine
2138     \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
2139         \bbl@ini@captions@aux{#1}{#2}}
2140 \else
2141     \def\bbl@inikv@captions#1=#2\@@{%
2142         \bbl@ini@captions@aux{#1}{#2}}
2143 \fi

```

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

```

2144 \def\bbl@ini@captions@aux#1#2{%
2145     \bbl@trim@def\bbl@tempa{#1}%
2146     \bbl@ifblank{#2}%
2147     {\bbl@exp{%
2148         \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2149     {\bbl@trim\toks@{#2}}}%
2150 \bbl@exp{%
2151     \bbl@add\bbl@savestrings{%
2152         \SetString\<\bbl@tempa name>{\the\toks@}}}

```

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

TODO. Remove cypypaste pattern.

```

2153 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%           for defaults
2154     \bbl@inidate#1...\relax{#2}}
2155 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
2156     \bbl@inidate#1...\relax{#2}{islamic}}
2157 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
2158     \bbl@inidate#1...\relax{#2}{hebrew}}
2159 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
2160     \bbl@inidate#1...\relax{#2}{persian}}
2161 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
2162     \bbl@inidate#1...\relax{#2}{indian}}
2163 \ifcase\bbl@engine
2164     \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%   override
2165         \bbl@inidate#1...\relax{#2}}
2166     \bbl@csarg\def{secpre@date.gregorian.licr}{%           discard uni
2167         \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
2168 \fi
2169 % eg: 1=months, 2=wide, 3=1, 4=dummy
2170 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2171     \bbl@trim@def\bbl@tempa{#1.#2}%
2172     \bbl@ifsamestring{\bbl@tempa}{months.wide}%           to savedate

```

```

2173 {\bbl@trim@def\bbl@tempa{#3}%
2174 \bbl@trim\toks@{#5}%
2175 \bbl@exp{%
2176 \\\bbl@add\\\bbl@savestate{%
2177 \\\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}}}%
2178 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
2179 {\bbl@trim@def\bbl@toreplace{#5}%
2180 \bbl@TG@date
2181 \global\bbl@csarg\let{date@\language}\bbl@toreplace
2182 \bbl@exp{%
2183 \gdef\<\language date>\protect\<\language date >}%
2184 \gdef\<\language date >####1####2####3{%
2185 \\\bbl@usedategrouptrue
2186 \<bbl@ensure@\language>{%
2187 \<bbl@date@\language>{####1}{####2}{####3}}}%
2188 \\\bbl@add\\\bbl@savetoday{%
2189 \\\SetString\the\today{%
2190 \<\language date>\the\year\the\month\the\day}}}%
2191 {}

```

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.

```

2192 \let\bbl@calendar\empty
2193 \newcommand\BabelDateSpace{\nobreakspace}
2194 \newcommand\BabelDateDot{.\@}
2195 \newcommand\BabelDated[1]{\number#1}
2196 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
2197 \newcommand\BabelDateM[1]{\number#1}
2198 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2199 \newcommand\BabelDateMMMM[1]{%
2200 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
2201 \newcommand\BabelDatey[1]{\number#1}%
2202 \newcommand\BabelDateyy[1]{%
2203 \ifnum#1<10 0\number#1 %
2204 \else\ifnum#1<100 \number#1 %
2205 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2206 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2207 \else
2208 \bbl@error
2209 {Currently two-digit years are restricted to the\
2210 range 0-9999.}%
2211 {There is little you can do. Sorry.}%
2212 \fi\fi\fi}}
2213 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2214 \def\bbl@replace@finish@iii#1{%
2215 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
2216 \def\bbl@TG@date{%
2217 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
2218 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
2219 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
2220 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
2221 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
2222 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
2223 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
2224 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
2225 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
2226 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
2227 % Note after \bbl@replace \toks@ contains the resulting string.

```

```

2228% TODO - Using this implicit behavior doesn't seem a good idea.
2229 \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.

```

2230 \def\bbl@provide@lsys#1{%
2231   \bbl@ifunset{\bbl@lname@#1}%
2232     {\bbl@ini@ids{#1}}%
2233   }%
2234   \bbl@csarg\let{lsys@#1}\empty
2235   \bbl@ifunset{\bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
2236   \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
2237   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2238   \bbl@ifunset{\bbl@lname@#1}{}%
2239   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2240   \bbl@csarg\bbl@toGLOBAL{lsys@#1}}

```

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

```

2241 \def\bbl@ini@ids#1{%
2242   \def\BabelBeforeIni##1##2{%
2243     \begingroup
2244       \bbl@add\bbl@secpost@identification{\closein1 }%
2245       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
2246       \bbl@read@ini{##1}%
2247     \endgroup}%
2248   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%

```

10 The kernel of Babel (babel.def, only L^AT_EX)

10.1 The redefinition of the style commands

The rest of the code in this file can only be processed by L^AT_EX, so we check the current format. If it is plain T_EX, 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 T_EX 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`.

```

2249 {\def\format{lplain}
2250 \ifx\fmtname\format
2251 \else
2252   \def\format{LaTeX2e}
2253   \ifx\fmtname\format
2254   \else
2255     \aftergroup\endinput
2256   \fi
2257 \fi}

```

10.2 Cross referencing macros

The L^AT_EX book states:

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

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

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

The only way to accomplish this in most cases is to use the trick described in the *T_EXbook* [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.

```
2258 %\bbl@redefine\newlabel#1#2{%
2259 % \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

`\@newl@bel` We need to change the definition of the *L^AT_EX*-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.

```
2260 <<*More package options>> ≡
2261 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2262 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2263 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2264 <</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.

```
2265 \bbl@trace{Cross referencing macros}
2266 \ifx\bbl@opt@safe\@empty\else
2267 \def\@newl@bel#1#2#3{%
2268   {\@safe@activestrue
2269     \bbl@ifunset{#1@#2}%
2270     \relax
2271     {\gdef\@multiplelabels{%
2272       \@latex@warning@no@line{There were multiply-defined labels}}}%
2273     \@latex@warning@no@line{Label `#2' multiply defined}}%
2274   \global\@namedef{#1@#2}{#3}}}
```

`\@testdef` An internal *L^AT_EX* 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 *L^AT_EX* keeps reporting that the labels may have changed.

```
2275 \CheckCommand*\@testdef[3]{%
2276   \def\reserved@a{#3}%
2277   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2278   \else
2279     \@tempwattrue
2280   \fi}
```

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

```
2281 \def\@testdef#1#2#3{%
2282   \@safe@activestrue
```

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

```
2283 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```
2284 \def\bbl@tempb{#3}%
```

```
2285 \@safe@activesfalse
```

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

```
2286 \ifx\bbl@tempa\relax
```

```
2287 \else
```

```
2288 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
```

```
2289 \fi
```

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

```
2290 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```
2291 \ifx\bbl@tempa\bbl@tempb
```

```
2292 \else
```

```
2293 \@tempswatrue
```

```
2294 \fi}
```

```
2295 \fi
```

`\ref` 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.

```
2296 \bbl@xin@{R}\bbl@opt@safe
```

```
2297 \ifin@
```

```
2298 \bbl@redefineroobust\ref{#1}%
```

```
2299 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
```

```
2300 \bbl@redefineroobust\pageref{#1}%
```

```
2301 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
```

```
2302 \else
```

```
2303 \let\org@ref\ref
```

```
2304 \let\org@pageref\pageref
```

```
2305 \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.

```
2306 \bbl@xin@{B}\bbl@opt@safe
```

```
2307 \ifin@
```

```
2308 \bbl@redefine\@citex[#1]#2{%
```

```
2309 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
```

```
2310 \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.

```
2311 \AtBeginDocument{%
```

```
2312 \@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.)

```
2313 \def\@citex[#1][#2]#3{%
2314   \@safe@activetrue\edef\@tempa{#3}\@safe@activetruefalse
2315   \org@citex[#1][#2]{\@tempa}}%
2316 }
```

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

```
2317 \AtBeginDocument{%
2318   \ifpackageloaded{cite}{%
2319     \def\@citex[#1]#2{%
2320       \@safe@activetrue\org@citex[#1]#2\@safe@activetruefalse}%
2321     }{}}
```

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

```
2322 \bbl@redefine\nocite#1{%
2323   \@safe@activetrue\org@nocite{#1}\@safe@activetruefalse}
```

`\bibcite` The macro that is used in the `.aux` file to define citation labels. When packages such as `natbib` or `cite` are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where `\@safe@activetrue` is in effect. This switch needs to be reset inside the `\hbox` which contains the citation label. In order to determine during `.aux` file processing which definition of `\bibcite` is needed we define `\bibcite` in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
2324 \bbl@redefine\bibcite{%
2325   \bbl@cite@choice
2326   \bibcite}
```

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

```
2327 \def\bbl@bibcite#1#2{%
2328   \org@bibcite{#1}{\@safe@activetruefalse#2}}
```

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

```
2329 \def\bbl@cite@choice{%
2330   \global\let\bibcite\bbl@bibcite}
```

Then, when `natbib` is loaded we restore the original definition of `\bibcite`. For `cite` we do the same.

```
2331 \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
2332   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%}
```

Make sure this only happens once.

```
2333 \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.

```
2334 \AtBeginDocument{\bbl@cite@choice}
```


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

```

2335 \bbl@redefine\@bibitem#1{%
2336   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
2337 \else
2338   \let\org@nocite\nocite
2339   \let\org@@citex\@citex
2340   \let\org@bibcite\bibcite
2341   \let\org@bibitem\@bibitem
2342 \fi

```

10.3 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat. We check whether the argument is empty; if it is, we just make sure the scratch token register is empty. Next, we store the argument to `\markright` in the scratch token register. This way these commands will not be expanded later, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while `\@safe@activetrue` is in effect.

```

2343 \bbl@trace{Marks}
2344 \IfBabelLayout{sectioning}
2345   {\ifx\bbl@opt@headfoot\@nnil
2346     \g@addto@macro\@resetactivechars{%
2347       \set@typeset@protect
2348       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2349       \let\protect\noexpand
2350       \edef\thepage{%
2351         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2352     \fi}
2353   {\bbl@redefine\markright#1{%
2354     \bbl@ifblank{#1}%
2355     {\org@markright{}}%
2356     {\toks@{#1}%
2357       \bbl@exp{%
2358         \\org@markright{\\protect\\foreignlanguage{\language}%
2359           {\\\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`.

`\@mkboth`

```

2360 \ifx\@mkboth\markboth
2361   \def\bbl@tempc{\let\@mkboth\markboth}
2362 \else
2363   \def\bbl@tempc{}
2364 \fi

```

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

```

2365 \bbl@redefine\markboth#1#2{%
2366   \protected@edef\bbl@tempb##1{%
2367     \protect\foreignlanguage
2368     {\language}{\protect\bbl@restore@actives##1}}%

```

```

2369 \bbl@ifblank{#1}%
2370 {\toks@{}}%
2371 {\toks@expandafter{\bbl@tempb{#1}}}%
2372 \bbl@ifblank{#2}%
2373 {\@temptokena{}}%
2374 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2375 \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}

```

and copy it to \@mkboth if necessary.

```

2376 \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.

```

2377 \bbl@trace{Preventing clashes with other packages}
2378 \bbl@xin@{R}\bbl@opt@safe
2379 \ifin@
2380 \AtBeginDocument{%
2381 \ifpackageloaded{ifthen}{%

```

Then we can redefine `\ifthenelse`:

```

2382 \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.

```

2383 \let\bbl@temp@pref\pageref
2384 \let\pageref\org@pageref
2385 \let\bbl@temp@ref\ref
2386 \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.

```

2387 \@safe@activetrue
2388 \org@ifthenelse{#1}%
2389 {\let\pageref\bbl@temp@pref
2390 \let\ref\bbl@temp@ref
2391 \@safe@activesfalse
2392 #2}%
2393 {\let\pageref\bbl@temp@pref
2394 \let\ref\bbl@temp@ref
2395 \@safe@activesfalse
2396 #3}%

```

```

2397     }%
2398   }{}%
2399 }

```

10.4.2 varioref

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

```

2400 \AtBeginDocument{%
2401   \ifpackageloaded{varioref}{%
2402     \bbl@redefine\@@vpageref#1[#2]#3{%
2403       \safe@activetrue
2404       \org@@vpageref{#1}[#2]{#3}%
2405       \safe@activesfalse}%
2406     \bbl@redefine\vrefpagenum#1#2{%
2407       \safe@activetrue
2408       \org@vrefpagenum{#1}#2}%
2409     \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.

```

2410   \expandafter\def\csname Ref \endcsname#1{%
2411     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2412   }{}%
2413 }
2414 \fi

```

10.4.3 hhline

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

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

```

2415 \AtEndOfPackage{%
2416   \AtBeginDocument{%
2417     \ifpackageloaded{hhline}%

```

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

```

2418     {\expandafter\ifx\csname normal@char\string\endcsname\relax
2419       \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.

```

2420       \makeatletter
2421       \def\@currname{hhline}\input{hhline.sty}\makeatother
2422     \fi}%
2423   {}}}

```

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.

```
2424 \AtBeginDocument{%
2425   \ifx\pdfstringdefDisableCommands\@undefined\else
2426     \pdfstringdefDisableCommands{\languageshortands{system}}%
2427   \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 `\MakeUpper` case. To prevent unexpected results we need to define `\FOREIGNLANGUAGE` here.

```
2428 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2429   \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.

```
2430 \def\substitutefontfamily#1#2#3{%
2431   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2432   \immediate\write15{%
2433     \string\ProvidesFile{#1#2.fd}%
2434     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2435     \space generated font description file]^{}
2436     \string\DeclareFontFamily{#1}{#2}{}}^{}
2437     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^{}
2438     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^{}
2439     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^{}
2440     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^{}
2441     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^{}
2442     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^{}
2443     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^{}
2444     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^{}
2445   }%
2446   \closeout15
2447 }
```

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

```
2448 \@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
```

```
2449 \bbl@trace{Encoding and fonts}
```

```

2450 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
2451 \newcommand\BabelNonText{TS1,T3,TS3}
2452 \let\org@TeX\TeX
2453 \let\org@LaTeX\LaTeX
2454 \let\ensureasci\@firstofone
2455 \AtBeginDocument{%
2456   \in@false
2457   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2458     \ifin@false
2459       \lowercase{\bbl@xin@{,#1enc.def,},{,\@filelist,}}%
2460     \fi}%
2461   \ifin@ % if a text non-ascii has been loaded
2462     \def\ensureasci#1{{\fontencoding{OT1}\selectfont#1}}%
2463     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2464     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2465     \def\bbl@tempb#1\@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
2466     \def\bbl@tempc#1ENC.DEF#2\@{\%
2467       \ifx\@empty#2\else
2468         \bbl@ifunset{T@#1}%
2469         {}%
2470         {\bbl@xin@{,#1,},{,\BabelNonASCII,\BabelNonText,}}%
2471       \ifin@
2472         \DeclareTextCommand{\TeX}{#1}{\ensureasci{\org@TeX}}%
2473         \DeclareTextCommand{\LaTeX}{#1}{\ensureasci{\org@LaTeX}}%
2474       \else
2475         \def\ensureasci#1{{\fontencoding{#1}\selectfont#1}}%
2476       \fi}%
2477     \fi}%
2478   \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@ de mas??
2479   \bbl@xin@{,\cf@encoding,},{,\BabelNonASCII,\BabelNonText,}%
2480   \ifin@false
2481     \edef\ensureasci#1{%
2482       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2483   \fi
2484 \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.

```

2485 \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.

```

2486 \AtBeginDocument{%
2487   \@ifpackageloaded{fontspec}%
2488     {\xdef\latinencoding%
2489       \ifx\UTFencname\undefined
2490         EU\ifcase\bbl@engine\or2\or1\fi
2491       \else
2492         \UTFencname
2493       \fi}}%
2494     {\gdef\latinencoding{OT1}}%

```

```

2495 \ifx\cf@encoding\bbl@t@one
2496 \xdef\latinencoding{\bbl@t@one}%
2497 \else
2498 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
2499 \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.

```

2500 \DeclareRobustCommand{\latintext}{%
2501 \fontencoding{\latinencoding}\selectfont
2502 \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.

```

2503 \ifx\@undefined\DeclareTextFontCommand
2504 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2505 \else
2506 \DeclareTextFontCommand{\textlatin}{\latintext}
2507 \fi

```

10.6 Basic bidi support

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

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

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

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

```

2508 \bbl@trace{Basic (internal) bidi support}
2509 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2510 \def\bbl@rscripts{%
2511 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2512 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
2513 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
2514 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2515 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2516 Old South Arabian,}%

```

```

2517 \def\bbl@provide@dirs#1{%
2518   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2519   \ifin@
2520     \global\bbl@csarg\chardef{wdir@#1}\@ne
2521     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2522     \ifin@
2523       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2524     \fi
2525   \else
2526     \global\bbl@csarg\chardef{wdir@#1}\z@
2527   \fi
2528   \ifodd\bbl@engine
2529     \bbl@csarg\ifcase{wdir@#1}%
2530       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
2531     \or
2532       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
2533     \or
2534       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
2535     \fi
2536   \fi}
2537 \def\bbl@switchdir{%
2538   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2539   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2540   \bbl@exp{\bbl@setdirs\bbl@cs{wdir@\languagename}}
2541 \def\bbl@setdirs#1{% TODO - math
2542   \ifcase\bbl@select@type % TODO - strictly, not the right test
2543     \bbl@bodydir{#1}%
2544     \bbl@pardir{#1}%
2545   \fi
2546   \bbl@texmdir{#1}}
2547 \ifodd\bbl@engine % luatex=1
2548   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2549   \DisableBabelHook{babel-bidi}
2550   \chardef\bbl@thetexmdir\z@
2551   \chardef\bbl@thepardir\z@
2552   \def\bbl@getluadir#1{%
2553     \directlua{
2554       if tex.#1dir == 'TLT' then
2555         tex.sprint('0')
2556       elseif tex.#1dir == 'TRT' then
2557         tex.sprint('1')
2558       end}}
2559   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 r1
2560     \ifcase#3\relax
2561       \ifcase\bbl@getluadir{#1}\relax\else
2562         #2 TLT\relax
2563       \fi
2564     \else
2565       \ifcase\bbl@getluadir{#1}\relax
2566         #2 TRT\relax
2567       \fi
2568     \fi}
2569   \def\bbl@texmdir#1{%
2570     \bbl@setluadir{tex}\texmdir{#1}%
2571     \chardef\bbl@thetexmdir#1\relax
2572     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2573   \def\bbl@pardir#1{%
2574     \bbl@setluadir{par}\pardir{#1}%
2575     \chardef\bbl@thepardir#1\relax}

```

```

2576 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2577 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2578 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
2579 % Sadly, we have to deal with boxes in math with basic.
2580 % Activated every math with the package option bidi=:
2581 \def\bbl@mathboxdir{%
2582   \ifcase\bbl@thetextdir\relax
2583     \everyhbox{\textdir TLT\relax}%
2584   \else
2585     \everyhbox{\textdir TRT\relax}%
2586   \fi}
2587 \else % pdftex=0, xetex=2
2588   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2589   \DisableBabelHook{babel-bidi}
2590   \newcount\bbl@dirlevel
2591   \chardef\bbl@thetextdir\z@
2592   \chardef\bbl@thepardir\z@
2593   \def\bbl@textdir#1{%
2594     \ifcase#1\relax
2595       \chardef\bbl@thetextdir\z@
2596       \bbl@textdir@i\beginL\endL
2597     \else
2598       \chardef\bbl@thetextdir\@ne
2599       \bbl@textdir@i\beginR\endR
2600     \fi}
2601   \def\bbl@textdir@i#1#2{%
2602     \ifhmode
2603       \ifnum\currentgrouplevel>\z@
2604         \ifnum\currentgrouplevel=\bbl@dirlevel
2605           \bbl@error{Multiple bidi settings inside a group}%
2606           {I'll insert a new group, but expect wrong results.}%
2607           \bgroup\aftergroup#2\aftergroup\egroup
2608         \else
2609           \ifcase\currentgrouptype\or % 0 bottom
2610             \aftergroup#2% 1 simple {}
2611           \or
2612             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2613           \or
2614             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2615           \or\or\or % vbox vtop align
2616           \or
2617             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2618           \or\or\or\or\or\or % output math disc insert vcent mathchoice
2619           \or
2620             \aftergroup#2% 14 \begingroup
2621           \else
2622             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2623           \fi
2624         \fi
2625         \bbl@dirlevel\currentgrouplevel
2626       \fi
2627       #1%
2628     \fi}
2629   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2630   \let\bbl@bodydir\@gobble
2631   \let\bbl@pagedir\@gobble
2632   \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).

```

2633 \def\bbl@xebidipar{%
2634   \let\bbl@xebidipar\relax
2635   \TeXeTstate\@ne
2636   \def\bbl@xeverypar{%
2637     \ifcase\bbl@thepardir
2638       \ifcase\bbl@thetextdir\else\beginR\fi
2639     \else
2640       {\setbox\z@\lastbox\beginR\box\z@}%
2641     \fi}%
2642   \let\bbl@severypar\everypar
2643   \newtoks\everypar
2644   \everypar=\bbl@severypar
2645   \bbl@severypar{\bbl@xeverypar\the\everypar}}
2646 \@ifpackagewith{babel}{bidi=bidi}%
2647 {\let\bbl@textdir\i@gobbletwo
2648   \let\bbl@xebidipar\@empty
2649   \AddBabelHook{bidi}{foreign}{%
2650     \def\bbl@tempa{\def\BabelText###1}%
2651     \ifcase\bbl@thetextdir
2652       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
2653     \else
2654       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
2655     \fi}
2656   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}}
2657 {}%
2658 \fi

```

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

```

2659 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
2660 \AtBeginDocument{%
2661   \ifx\pdfstringdefDisableCommands\@undefined\else
2662     \ifx\pdfstringdefDisableCommands\relax\else
2663       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2664     \fi
2665   \fi}

```

10.7 Local Language Configuration

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

```

2666 \bbl@trace{Local Language Configuration}
2667 \ifx\loadlocalcfg\@undefined
2668   \@ifpackagewith{babel}{noconfigs}%
2669   {\let\loadlocalcfg\@gobble}%
2670   {\def\loadlocalcfg#1{%
2671     \InputIfFileExists{#1.cfg}%
2672     {\typeout{*****^J%
2673               * Local config file #1.cfg used^^J%
2674             *}}}%
2675     \@empty}}
2676 \fi

```

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

```

2677 \ifx\@unexpandable@protect\@undefined
2678   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2679   \long\def\protected@write#1#2#3{%
2680     \begingroup
2681       \let\thepage\relax
2682       #2%
2683       \let\protect\@unexpandable@protect
2684       \edef\reserved@a{\write#1{#3}}%
2685       \reserved@a
2686     \endgroup
2687     \if@nobreak\ifvmode\nobreak\fi\fi}
2688 \fi
2689 </core>
2690 <*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.

```

2691 <<Make sure ProvidesFile is defined>>
2692 \ProvidesFile{switch.def}[\<date>] [\<version>] Babel switching mechanism]
2693 <<Load macros for plain if not LaTeX>>
2694 <<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.

```

2695 \def\bbl@version{\<version>}
2696 \def\bbl@date{\<date>}
2697 \def\adddialect#1#2{%
2698   \global\chardef#1#2\relax
2699   \bbl@usehooks{adddialect}{\#1}{\#2}}%
2700   \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.

```

2701 \def\bbl@fixname#1{%
2702   \begingroup
2703     \def\bbl@tempe{l@}%
2704     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2705     \bbl@tempd
2706     {\lowercase\expandafter{\bbl@tempd}%
2707      \uppercase\expandafter{\bbl@tempd}%
2708      \@empty
2709      {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2710       \uppercase\expandafter{\bbl@tempd}}}%
2711     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2712      \lowercase\expandafter{\bbl@tempd}}}%
2713   \@empty

```

```

2714 \edef\bb1@tempd{\endgroup\def\noexpand#1{#1}}%
2715 \bb1@tempd}
2716 \def\bb1@iflanguage#1{%
2717 \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

```

2718 \def\iflanguage#1{%
2719 \bb1@iflanguage{#1}{%
2720 \ifnum\csname l@#1\endcsname=\language
2721 \expandafter\@firstoftwo
2722 \else
2723 \expandafter\@secondoftwo
2724 \fi}}

```

11.1 Selecting the language

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

To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character. To convert a control sequence to a string, we use the `\string` primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer `\escapechar` to a character number, we have to compare this number with the character of the string. To do this we have to use \TeX 's backquote notation to specify the character as a number. If the first character of the `\string`'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or `\escapechar` is set to a value outside of the character range 0–255. If the user gives an empty argument, we provide a default argument for `\string`. This argument should expand to nothing.

```

2725 \let\bb1@select@type\z@
2726 \edef\selectlanguage{%
2727 \noexpand\protect
2728 \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

2729 \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.

```

2730 \ifx\documentclass\@undefined
2731 \def\xstring{\string\string\string}
2732 \else
2733 \let\xstring\string
2734 \fi

```

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

`\bbl@pop@language` But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need T_EX'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.

```
2735 \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`

```
2736 \def\bbl@push@language{%
```

```
2737 \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.

```
2738 \def\bbl@pop@lang#1+#2-#3{%
```

```
2739 \edef\language{#1}\xdef#3{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before `\bbl@pop@lang` is executed T_EX 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.

```
2740 \let\bbl@ifrestoring\@secondoftwo
```

```
2741 \def\bbl@pop@language{%
```

```
2742 \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
```

```
2743 \let\bbl@ifrestoring\@firstoftwo
```

```
2744 \expandafter\bbl@set@language\expandafter{\language}%
```

```
2745 \let\bbl@ifrestoring\@secondoftwo}
```

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

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

```
2746 \chardef\localeid\z@
```

```
2747 \def\bbl@id@last{0} % No real need for a new counter
```

```
2748 \def\bbl@id@assign{%
```

```
2749 \bbl@ifunset{bbl@id@@\language}%
```

```

2750 {\count@bbl@id@last\relax
2751 \advance\count@\@ne
2752 \bbl@csarg\chardef{id@\language}\count@
2753 \edef\bbl@id@last{\the\count@}%
2754 \ifcase\bbl@engine\or
2755 \directlua{
2756 Babel = Babel or {}
2757 Babel.locale_props = Babel.locale_props or {}
2758 Babel.locale_props[\bbl@id@last] = {}
2759 }%
2760 \fi}%
2761 {}

```

The unprotected part of `\selectlanguage`.

```

2762 \expandafter\def\csname selectlanguage \endcsname#1{%
2763 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@fi
2764 \bbl@push@language
2765 \aftergroup\bbl@pop@language
2766 \bbl@set@language{#1}}

```

`\bbl@set@language` The macro `\bbl@set@language` takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of `\language`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\language` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards. We also write a command to change the current language in the auxiliary files.

```

2767 \def\BabelContentsFiles{toc,lof,lot}
2768 \def\bbl@set@language#1{% from selectlanguage, pop@
2769 \edef\language{%
2770 \ifnum\escapechar=\expandafter`\string#1\@empty
2771 \else\string#1\@empty\fi}%
2772 \select@language{\language}%
2773 % write to aux
2774 \expandafter\ifx\csname date\language\endcsname\relax\else
2775 \if@filesw
2776 \protected@write\@auxout{{}\string\babel@aux{\language}}}%
2777 \bbl@usehooks{write}}}%
2778 \fi
2779 \fi}
2780 \def\select@language#1{% from set@, babel@aux
2781 % set hmap
2782 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2783 % set name
2784 \edef\language{#1}%
2785 \bbl@fixname\language
2786 \bbl@iflanguage\language{%
2787 \expandafter\ifx\csname date\language\endcsname\relax
2788 \bbl@error
2789 {Unknown language `#1'. Either you have\\%
2790 misspelled its name, it has not been installed,\\%
2791 or you requested it in a previous run. Fix its name,\\%
2792 install it or just rerun the file, respectively. In\\%
2793 some cases, you may need to remove the aux file}%
2794 {You may proceed, but expect wrong results}%
2795 \else
2796 % set type

```

```

2797 \let\bbl@select@type\z@
2798 \expandafter\bbl@switch\expandafter{\language}%
2799 \fi}}
2800 \def\babel@aux#1#2{%
2801 \expandafter\ifx\csname date#1\endcsname\relax
2802 \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
2803 \@namedef{bbl@auxwarn@#1}{}%
2804 \bbl@warning
2805 {Unknown language `#1'. Very likely you\\%
2806 requested it in a previous run. Expect some\\%
2807 wrong results in this run, which should vanish\\%
2808 in the next one. Reported}%
2809 \fi
2810 \else
2811 \select@language{#1}%
2812 \bbl@foreach\BabelContentsFiles{%
2813 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
2814 \fi}
2815 \def\babel@toc#1#2{%
2816 \select@language{#1}}

```

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

```

2817 \let\select@language@x\select@language

```

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

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

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

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

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

```

2818 \newif\ifbbl@usedategroup
2819 \def\bbl@switch#1{% from select@, foreign@
2820 % restore
2821 \originalTeX
2822 \expandafter\def\expandafter\originalTeX\expandafter{%
2823 \csname noextras#1\endcsname
2824 \let\originalTeX\@empty
2825 \babel@beginsave}%
2826 \bbl@usehooks{afterreset}{}%
2827 \languageshorthands{none}%
2828 % set the locale id
2829 \bbl@id@assign
2830 \chardef\localeid\@nameuse{bbl@id@\@language}%
2831 % switch captions, date
2832 \ifcase\bbl@select@type
2833 \ifhmode
2834 \hskip\z@skip % trick to ignore spaces
2835 \csname captions#1\endcsname\relax

```

```

2836     \csname date#1\endcsname\relax
2837     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2838     \else
2839         \csname captions#1\endcsname\relax
2840         \csname date#1\endcsname\relax
2841     \fi
2842 \else
2843     \ifbbl@usedategroup % if \foreign... within \<lang>date
2844         \bbl@usedategroupfalse
2845         \ifhmode
2846             \hskip\z@skip % trick to ignore spaces
2847             \csname date#1\endcsname\relax
2848             \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2849         \else
2850             \csname date#1\endcsname\relax
2851         \fi
2852     \fi
2853 \fi
2854 % switch extras
2855 \bbl@usehooks{beforeextras}{}%
2856 \csname extras#1\endcsname\relax
2857 \bbl@usehooks{afterextras}{}%
2858 % > babel-ensure
2859 % > babel-sh-<short>
2860 % > babel-bidi
2861 % > babel-fontspec
2862 % hyphenation - case mapping
2863 \ifcase\bbl@opt@hyphenmap\or
2864     \def\BabelLower##1##2{\lccode##1=##2\relax}%
2865     \ifnum\bbl@hymapsel>4\else
2866         \csname\language @bbl@hyphenmap\endcsname
2867     \fi
2868     \chardef\bbl@opt@hyphenmap\z@
2869 \else
2870     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2871         \csname\language @bbl@hyphenmap\endcsname
2872     \fi
2873 \fi
2874 \global\let\bbl@hymapsel@cclv
2875 % hyphenation - patterns
2876 \bbl@patterns{#1}%
2877 % hyphenation - mins
2878 \babel@savevariable\lefthyphenmin
2879 \babel@savevariable\righthyphenmin
2880 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2881     \set@hyphenmins\tw@\thr@@\relax
2882 \else
2883     \expandafter\expandafter\expandafter\set@hyphenmins
2884     \csname #1hyphenmins\endcsname\relax
2885 \fi}

```

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

```

2886 \long\def\otherlanguage#1{%

```

```

2887 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
2888 \csname selectlanguage \endcsname{#1}%
2889 \ignorespaces}

```

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

```

2890 \long\def\endotherlanguage{%
2891 \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`.

```

2892 \expandafter\def\csname otherlanguage*\endcsname#1{%
2893 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2894 \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”.

```

2895 \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`.

```

2896 \providecommand\bbl@beforeforeign{}
2897 \edef\foreignlanguage{%
2898 \noexpand\protect
2899 \expandafter\noexpand\csname foreignlanguage \endcsname}
2900 \expandafter\def\csname foreignlanguage \endcsname{%
2901 \@ifstar\bbl@foreign@s\bbl@foreign@x}
2902 \def\bbl@foreign@x#1#2{%
2903 \begingroup
2904 \let\BabelText\@firstofone
2905 \bbl@beforeforeign
2906 \foreign@language{#1}%
2907 \bbl@usehooks{foreign}{}%
2908 \BabelText{#2}% Now in horizontal mode!

```



```

2909 \endgroup}
2910 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
2911 \begingroup
2912 {\par}%
2913 \let\BabelText\@firstofone
2914 \foreign@language{#1}%
2915 \bbl@usehooks{foreign*}{}%
2916 \bbl@dirparastext
2917 \BabelText{#2}% Still in vertical mode!
2918 {\par}%
2919 \endgroup}

```

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

```

2920 \def\foreign@language#1{%
2921 % set name
2922 \edef\language#1}%
2923 \bbl@fixname\language
2924 \bbl@iflanguage\language{%
2925 \expandafter\ifx\csname date\language\endcsname\relax
2926 \bbl@warning % TODO - why a warning, not an error?
2927 {Unknown language `#1'. Either you have\\%
2928 misspelled its name, it has not been installed,\\%
2929 or you requested it in a previous run. Fix its name,\\%
2930 install it or just rerun the file, respectively. In\\%
2931 some cases, you may need to remove the aux file.\\%
2932 I'll proceed, but expect wrong results.\\%
2933 Reported}%
2934 \fi
2935 % set type
2936 \let\bbl@select@type\@ne
2937 \expandafter\bbl@switch\expandafter{\language}}

```

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

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

```

2938 \let\bbl@hyphlist\@empty
2939 \let\bbl@hyphenation@\relax
2940 \let\bbl@pttnlist\@empty
2941 \let\bbl@patterns@\relax
2942 \let\bbl@hymapsel=\@cclv
2943 \def\bbl@patterns#1{%
2944 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2945 \csname l@#1\endcsname
2946 \edef\bbl@tempa{#1}%
2947 \else
2948 \csname l@#1:\f@encoding\endcsname
2949 \edef\bbl@tempa{#1:\f@encoding}%
2950 \fi
2951 \@expandtwoargs\bbl@usehooks{patterns}{\bbl@tempa}}%

```

```

2952 % > luatex
2953 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
2954   \begingroup
2955     \bbl@xin@{\number\language,}{\bbl@hyphlist}%
2956     \ifin\else
2957       \@expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
2958     \hyphenation{%
2959       \bbl@hyphenation@
2960       \@ifundefined{bbl@hyphenation@#1}%
2961       \@empty
2962       {\space\csname bbl@hyphenation@#1\endcsname}}%
2963     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2964     \fi
2965   \endgroup}}

```

`hyphenrules` The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change `\language` and when the hyphenation rules specified were not loaded it has no effect. Note however, `\lccode`'s and font encodings are not set at all, so in most cases you should use `other language*`.

```

2966 \def\hyphenrules#1{%
2967   \edef\bbl@tempf{#1}%
2968   \bbl@fixname\bbl@tempf
2969   \bbl@iflanguage\bbl@tempf{%
2970     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2971     \languageshortands{none}%
2972     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2973       \set@hyphenmins\tw@\thr@\relax
2974     \else
2975       \expandafter\expandafter\expandafter\set@hyphenmins
2976       \csname\bbl@tempf hyphenmins\endcsname\relax
2977     \fi}}
2978 \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.

```

2979 \def\providehyphenmins#1#2{%
2980   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2981     \@namedef{#1hyphenmins}{#2}%
2982   \fi}

```

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

```

2983 \def\set@hyphenmins#1#2{%
2984   \lefthyphenmin#1\relax
2985   \righthyphenmin#2\relax}

```

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

```

2986 \ifx\ProvidesFile\@undefined
2987   \def\ProvidesLanguage#1[#2 #3 #4]{%
2988     \wlog{Language: #1 #4 #3 <#2>}%
2989   }
2990 \else
2991   \def\ProvidesLanguage#1{%

```

```

2992 \begingroup
2993 \catcode`\ 10 %
2994 \@makeother\/%
2995 \@ifnextchar[%]
2996   {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2997 \def\@provideslanguage#1[#2]{%
2998   \wlog{Language: #1 #2}%
2999   \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
3000 \endgroup}
3001 \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`

```

3002 \def\LdfInit{%
3003   \chardef\atcatcode=\catcode`\@
3004   \catcode`\@=11\relax
3005   \input babel.def\relax
3006   \catcode`\@=\atcatcode \let\atcatcode\relax
3007   \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`.

```

3008 \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.

```

3009 \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’:

```

3010 \providecommand\setlocale{%
3011   \bbl@error
3012   {Not yet available}%
3013   {Find an armchair, sit down and wait}}
3014 \let\uselocale\setlocale
3015 \let\locale\setlocale
3016 \let\selectlocale\setlocale
3017 \let\textlocale\setlocale
3018 \let\textlanguage\setlocale
3019 \let\languagegetext\setlocale

```

11.2 Errors

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

`\@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'.

```

3020 \edef\bbl@nulllanguage{\string\language=0}
3021 \ifx\PackageError\@undefined
3022   \def\bbl@error#1#2{%
3023     \begingroup
3024       \newlinechar=`^^J
3025       \def\{^^J(babel) }%
3026       \errhelp{#2}\errmessage{\{#1}%
3027     \endgroup}
3028   \def\bbl@warning#1{%
3029     \begingroup
3030       \newlinechar=`^^J
3031       \def\{^^J(babel) }%
3032       \message{\{#1}%
3033     \endgroup}
3034   \def\bbl@info#1{%
3035     \begingroup
3036       \newlinechar=`^^J
3037       \def\{^^J}%
3038       \wlog{#1}%
3039     \endgroup}
3040 \else
3041   \def\bbl@error#1#2{%
3042     \begingroup
3043       \def\{\MessageBreak}%
3044       \PackageError{babel}{#1}{#2}%
3045     \endgroup}
3046   \def\bbl@warning#1{%
3047     \begingroup
3048       \def\{\MessageBreak}%
3049       \PackageWarning{babel}{#1}%
3050     \endgroup}
3051   \def\bbl@info#1{%
3052     \begingroup
3053       \def\{\MessageBreak}%
3054       \PackageInfo{babel}{#1}%
3055     \endgroup}
3056 \fi
3057 \@ifpackagewith{babel}{silent}
3058   {\let\bbl@info@gobble
3059    \let\bbl@warning@gobble}
3060 {}
3061 \def\bbl@nocaption{\protect\bbl@nocaption@i}
3062 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
3063   \global\@namedef{#2}{\textbf{?#1?}}%
3064   \@nameuse{#2}%
3065   \bbl@warning{%
3066     \@backslashchar#2 not set. Please, define\\%
3067     it in the preamble with something like:\\%
3068     \string\renewcommand\@backslashchar#2{..}\\%
3069     Reported}}
3070 \def\bbl@tentative{\protect\bbl@tentative@i}
3071 \def\bbl@tentative@i#1{%
3072   \bbl@warning{%
3073     Some functions for '#1' are tentative.\\%
3074     They might not work as expected and their behavior\\%
3075     could change in the future.\\%

```

```

3076     Reported}}
3077 \def\@nolanerr#1{%
3078   \bbl@error
3079   {You haven't defined the language #1\space yet}%
3080   {Your command will be ignored, type <return> to proceed}}
3081 \def\@nopatterns#1{%
3082   \bbl@warning
3083   {No hyphenation patterns were preloaded for\\%
3084     the language `#1' into the format.\\%
3085     Please, configure your TeX system to add them and\\%
3086     rebuild the format. Now I will use the patterns\\%
3087     preloaded for \bbl@nulllanguage\space instead}}
3088 \let\bbl@usehooks\@gobbbletwo
3089 </kernel>
3090 <*patterns>

```

12 Loading hyphenation patterns

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

We want to add a message to the message $\text{\texttt{L\TeX}}$ 2.09 puts in the `\everyjob` register. This could be done by the following code:

```

\let\orgeveryjob\everyjob
\def\everyjob#1{%
  \orgeveryjob{#1}%
  \orgeveryjob\expandafter{\the\orgeveryjob\immediate\write16{%
    hyphenation patterns for \the\loaded@patterns loaded.}}%
  \let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}

```

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before $\text{\texttt{L\TeX}}$ fills the register.

There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with $\text{\texttt{SL\TeX}}$ the above scheme won't work. The reason is that $\text{\texttt{SL\TeX}}$ overwrites the contents of the `\everyjob` register with its own message.
- Plain $\text{\texttt{TeX}}$ 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 $\text{\texttt{L\TeX}}$ 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

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

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

```

3091 <<Make sure ProvidesFile is defined>>
3092 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
3093 \xdef\bbl@format{\jobname}

```

```

3094 \ifx\AtBeginDocument\@undefined
3095   \def\@empty{}
3096   \let\orig@dump\dump
3097   \def\dump{%
3098     \ifx\@ztryfc\@undefined
3099       \else
3100         \toks0=\expandafter{\@preamblecmds}%
3101         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3102         \def\@begindocumenthook{}%
3103       \fi
3104       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3105 \fi
3106 <<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.

```

3107 \def\process@line#1#2 #3 #4 {%
3108   \ifx=#1%
3109     \process@synonym{#2}%
3110   \else
3111     \process@language{#1#2}{#3}{#4}%
3112   \fi
3113   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bbl@languages` is also set to empty.

```

3114 \toks@{}
3115 \def\bbl@languages{}

```

When no languages have been loaded yet, the name following the `=` will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.)

Otherwise the name will be a synonym for the language loaded last.

We also need to copy the hyphenmin parameters for the synonym.

```

3116 \def\process@synonym#1{%
3117   \ifnum\last@language=\m@ne
3118     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3119   \else
3120     \expandafter\chardef\csname l@#1\endcsname\last@language
3121     \wlog{\string\l@#1=\string\language\the\last@language}%
3122     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3123       \csname\language\hyphenmins\endcsname
3124     \let\bbl@elt\relax
3125     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
3126   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read. For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the

name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

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

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form `\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

3127 \def\process@language#1#2#3{%
3128   \expandafter\addlanguage\csname l@#1\endcsname
3129   \expandafter\language\csname l@#1\endcsname
3130   \edef\language#1#2#3{%
3131     \bbl@hook@everylanguage{#1}%
3132     % > luatex
3133     \bbl@get@enc#1::\@@@
3134     \begin{group}
3135       \lefthyphenmin=\m@ne
3136       \bbl@hook@loadpatterns{#2}%
3137       % > luatex
3138       \ifnum\lefthyphenmin=\m@ne
3139         \else
3140           \expandafter\xdef\csname #1hyphenmins\endcsname{%
3141             \the\lefthyphenmin\the\righthyphenmin}%
3142           \fi
3143     \end{group}
3144     \def\bbl@tempa{#3}%
3145     \ifx\bbl@tempa\@empty\else
3146       \bbl@hook@loadexceptions{#3}%
3147       % > luatex
3148     \fi
3149     \let\bbl@elt\relax
3150     \edef\bbl@languages{%
3151       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3152     \ifnum\the\language=\z@
3153       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3154         \set@hyphenmins\tw@\thr@@\relax
3155       \else
3156         \expandafter\expandafter\expandafter\set@hyphenmins
3157         \csname #1hyphenmins\endcsname
3158       \fi
3159       \the\toks@
3160       \toks@{}%

```

```
3161 \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.

```
3162 \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.

```
3163 \def\bbl@hook@everylanguage#1{}
3164 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3165 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3166 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
3167 \begingroup
3168   \def\AddBabelHook#1#2{%
3169     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3170       \def\next{\toks1}%
3171     \else
3172       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3173     \fi
3174     \next}
3175 \ifx\directlua\undefined
3176   \ifx\XeTeXinputencoding\undefined\else
3177     \input xebabel.def
3178   \fi
3179 \else
3180   \input luababel.def
3181 \fi
3182 \openin1 = babel-\bbl@format.cfg
3183 \ifeof1
3184 \else
3185   \input babel-\bbl@format.cfg\relax
3186 \fi
3187 \closein1
3188 \endgroup
3189 \bbl@hook@loadkernel{switch.def}
```

`\readconfigfile` The configuration file can now be opened for reading.

```
3190 \openin1 = language.dat
```

See if the file exists, if not, use the default hyphenation file `hyphen.tex`. The user will be informed about this.

```
3191 \def\languagename{english}%
3192 \ifeof1
3193   \message{I couldn't find the file language.dat,\space
3194           I will try the file hyphen.tex}
3195   \input hyphen.tex\relax
3196   \chardef\l@english\z@
3197 \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`.

```
3198   \last@language\m@ne
```

We now read lines from the file until the end is found

```
3199   \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.

```
3200 \endlinechar\m@ne
3201 \read1 to \bbl@line
3202 \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.

```
3203 \if T\ifeof1F\fi T\relax
3204 \ifx\bbl@line\@empty\else
3205 \edef\bbl@line{\bbl@line\space\space\space}%
3206 \expandafter\process@line\bbl@line\relax
3207 \fi
3208 \repeat
```

Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns.

```
3209 \begingroup
3210 \def\bbl@elt#1#2#3#4{%
3211 \global\language=#2\relax
3212 \gdef\language#1}%
3213 \def\bbl@elt##1##2##3##4{}}%
3214 \bbl@languages
3215 \endgroup
3216 \fi
```

and close the configuration file.

```
3217 \closein1
```

We add a message about the fact that babel is loaded in the format and with which language patterns to the \everyjob register.

```
3218 \if/\the\toks@/\else
3219 \errhelp{language.dat loads no language, only synonyms}
3220 \errmessage{0rphan language synonym}
3221 \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.

```
3222 \let\bbl@line\@undefined
3223 \let\process@line\@undefined
3224 \let\process@synonym\@undefined
3225 \let\process@language\@undefined
3226 \let\bbl@get@enc\@undefined
3227 \let\bbl@hyph@enc\@undefined
3228 \let\bbl@tempa\@undefined
3229 \let\bbl@hook@loadkernel\@undefined
3230 \let\bbl@hook@everylanguage\@undefined
3231 \let\bbl@hook@loadpatterns\@undefined
3232 \let\bbl@hook@loadexceptions\@undefined
3233 \</patterns>
```

Here the code for `initTeX` ends.

13 Font handling with fontspec

Add the bidi handler just before luaotfload, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
3234 <<{*More package options}>> ≡
3235 \ifodd\bb1@engine
3236   \DeclareOption{bidi=basic-r}%
3237     {\ExecuteOptions{bidi=basic}}
3238   \DeclareOption{bidi=basic}%
3239     {\let\bb1@beforeforeign\leavevmode
3240      % TODO - to locale_props, not as separate attribute
3241      \newattribute\bb1@attr@dir
3242      % I don't like it, hackish:
3243      \frozen@everymath\expandafter{%
3244        \expandafter\bb1@mathboxdir\the\frozen@everymath}%
3245      \frozen@everydisplay\expandafter{%
3246        \expandafter\bb1@mathboxdir\the\frozen@everydisplay}%
3247      \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
3248      \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}%
3249 \else
3250   \DeclareOption{bidi=basic-r}%
3251     {\ExecuteOptions{bidi=basic}}
3252   \DeclareOption{bidi=basic}%
3253     {\bb1@error
3254      {The bidi method `basic' is available only in\\%
3255       luatex. I'll continue with `bidi=default', so\\%
3256       expect wrong results}%
3257      {See the manual for further details.}%
3258      \let\bb1@beforeforeign\leavevmode
3259      \AtEndOfPackage{%
3260        \EnableBabelHook{babel-bidi}%
3261        \bb1@xebidipar}}
3262   \def\bb1@loadxebidi#1{%
3263     \ifx\RTLfootnotetext\@undefined
3264       \AtEndOfPackage{%
3265         \EnableBabelHook{babel-bidi}%
3266         \ifx\fontspec\@undefined
3267           \usepackage{fontspec}% bidi needs fontspec
3268         \fi
3269         \usepackage#1{bidi}}}%
3270     \fi}
3271   \DeclareOption{bidi=bidi}%
3272     {\bb1@tentative{bidi=bidi}%
3273      \bb1@loadxebidi{}}
3274   \DeclareOption{bidi=bidi-r}%
3275     {\bb1@tentative{bidi=bidi-r}%
3276      \bb1@loadxebidi{[rldocument]}}
3277   \DeclareOption{bidi=bidi-l}%
3278     {\bb1@tentative{bidi=bidi-l}%
3279      \bb1@loadxebidi{}}
3280 \fi
3281 \DeclareOption{bidi=default}%
3282   {\let\bb1@beforeforeign\leavevmode
3283    \ifodd\bb1@engine
3284      \newattribute\bb1@attr@dir
3285      \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
3286    \fi
```

```

3287 \AtEndOfPackage{%
3288   \EnableBabelHook{babel-bidi}%
3289   \ifodd\bbbl@engine\else
3290     \bbbl@xebidipar
3291   \fi}}
3292 <</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.

```

3293 <<{*Font selection}>> ≡
3294 \bbbl@trace{Font handling with fontspec}
3295 \@onlypreamble\babelfont
3296 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
3297   \edef\bbbl@tempa{#1}%
3298   \def\bbbl@tempb{#2}%
3299   \ifx\fontspec\undefined
3300     \usepackage{fontspec}%
3301   \fi
3302   \EnableBabelHook{babel-fontspec}% Just calls \bbbl@switchfont
3303   \bbbl@babelfont}
3304 \newcommand\bbbl@babelfont[2][]{% 1=features 2=fontname
3305   \bbbl@ifunset{\bbbl@tempb family}%
3306     {\bbbl@providfam{\bbbl@tempb}}%
3307     {\bbbl@exp{%
3308       \\\bbbl@sreplace\<\bbbl@tempb family >%
3309       {\@nameuse{\bbbl@tempb default}}{\<\bbbl@tempb default>}}}%
3310   % For the default font, just in case:
3311   \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\@languagename}}{}%
3312   \expandafter\bbbl@ifblank\expandafter{\bbbl@tempa}%
3313     {\bbbl@csarg\edef{\bbbl@tempb dflt@}{<>{#1}{#2}}% save \bbbl@rmdflt@
3314     \bbbl@exp{%
3315       \let\<\bbbl@\bbbl@tempb dflt@\@languagename>\<\bbbl@\bbbl@tempb dflt@>%
3316       \\\bbbl@font@set\<\bbbl@\bbbl@tempb dflt@\@languagename>%
3317       \<\bbbl@tempb default>\<\bbbl@tempb family>}}%
3318     {\bbbl@foreach\bbbl@tempa{% ie \bbbl@rmdflt@lang / *scrt
3319       \bbbl@csarg\def{\bbbl@tempb dflt@##1}{<>{#1}{#2}}}}}%

```

If the family in the previous command does not exist, it must be defined. Here is how:

```

3320 \def\bbbl@providfam#1{%
3321   \bbbl@exp{%
3322     \\\newcommand\<#1default>{}% Just define it
3323     \\\bbbl@add@list\bbbl@font@fams{#1}%
3324     \\\DeclareRobustCommand\<#1family>{%
3325       \\\not@math@alphabet\<#1family>\relax
3326       \\\fontfamily\<#1default>\selectfont}%
3327     \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}%

```

The following macro is activated when the hook babel-fontspec is enabled.

```

3328 \def\bbbl@switchfont{%
3329   \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\@languagename}}{}%
3330   \bbbl@exp{% eg Arabic -> arabic
3331     \lowercase{\edef\bbbl@tempa{\bbbl@cs{sname@\@languagename}}}%
3332     \bbbl@foreach\bbbl@font@fams{%
3333       \bbbl@ifunset{\bbbl@##1dflt@\@languagename}% (1) language?
3334       {\bbbl@ifunset{\bbbl@##1dflt@*\bbbl@tempa}% (2) from script?
3335         {\bbbl@ifunset{\bbbl@##1dflt@}% 2=F - (3) from generic?
3336           {}% 123=F - nothing!
3337           {\bbbl@exp{% 3=T - from generic
3338             \global\let\<\bbbl@##1dflt@\@languagename>%

```

```

3339 \<bbl@##1dflt@>}}}%
3340 {\bbl@exp{% 2=T - from script
3341 \global\let\<bbl@##1dflt@\language>%
3342 \<bbl@##1dflt@*\bbl@tempa>}}}%
3343 {}}}% 1=T - language, already defined
3344 \def\bbl@tempa{%
3345 \bbl@warning{The current font is not a babel standard family:\%
3346 \fontname\font\%
3347 There is nothing intrinsically wrong with it, but\%
3348 script and Language will not be applied. Consider\%
3349 defining a new family with \string\babelfont.\%
3350 Reported}}}%
3351 \bbl@foreach\bbl@font@fams{% don't gather with prev for
3352 \bbl@ifunset\bbl@##1dflt@\language}%
3353 {\bbl@cs{famrst@##1}%
3354 \global\bbl@csarg\let{famrst@##1}\relax}%
3355 {\bbl@exp{% order is relevant
3356 \\\bbl@add\\originalTeX{%
3357 \\\bbl@font@rst{\bbl@cs{##1dflt@\language}}}%
3358 \<##1default>\<##1family>{##1}}}%
3359 \\\bbl@font@set\<bbl@##1dflt@\language>% the main part!
3360 \<##1default>\<##1family>}}}%
3361 \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.

```

3362 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
3363 \bbl@xin@{<>}{#1}%
3364 \ifin@
3365 \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
3366 \fi
3367 \bbl@exp{%
3368 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
3369 \\\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\bbl@tempa\relax}}}%
3370 % TODO - next should be global?, but even local does its job. I'm
3371 % still not sure -- must investigate:
3372 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
3373 \let\bbl@tempa\bbl@mapselect
3374 \let\bbl@mapselect\relax
3375 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
3376 \let#4\relax % So that can be used with \newfontfamily
3377 \bbl@exp{%
3378 \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
3379 \<keys_if_exist:nnF>{fontspec-opentype}%
3380 {Script/\bbl@cs{sname@\language}}}%
3381 {\\\newfontscript{\bbl@cs{sname@\language}}}%
3382 {\bbl@cs{sotf@\language}}}%
3383 \<keys_if_exist:nnF>{fontspec-opentype}%
3384 {Language/\bbl@cs{lname@\language}}}%
3385 {\\\newfontlanguage{\bbl@cs{lname@\language}}}%
3386 {\bbl@cs{lotf@\language}}}%
3387 \\\newfontfamily\\#4%
3388 [\bbl@cs{lsys@\language},#2]{#3}% ie \bbl@exp{.}{#3}
3389 \begingroup
3390 #4%
3391 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}

```

```

3392 \endgroup
3393 \let#4\bb1@temp@fam
3394 \bb1@exp{\let\<\bb1@stripslash#4\space>}\bb1@temp@pfam
3395 \let\bb1@mapselect\bb1@tempe}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```

3396 \def\bb1@font@rst#1#2#3#4{%
3397 \bb1@csarg\def{famrst@#4}{\bb1@font@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```

3398 \def\bb1@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 :-).

```

3399 \newcommand\babelFSstore[2][{}]{%
3400 \bb1@ifblank{#1}%
3401 {\bb1@csarg\def{sname@#2}{Latin}}%
3402 {\bb1@csarg\def{sname@#2}{#1}}%
3403 \bb1@provide@dirs{#2}%
3404 \bb1@csarg\ifnum{wdir@#2}>\z@
3405 \let\bb1@beforeforeign\leavevmode
3406 \EnableBabelHook{babel-bidi}%
3407 \fi
3408 \bb1@foreach{#2}{%
3409 \bb1@FSstore{##1}{rm}\rmdefault\bb1@save@rmdefault
3410 \bb1@FSstore{##1}{sf}\sfdefault\bb1@save@sfdefault
3411 \bb1@FSstore{##1}{tt}\ttdefault\bb1@save@ttdefault}}
3412 \def\bb1@FSstore#1#2#3#4{%
3413 \bb1@csarg\edef{#2default#1}{#3}%
3414 \expandafter\addto\csname extras#1\endcsname{%
3415 \let#4#3%
3416 \ifx#3\f@family
3417 \edef#3{\csname bbl@#2default#1\endcsname}%
3418 \fontfamily{#3}\selectfont
3419 \else
3420 \edef#3{\csname bbl@#2default#1\endcsname}%
3421 \fi}%
3422 \expandafter\addto\csname noextras#1\endcsname{%
3423 \ifx#3\f@family
3424 \fontfamily{#4}\selectfont
3425 \fi
3426 \let#3#4}}
3427 \let\bb1@langfeatures\@empty
3428 \def\babelFSfeatures{% make sure \fontspec is redefined once
3429 \let\bb1@ori@fontspec\fontspec
3430 \renewcommand\fontspec[1][{}]{%
3431 \bb1@ori@fontspec[\bb1@langfeatures##1]}
3432 \let\babelFSfeatures\bb1@FSfeatures
3433 \babelFSfeatures}
3434 \def\bb1@FSfeatures#1#2{%
3435 \expandafter\addto\csname extras#1\endcsname{%
3436 \babel@save\bb1@langfeatures
3437 \edef\bb1@langfeatures{#2,}}
3438 <</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.

`ℒTeX` sets many “codes” just before loading `hyphen.cfg`. That is not a problem in `luaTeX`, but in `xetTeX` 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 `ℒTeX`. Anyway, for consistency `LuaTeX` also resets the catcodes.

```
3439 <<*Restore Unicode catcodes before loading patterns>> ≡
3440 \begingroup
3441 % Reset chars "80-"C0 to category "other", no case mapping:
3442 \catcode\@=11 \count@=128
3443 \loop\ifnum\count@<192
3444   \global\uccode\count@=0 \global\lccode\count@=0
3445   \global\catcode\count@=12 \global\sffcode\count@=1000
3446   \advance\count@ by 1 \repeat
3447 % Other:
3448 \def\O ##1 {%
3449   \global\uccode"##1=0 \global\lccode"##1=0
3450   \global\catcode"##1=12 \global\sffcode"##1=1000 }%
3451 % Letter:
3452 \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3453   \global\uccode"##1="##2
3454   \global\lccode"##1="##3
3455   % Uppercase letters have sffcode=999:
3456   \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
3457 % Letter without case mappings:
3458 \def\l ##1 {\L ##1 ##1 ##1 }%
3459 \l 00AA
3460 \L 00B5 039C 00B5
3461 \l 00BA
3462 \O 00D7
3463 \l 00DF
3464 \O 00F7
3465 \L 00FF 0178 00FF
3466 \endgroup
3467 \input #1\relax
3468 <</Restore Unicode catcodes before loading patterns>>
```

Some more common code.

```
3469 <<*Footnote changes>> ≡
3470 \bbl@trace{Bidi footnotes}
3471 \ifx\bbl@beforeforeign\leavevmode
3472   \def\bbl@footnote#1#2#3{%
3473     \@ifnextchar[%
3474       {\bbl@footnote@o{#1}{#2}{#3}}%
3475       {\bbl@footnote@x{#1}{#2}{#3}}}
3476   \def\bbl@footnote@x#1#2#3#4{%
3477     \bgroup
3478     \select@language@x{\bbl@main@language}%
3479     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3480     \egroup}
3481   \def\bbl@footnote@o#1#2#3[#4]#5{%
3482     \bgroup
3483     \select@language@x{\bbl@main@language}%
3484     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3485     \egroup}
```

```

3485 \egroup}
3486 \def\bbl@footnotetext#1#2#3{%
3487 \@ifnextchar[%
3488 {\bbl@footnotetext@o{#1}{#2}{#3}}%
3489 {\bbl@footnotetext@x{#1}{#2}{#3}}}
3490 \def\bbl@footnotetext@x#1#2#3#4{%
3491 \bgroup
3492 \select@language@x{\bbl@main@language}%
3493 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3494 \egroup}
3495 \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3496 \bgroup
3497 \select@language@x{\bbl@main@language}%
3498 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3499 \egroup}
3500 \def\BabelFootnote#1#2#3#4{%
3501 \ifx\bbl@fn@footnote\@undefined
3502 \let\bbl@fn@footnote\footnote
3503 \fi
3504 \ifx\bbl@fn@footnotetext\@undefined
3505 \let\bbl@fn@footnotetext\footnotetext
3506 \fi
3507 \bbl@ifblank{#2}%
3508 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3509 \@namedef{\bbl@stripslash#1text}%
3510 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3511 {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
3512 \@namedef{\bbl@stripslash#1text}%
3513 {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
3514 \fi
3515 <</Footnote changes>>

```

Now, the code.

```

3516 (*xetex)
3517 \def\BabelStringsDefault{unicode}
3518 \let\xebbl@stop\relax
3519 \AddBabelHook{xetex}{encodedcommands}{%
3520 \def\bbl@tempa{#1}%
3521 \ifx\bbl@tempa\@empty
3522 \XeTeXinputencoding"bytes"%
3523 \else
3524 \XeTeXinputencoding"#1"%
3525 \fi
3526 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3527 \AddBabelHook{xetex}{stopcommands}{%
3528 \xebbl@stop
3529 \let\xebbl@stop\relax}
3530 \def\bbl@intraspace#1 #2 #3\@@{%
3531 \bbl@csarg\gdef{\xeisp@bbl@cs{\sbcp@languagename}}%
3532 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3533 \def\bbl@intrapenalty#1\@@{%
3534 \bbl@csarg\gdef{\xeipn@bbl@cs{\sbcp@languagename}}%
3535 {\XeTeXlinebreakpenalty #1\relax}}
3536 \AddBabelHook{xetex}{loadkernel}{%
3537 <<Restore Unicode catcodes before loading patterns>>
3538 \ifx\DisableBabelHook\@undefined\endinput\fi
3539 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3540 \DisableBabelHook{babel-fontspec}
3541 <<Font selection>>

```

```

3542 \input txtbabel.def
3543 \xetex

```

14.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the 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.

```

3544 (*texxet)
3545 \bbl@trace{Redefinitions for bidi layout}
3546 \def\bbl@sspre@caption{%
3547   \bbl@exp{\everybox{\bbl@textdir\bbl@cs{wdir\bbl@main@language}}}}
3548 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
3549 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3550 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3551 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3552   \def\@hangfrom#1{%
3553     \setbox\@tempboxa\hbox{#1}%
3554     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3555     \noindent\box\@tempboxa}
3556 \def\raggedright{%
3557   \let\@centercr
3558   \bbl@startskip\z@skip
3559   \@rightskip\@flushglue
3560   \bbl@endskip\@rightskip
3561   \parindent\z@
3562   \parfillskip\bbl@startskip}
3563 \def\raggedleft{%
3564   \let\@centercr
3565   \bbl@startskip\@flushglue
3566   \bbl@endskip\z@skip
3567   \parindent\z@
3568   \parfillskip\bbl@endskip}
3569 \fi
3570 \IfBabelLayout{lists}
3571   {\bbl@sreplace\list
3572     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
3573     \def\bbl@listleftmargin{%
3574       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
3575     \ifcase\bbl@engine
3576       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
3577       \def\p@enumiii{\p@enumii}\theenumii{}\fi
3578     \bbl@sreplace\@verbatim
3579       {\leftskip\@totalleftmargin}%
3580       {\bbl@startskip\textwidth
3581         \advance\bbl@startskip-\linewidth}%
3582     \bbl@sreplace\@verbatim
3583       {\rightskip\z@skip}%
3584       {\bbl@endskip\z@skip}}%
3586   {}
3587 \IfBabelLayout{contents}

```



```

3588 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
3589 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
3590 {}
3591 \IfBabelLayout{columns}
3592 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
3593 \def\bbl@outputbox#1{%
3594 \hb@xt@\textwidth{%
3595 \hskip\columnwidth
3596 \hfil
3597 {\normalcolor\vrule \@width\columnseprule}%
3598 \hfil
3599 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3600 \hskip-\textwidth
3601 \hb@xt@\columnwidth{\box\@outputbox \hss}%
3602 \hskip\columnsep
3603 \hskip\columnwidth}}}%
3604 {}
3605 <<Footnote changes>>
3606 \IfBabelLayout{footnotes}%
3607 {\BabelFootnote\footnote\language\language{}{}}%
3608 \BabelFootnote\localfootnote\language\language{}{}}%
3609 \BabelFootnote\mainfootnote{}{}{}%
3610 {}

```

Implicitly reverses sectioning labels in bidi=basic, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

3611 \IfBabelLayout{counters}%
3612 {\let\bbl@latinarabic=\@arabic
3613 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3614 \let\bbl@asciroman=\@roman
3615 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3616 \let\bbl@asciiRoman=\@Roman
3617 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
3618 </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.

```

3619 (*luatex)
3620 \ifx\AddBabelHook\@undefined
3621 \bbl@trace{Read language.dat}
3622 \begingroup
3623 \toks@{}
3624 \count@ \z@ % 0=start, 1=0th, 2=normal
3625 \def\bbl@process@line#1#2 #3 #4 {%
3626   \ifx=#1%
3627     \bbl@process@synonym{#2}%
3628   \else
3629     \bbl@process@language{#1#2}{#3}{#4}%
3630   \fi
3631   \ignorespaces}
3632 \def\bbl@manylang{%
3633   \ifnum\bbl@last>\@ne
3634     \bbl@info{Non-standard hyphenation setup}%
3635   \fi
3636   \let\bbl@manylang\relax}
3637 \def\bbl@process@language#1#2#3{%
3638   \ifcase\count@
3639     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
3640   \or
3641     \count@\tw@
3642   \fi
3643   \ifnum\count@=\tw@
3644     \expandafter\addlanguage\csname l@#1\endcsname
3645     \language\allocationnumber
3646     \chardef\bbl@last\allocationnumber
3647     \bbl@manylang
3648     \let\bbl@elt\relax
3649     \xdef\bbl@languages{%
3650       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3651   \fi
3652   \the\toks@
3653   \toks@{}}
3654 \def\bbl@process@synonym@aux#1#2{%
3655   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3656   \let\bbl@elt\relax
3657   \xdef\bbl@languages{%
3658     \bbl@languages\bbl@elt{#1}{#2}{}}}%
3659 \def\bbl@process@synonym#1{%
3660   \ifcase\count@
3661     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3662   \or
3663     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3664   \else

```

```

3665     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3666     \fi}
3667 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
3668     \chardef\l@english\z@
3669     \chardef\l@USenglish\z@
3670     \chardef\bbl@last\z@
3671     \global\@namedef{bbl@hyphendata@0}{\hyphen.tex{}}
3672     \gdef\bbl@languages{%
3673         \bbl@elt{english}{0}{\hyphen.tex}{}}%
3674     \bbl@elt{USenglish}{0}{}}{}
3675 \else
3676     \global\let\bbl@languages@format\bbl@languages
3677     \def\bbl@elt#1#2#3#4{% Remove all except language 0
3678         \ifnum#2>\z@\else
3679             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3680             \fi}%
3681     \xdef\bbl@languages{\bbl@languages}%
3682 \fi
3683 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3684 \bbl@languages
3685 \openin1=language.dat
3686 \ifeof1
3687     \bbl@warning{I couldn't find language.dat. No additional\\%
3688         patterns loaded. Reported}%
3689 \else
3690     \loop
3691         \endlinechar\m@ne
3692         \read1 to \bbl@line
3693         \endlinechar\^^M
3694         \if T\ifeof1F\fi T\relax
3695         \ifx\bbl@line\@empty\else
3696             \edef\bbl@line{\bbl@line\space\space\space}%
3697             \expandafter\bbl@process@line\bbl@line\relax
3698         \fi
3699     \repeat
3700 \fi
3701 \endgroup
3702 \bbl@trace{Macros for reading patterns files}
3703 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
3704 \ifx\babelcatcodetablenum\@undefined
3705     \def\babelcatcodetablenum{5211}
3706 \fi
3707 \def\bbl@luapatterns#1#2{%
3708     \bbl@get@enc#1::\@@@
3709     \setbox\z@\hbox\bgroup
3710     \begingroup
3711         \ifx\catcodetable\@undefined
3712             \let\savecatcodetable\luatexsavecatcodetable
3713             \let\initcatcodetable\luatexinitcatcodetable
3714             \let\catcodetable\luatexcatcodetable
3715         \fi
3716         \savecatcodetable\babelcatcodetablenum\relax
3717         \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3718         \catcodetable\numexpr\babelcatcodetablenum+1\relax
3719         \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3720         \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\-=13
3721         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3722         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3723         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12

```

```

3724 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
3725 \input #1\relax
3726 \catcodetable\babelcatcodetablenum\relax
3727 \endgroup
3728 \def\bbl@tempa{#2}%
3729 \ifx\bbl@tempa\@empty\else
3730 \input #2\relax
3731 \fi
3732 \egroup}%
3733 \def\bbl@patterns@lua#1{%
3734 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3735 \csname l@#1\endcsname
3736 \edef\bbl@tempa{#1}%
3737 \else
3738 \csname l@#1:\f@encoding\endcsname
3739 \edef\bbl@tempa{#1:\f@encoding}%
3740 \fi\relax
3741 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3742 \@ifundefined{bbl@hyphendata@the\language}%
3743 {\def\bbl@elt##1##2##3##4{%
3744 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3745 \def\bbl@tempb{##3}%
3746 \ifx\bbl@tempb\@empty\else % if not a synonymous
3747 \def\bbl@tempc{##3}##4}%
3748 \fi
3749 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3750 \fi}%
3751 \bbl@languages
3752 \@ifundefined{bbl@hyphendata@the\language}%
3753 {\bbl@info{No hyphenation patterns were set for\%
3754 language '\bbl@tempa'. Reported}}%
3755 {\expandafter\expandafter\expandafter\bbl@luapatterns
3756 \csname bbl@hyphendata@the\language\endcsname}}}%
3757 \endinput\fi
3758 \begingroup
3759 \catcode`\%=12
3760 \catcode`\'=12
3761 \catcode`\`=12
3762 \catcode`\:=12
3763 \directlua{
3764 Babel = Babel or {}
3765 function Babel.bytes(line)
3766 return line:gsub(".",
3767 function (chr) return unicode.utf8.char(string.byte(chr)) end)
3768 end
3769 function Babel.begin_process_input()
3770 if luatexbase and luatexbase.add_to_callback then
3771 luatexbase.add_to_callback('process_input_buffer',
3772 Babel.bytes, 'Babel.bytes')
3773 else
3774 Babel.callback = callback.find('process_input_buffer')
3775 callback.register('process_input_buffer', Babel.bytes)
3776 end
3777 end
3778 function Babel.end_process_input ()
3779 if luatexbase and luatexbase.remove_from_callback then
3780 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3781 else
3782 callback.register('process_input_buffer', Babel.callback)

```

```

3783     end
3784 end
3785 function Babel.addpatterns(pp, lg)
3786     local lg = lang.new(lg)
3787     local pats = lang.patterns(lg) or ''
3788     lang.clear_patterns(lg)
3789     for p in pp:gmatch('[^%s]+') do
3790         ss = ''
3791         for i in string.utfcharacters(p:gsub('%d', '')) do
3792             ss = ss .. '%d?' .. i
3793         end
3794         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
3795         ss = ss:gsub('%.%d%?$', '%%.')
3796         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3797         if n == 0 then
3798             tex.sprint(
3799                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
3800                 .. p .. [[]])
3801             pats = pats .. ' ' .. p
3802         else
3803             tex.sprint(
3804                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
3805                 .. p .. [[]])
3806         end
3807     end
3808     lang.patterns(lg, pats)
3809 end
3810 }
3811 \endgroup
3812 \ifx\newattribute\@undefined\else
3813     \newattribute\bbl@attr@locale
3814     \AddBabelHook{luatex}{beforeextras}{%
3815         \setattribute\bbl@attr@locale\localeid}
3816 \fi
3817 \def\BabelStringsDefault{unicode}
3818 \let\luabbl@stop\relax
3819 \AddBabelHook{luatex}{encodedcommands}{%
3820     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3821     \ifx\bbl@tempa\bbl@tempb\else
3822         \directlua{Babel.begin_process_input()}%
3823     \def\luabbl@stop{%
3824         \directlua{Babel.end_process_input()}}%
3825 \fi}%
3826 \AddBabelHook{luatex}{stopcommands}{%
3827     \luabbl@stop
3828     \let\luabbl@stop\relax}
3829 \AddBabelHook{luatex}{patterns}{%
3830     \@ifundefined{bbl@hyphendata@the\language}%
3831     {\def\bbl@elt##1##2##3##4{%
3832         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3833         \def\bbl@tempb{##3}%
3834         \ifx\bbl@tempb@empty\else % if not a synonymous
3835             \def\bbl@tempc{##3}{##4}}%
3836         \fi
3837         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3838         \fi}%
3839     \bbl@languages
3840     \@ifundefined{bbl@hyphendata@the\language}%
3841     {\bbl@info{No hyphenation patterns were set for\%

```

```

3842         language '#2'. Reported}}%
3843     {\expandafter\expandafter\expandafter\bbbl@luapatterns
3844       \csname bbl@hyphendata@the\language\endcsname}}}%
3845 \ifundefined{bbl@patterns@}{}%
3846   \begingroup
3847     \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
3848     \ifin@else
3849       \ifx\bbbl@patterns@\empty\else
3850         \directlua{ Babel.addpatterns(
3851           [[\bbbl@patterns@]], \number\language) }%
3852         \fi
3853       \ifundefined{bbl@patterns@#1}%
3854         \empty
3855       {\directlua{ Babel.addpatterns(
3856         [[\space\csname bbl@patterns@#1\endcsname]],
3857         \number\language) }}%
3858       \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
3859     \fi
3860   \endgroup}}
3861 \AddBabelHook{luatex}{everylanguage}{%
3862   \def\process@language##1##2##3{%
3863     \def\process@line####1####2 ####3 ####4 {}}
3864 \AddBabelHook{luatex}{loadpatterns}{%
3865   \input #1\relax
3866   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3867     {{#1}}}%
3868 \AddBabelHook{luatex}{loadexceptions}{%
3869   \input #1\relax
3870   \def\bbbl@tempb##1##2{{##1}{#1}}%
3871   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3872     {\expandafter\expandafter\expandafter\bbbl@tempb
3873       \csname bbl@hyphendata@the\language\endcsname}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbbl@patterns@` for the global ones and `\bbbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

3874 \@onlypreamble\babelpatterns
3875 \AtEndOfPackage{%
3876   \newcommand\babelpatterns[2][\empty]{%
3877     \ifx\bbbl@patterns@relax
3878       \let\bbbl@patterns@\empty
3879     \fi
3880     \ifx\bbbl@pttnlist\empty\else
3881       \bbbl@warning{%
3882         You must not intermingle \string\selectlanguage\space and\%
3883         \string\babelpatterns\space or some patterns will not\%
3884         be taken into account. Reported}%
3885       \fi
3886       \ifx\empty#1%
3887         \protected@edef\bbbl@patterns@{\bbbl@patterns@\space#2}%
3888       \else
3889         \edef\bbbl@tempb{\zap@space#1 \empty}%
3890         \bbbl@for\bbbl@tempa\bbbl@tempb{%
3891           \bbbl@fixname\bbbl@tempa
3892           \bbbl@iflanguage\bbbl@tempa{%
3893             \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
3894               \ifundefined{bbl@patterns@\bbbl@tempa}%
3895                 \empty

```

```

3896         {\csname bbl@patterns@bbl@tempa\endcsname\space}%
3897         #2}}}%
3898     \fi}}

```

14.4 Southeast Asian scripts

In progress. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

3899 \def\bbl@intraspace#1 #2 #3\@@{%
3900     \directlua{
3901         Babel = Babel or {}
3902         Babel.intraspaces = Babel.intraspaces or {}
3903         Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
3904             {b = #1, p = #2, m = #3}
3905         Babel.locale_props[\the\localeid].intraspace = %
3906             {b = #1, p = #2, m = #3}
3907     }}
3908 \def\bbl@intrapenalty#1\@@{%
3909     \directlua{
3910         Babel = Babel or {}
3911         Babel.intrapenalties = Babel.intrapenalties or {}
3912         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
3913         Babel.locale_props[\the\localeid].intrapenalty = #1
3914     }}
3915 \begingroup
3916 \catcode`\%=12
3917 \catcode`\^=14
3918 \catcode`\'=12
3919 \catcode`\~=12
3920 \gdef\bbl@seaintraspace{^
3921     \let\bbl@seaintraspace\relax
3922     \directlua{
3923         Babel = Babel or {}
3924         Babel.sea_enabled = true
3925         Babel.sea_ranges = Babel.sea_ranges or {}
3926         function Babel.set_chranges (script, chrng)
3927             local c = 0
3928             for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
3929                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
3930                 c = c + 1
3931             end
3932         end
3933         function Babel.sea_disc_to_space (head)
3934             local sea_ranges = Babel.sea_ranges
3935             local last_char = nil
3936             local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
3937             for item in node.traverse(head) do
3938                 local i = item.id
3939                 if i == node.id'glyph' then
3940                     last_char = item
3941                 elseif i == 7 and item.subtype == 3 and last_char
3942                     and last_char.char > 0x0C99 then
3943                     quad = font.getfont(last_char.font).size
3944                     for lg, rg in pairs(sea_ranges) do
3945                         if last_char.char > rg[1] and last_char.char < rg[2] then
3946                             lg = lg:sub(1, 4)

```

```

3947         local intraspace = Babel.intraspaces[lg]
3948         local intrapenalty = Babel.intrapanalties[lg]
3949         local n
3950         if intrapenalty ~= 0 then
3951             n = node.new(14, 0)    ^^ penalty
3952             n.penalty = intrapenalty
3953             node.insert_before(head, item, n)
3954         end
3955         n = node.new(12, 13)    ^^ (glue, spaceskip)
3956         node.setglue(n, intraspace.b * quad,
3957                     intraspace.p * quad,
3958                     intraspace.m * quad)
3959         node.insert_before(head, item, n)
3960         node.remove(head, item)
3961     end
3962 end
3963 end
3964 end
3965 end
3966 }^^
3967 \bbl@luahyphenate}
3968 \catcode`\%=14
3969 \gdef\bbl@cjkintraspaces{%
3970   \let\bbl@cjkintraspaces\relax
3971   \directlua{
3972     Babel = Babel or {}
3973     require'babel-data-cjk.lua'
3974     Babel.cjk_enabled = true
3975     function Babel.cjk_linebreak(head)
3976       local GLYPH = node.id'glyph'
3977       local last_char = nil
3978       local quad = 655360      % 10 pt = 655360 = 10 * 65536
3979       local last_class = nil
3980       local last_lang = nil
3981
3982       for item in node.traverse(head) do
3983         if item.id == GLYPH then
3984
3985           local lang = item.lang
3986
3987           local LOCALE = node.get_attribute(item,
3988             luatexbase.registernumber'bbl@attr@locale')
3989           local props = Babel.locale_props[LOCALE]
3990
3991           class = Babel.cjk_class[item.char].c
3992
3993           if class == 'cp' then class = 'cl' end % )] as CL
3994           if class == 'id' then class = 'I' end
3995
3996           if class and last_class and Babel.cjk_breaks[last_class][class] then
3997             br = Babel.cjk_breaks[last_class][class]
3998           else
3999             br = 0
4000           end
4001
4002           if br == 1 and props.linebreak == 'c' and
4003             lang ~= \the\l@nohyphenation\space and
4004             last_lang ~= \the\l@nohyphenation then
4005             local intrapenalty = props.intrapenalty

```



```

4006         if intrapenalty ~= 0 then
4007             local n = node.new(14, 0)      % penalty
4008             n.penalty = intrapenalty
4009             node.insert_before(head, item, n)
4010         end
4011         local intraspace = props.intraspace
4012         local n = node.new(12, 13)         % (glue, spaceskip)
4013         node.setglue(n, intraspace.b * quad,
4014                     intraspace.p * quad,
4015                     intraspace.m * quad)
4016         node.insert_before(head, item, n)
4017     end
4018
4019     quad = font.getfont(item.font).size
4020     last_class = class
4021     last_lang = lang
4022     else % if penalty, glue or anything else
4023         last_class = nil
4024     end
4025 end
4026 lang.hyphenate(head)
4027 end
4028 }%
4029 \bbl@luahyphenate}
4030 \gdef\bbl@luahyphenate{%
4031 \let\bbl@luahyphenate\relax
4032 \directlua{
4033     luatexbase.add_to_callback('hyphenate',
4034     function (head, tail)
4035         if Babel.cjk_enabled then
4036             Babel.cjk_linebreak(head)
4037         end
4038         lang.hyphenate(head)
4039         if Babel.sea_enabled then
4040             Babel.sea_disc_to_space(head)
4041         end
4042     end,
4043     'Babel.hyphenate')
4044 }
4045 }
4046 \endgroup

```

14.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

Work in progress.

Common stuff.

```

4047 \AddBabelHook{luatex}{loadkernel}{%
4048 <<Restore Unicode catcodes before loading patterns>>}
4049 \ifx\DisableBabelHook\@undefined\endinput\fi
4050 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}

```

```

4051 \DisableBabelHook{babel-fontspec}
4052 <<Font selection>>

```

Temporary fix for luatex <1.10, which sometimes inserted a spurious closing dir node with a \texdir within \hboxes. This will be eventually removed.

```

4053 \def\bbl@luafixboxdir{%
4054   \setbox\z@\hbox{\texdir TLT}%
4055   \directlua{
4056     function Babel.first_dir(head)
4057       for item in node.traverse_id(node.id'dir', head) do
4058         return item
4059       end
4060       return nil
4061     end
4062     if Babel.first_dir(tex.box[0].head) then
4063       function Babel.fixboxdirs(head)
4064         local fd = Babel.first_dir(head)
4065         if fd and fd.dir:sub(1,1) == '-' then
4066           head = node.remove(head, fd)
4067         end
4068         return head
4069       end
4070     end
4071   }}
4072 \AtBeginDocument{\bbl@luafixboxdir}

```

The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.

```

4073 \newcommand\babelcharproperty[1]{%
4074   \count@=#1\relax
4075   \ifvmode
4076     \expandafter\bbl@chprop
4077   \else
4078     \bbl@error{\string\babelcharproperty\space can be used only in\\%
4079       vertical mode (preamble or between paragraphs)}%
4080     {See the manual for futher info}%
4081   \fi}
4082 \newcommand\bbl@chprop[3][\the\count@]{%
4083   \@tempcnta=#1\relax
4084   \bbl@ifunset{\bbl@chprop@#2}%
4085   {\bbl@error{No property named '#2'. Allowed values are\\%
4086     direction (bc), mirror (bmg), and linebreak (lb)}%
4087     {See the manual for futher info}}%
4088   }%
4089   \loop
4090     \@nameuse{\bbl@chprop@#2}{#3}%
4091   \ifnum\count@<\@tempcnta
4092     \advance\count@\@ne
4093   \repeat}
4094 \def\bbl@chprop@direction#1{%
4095   \directlua{
4096     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4097     Babel.characters[\the\count@]['d'] = '#1'
4098   }}
4099 \let\bbl@chprop@bc\bbl@chprop@direction
4100 \def\bbl@chprop@mirror#1{%
4101   \directlua{
4102     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4103     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

4104 }}
4105 \let\bbl@chprop@bmg\bbl@chprop@mirror
4106 \def\bbl@chprop@linebreak#1{%
4107   \directlua{
4108     Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4109     Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4110   }}
4111 \let\bbl@chprop@lb\bbl@chprop@linebreak

```

14.6 Layout

Work in progress.

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

4112 \bbl@trace{Redefinitions for bidi layout}
4113 \ifx\@eqnnum\@undefined\else
4114   \ifx\bbl@attr@dir\@undefined\else
4115     \edef\@eqnnum{%
4116       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
4117       \unexpanded\expandafter{\@eqnnum}}}%
4118   \fi
4119 \fi
4120 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
4121 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4122   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
4123     \bbl@exp{%
4124       \mathdir\the\bodydir
4125       #1%           Once entered in math, set boxes to restore values
4126       \<ifmmode>%
4127       \everyvbox{%
4128         \the\everyvbox
4129         \bodydir\the\bodydir
4130         \mathdir\the\mathdir
4131         \everyhbox{\the\everyhbox}%
4132         \everyvbox{\the\everyvbox}}%
4133       \everyhbox{%
4134         \the\everyhbox
4135         \bodydir\the\bodydir
4136         \mathdir\the\mathdir
4137         \everyhbox{\the\everyhbox}%
4138         \everyvbox{\the\everyvbox}}%
4139       \<fi>}}%
4140   \def\@hangfrom#1{%
4141     \setbox\@tempboxa\hbox{{#1}}%
4142     \hangindent\wd\@tempboxa
4143     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4144       \shapemode\@ne

```

```

4145 \fi
4146 \noindent\box\@tempboxa}
4147 \fi
4148 \IfBabelLayout{tabular}
4149 {\bbl@replace\@tabular{$}\bbl@nextfake$}%
4150 \let\bbl@tabular\@tabular
4151 \AtBeginDocument{%
4152 \ifx\bbl@tabular\@tabular\else
4153 \bbl@replace\@tabular{$}\bbl@nextfake$}%
4154 \fi}}
4155 {}
4156 \IfBabelLayout{lists}
4157 {\bbl@sreplace\list{\parshape}{\bbl@listparshape}%
4158 \def\bbl@listparshape#1#2#3{%
4159 \parshape #1 #2 #3 %
4160 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4161 \shapemode\tw@
4162 \fi}}
4163 {}
4164 \IfBabelLayout{graphics}
4165 {\let\bbl@pictresetdir\relax
4166 \def\bbl@pictsetdir{%
4167 \ifcase\bbl@thetextdir
4168 \let\bbl@pictresetdir\relax
4169 \else
4170 \textdir TLT\relax
4171 \def\bbl@pictresetdir{\textdir TRT\relax}%
4172 \fi}%
4173 \bbl@sreplace\@picture{\hskip-}{\bbl@pictsetdir\hskip-}%
4174 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
4175 \@killglue
4176 \raise#2\unitlength
4177 \hb@xt@#1\z@{\kern#1\unitlength{\bbl@pictresetdir#3}\hss}}%
4178 \AtBeginDocument
4179 {\ifx\tikz@atbegin@node\undefined\else
4180 \bbl@sreplace\pgfpicture{\pgfpicturetrue}{\bbl@pictsetdir\pgfpicturetrue}%
4181 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir}%
4182 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
4183 \fi}}
4184 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

```

4185 \IfBabelLayout{counters}%
4186 {\bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
4187 \let\bbl@latinarabic=\@arabic
4188 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4189 \@ifpackagewith{babel}{bidi=default}%
4190 {\let\bbl@asciroman=\@roman
4191 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4192 \let\bbl@asciiRoman=\@Roman
4193 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
4194 \def\labelenumii{}\theenumii{}%
4195 \def\p@enumiii{\p@enumii}\theenumii{}\}}%
4196 <<Footnote changes>>
4197 \IfBabelLayout{footnotes}%
4198 {\BabelFootnote\footnote\languagename{}\}%
4199 \BabelFootnote\localfootnote\languagename{}\}%

```

```

4200 \BabelFootnote\mainfootnote{}{}{}
4201 {}

```

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.

```

4202 \IfBabelLayout{extras}%
4203 {\bbl@sreplace\underline{$\@@underline}\bbl@nextfake$\@@underline}%
4204 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
4205   \if b\expandafter\@car\@series\@nil\boldmath\fi
4206   \babelsublr}%
4207   \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
4208 {}
4209 </luatex>

```

14.7 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it'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):

Arrrrgh!! 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.

```

4210 <(*basic-r)>
4211 Babel = Babel or {}
4212
4213 Babel.bidi_enabled = true
4214
4215 require('babel-data-bidi.lua')
4216
4217 local characters = Babel.characters
4218 local ranges = Babel.ranges
4219
4220 local DIR = node.id("dir")

```

```

4221
4222 local function dir_mark(head, from, to, outer)
4223   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
4224   local d = node.new(DIR)
4225   d.dir = '+' .. dir
4226   node.insert_before(head, from, d)
4227   d = node.new(DIR)
4228   d.dir = '-' .. dir
4229   node.insert_after(head, to, d)
4230 end
4231
4232 function Babel.bidi(head, ispar)
4233   local first_n, last_n          -- first and last char with nums
4234   local last_es                  -- an auxiliary 'last' used with nums
4235   local first_d, last_d          -- first and last char in L/R block
4236   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):

```

4237   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
4238   local strong_lr = (strong == 'l') and 'l' or 'r'
4239   local outer = strong
4240
4241   local new_dir = false
4242   local first_dir = false
4243   local inmath = false
4244
4245   local last_lr
4246
4247   local type_n = ''
4248
4249   for item in node.traverse(head) do
4250
4251     -- three cases: glyph, dir, otherwise
4252     if item.id == node.id'glyph'
4253       or (item.id == 7 and item.subtype == 2) then
4254
4255       local itemchar
4256       if item.id == 7 and item.subtype == 2 then
4257         itemchar = item.replace.char
4258       else
4259         itemchar = item.char
4260       end
4261       local chardata = characters[itemchar]
4262       dir = chardata and chardata.d or nil
4263       if not dir then
4264         for nn, et in ipairs(ranges) do
4265           if itemchar < et[1] then
4266             break
4267           elseif itemchar <= et[2] then
4268             dir = et[3]
4269             break
4270           end
4271         end
4272       end
4273       dir = dir or 'l'
4274       if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end

```

Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a 'dir' node. We don't know the current language until then. This is not exactly true, as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute force (just above).

```

4275     if new_dir then
4276         attr_dir = 0
4277         for at in node.traverse(item.attr) do
4278             if at.number == luatexbase.registernumber'bbl@attr@dir' then
4279                 attr_dir = at.value % 3
4280             end
4281         end
4282         if attr_dir == 1 then
4283             strong = 'r'
4284         elseif attr_dir == 2 then
4285             strong = 'al'
4286         else
4287             strong = 'l'
4288         end
4289         strong_lr = (strong == 'l') and 'l' or 'r'
4290         outer = strong_lr
4291         new_dir = false
4292     end
4293
4294     if dir == 'nsm' then dir = strong end          -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

4295     dir_real = dir          -- We need dir_real to set strong below
4296     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:

```

4297     if strong == 'al' then
4298         if dir == 'en' then dir = 'an' end          -- W2
4299         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
4300         strong_lr = 'r'          -- W3
4301     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

4302     elseif item.id == node.id'dir' and not inmath then
4303         new_dir = true
4304         dir = nil
4305     elseif item.id == node.id'math' then
4306         inmath = (item.subtype == 0)
4307     else
4308         dir = nil          -- Not a char
4309     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>.

```

4310     if dir == 'en' or dir == 'an' or dir == 'et' then
4311         if dir ~= 'et' then
4312             type_n = dir

```

```

4313     end
4314     first_n = first_n or item
4315     last_n = last_es or item
4316     last_es = nil
4317   elseif dir == 'es' and last_n then -- W3+W6
4318     last_es = item
4319   elseif dir == 'cs' then           -- it's right - do nothing
4320   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
4321     if strong_lr == 'r' and type_n ~= '' then
4322       dir_mark(head, first_n, last_n, 'r')
4323     elseif strong_lr == 'l' and first_d and type_n == 'an' then
4324       dir_mark(head, first_n, last_n, 'r')
4325       dir_mark(head, first_d, last_d, outer)
4326       first_d, last_d = nil, nil
4327     elseif strong_lr == 'l' and type_n ~= '' then
4328       last_d = last_n
4329     end
4330     type_n = ''
4331     first_n, last_n = nil, nil
4332   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:

```

4333   if dir == 'l' or dir == 'r' then
4334     if dir ~= outer then
4335       first_d = first_d or item
4336       last_d = item
4337     elseif first_d and dir ~= strong_lr then
4338       dir_mark(head, first_d, last_d, outer)
4339       first_d, last_d = nil, nil
4340     end
4341   end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resptly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

4342   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
4343     item.char = characters[item.char] and
4344       characters[item.char].m or item.char
4345   elseif (dir or new_dir) and last_lr ~= item then
4346     local mir = outer .. strong_lr .. (dir or outer)
4347     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
4348       for ch in node.traverse(node.next(last_lr)) do
4349         if ch == item then break end
4350         if ch.id == node.id'glyph' then
4351           ch.char = characters[ch.char].m or ch.char
4352         end
4353       end
4354     end
4355   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).

```

4356   if dir == 'l' or dir == 'r' then

```



```

4357     last_lr = item
4358     strong = dir_real          -- Don't search back - best save now
4359     strong_lr = (strong == 'l') and 'l' or 'r'
4360     elseif new_dir then
4361         last_lr = nil
4362     end
4363 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

4364 if last_lr and outer == 'r' then
4365     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
4366         ch.char = characters[ch.char].m or ch.char
4367     end
4368 end
4369 if first_n then
4370     dir_mark(head, first_n, last_n, outer)
4371 end
4372 if first_d then
4373     dir_mark(head, first_d, last_d, outer)
4374 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

4375 return node.prev(head) or head
4376 end
4377 </basic-r>

```

And here the Lua code for bidi=basic:

```

4378 (*basic)
4379 Babel = Babel or {}
4380
4381 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
4382
4383 Babel.fontmap = Babel.fontmap or {}
4384 Babel.fontmap[0] = {}      -- l
4385 Babel.fontmap[1] = {}      -- r
4386 Babel.fontmap[2] = {}      -- al/an
4387
4388 Babel.bidi_enabled = true
4389 Babel.mirroring_enabled = true
4390
4391 -- Temporary:
4392
4393 if harf then
4394     Babel.mirroring_enabled = false
4395 end
4396
4397 require('babel-data-bidi.lua')
4398
4399 local characters = Babel.characters
4400 local ranges = Babel.ranges
4401
4402 local DIR = node.id('dir')
4403 local GLYPH = node.id('glyph')
4404
4405 local function insert_implicit(head, state, outer)
4406     local new_state = state
4407     if state.sim and state.eim and state.sim ~= state.eim then
4408         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse

```

```

4409     local d = node.new(DIR)
4410     d.dir = '+' .. dir
4411     node.insert_before(head, state.sim, d)
4412     local d = node.new(DIR)
4413     d.dir = '-' .. dir
4414     node.insert_after(head, state.eim, d)
4415 end
4416 new_state.sim, new_state.eim = nil, nil
4417 return head, new_state
4418 end
4419
4420 local function insert_numeric(head, state)
4421     local new
4422     local new_state = state
4423     if state.san and state.ean and state.san ~= state.ean then
4424         local d = node.new(DIR)
4425         d.dir = '+TLT'
4426         _, new = node.insert_before(head, state.san, d)
4427         if state.san == state.sim then state.sim = new end
4428         local d = node.new(DIR)
4429         d.dir = '-TLT'
4430         _, new = node.insert_after(head, state.ean, d)
4431         if state.ean == state.eim then state.eim = new end
4432     end
4433     new_state.san, new_state.ean = nil, nil
4434     return head, new_state
4435 end
4436
4437 -- TODO - \hbox with an explicit dir can lead to wrong results
4438 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
4439 -- was s made to improve the situation, but the problem is the 3-dir
4440 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
4441 -- well.
4442
4443 function Babel.bidi(head, ispar, hdir)
4444     local d -- d is used mainly for computations in a loop
4445     local prev_d = ''
4446     local new_d = false
4447
4448     local nodes = {}
4449     local outer_first = nil
4450     local inmath = false
4451
4452     local glue_d = nil
4453     local glue_i = nil
4454
4455     local has_en = false
4456     local first_et = nil
4457
4458     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
4459
4460     local save_outer
4461     local temp = node.get_attribute(head, ATDIR)
4462     if temp then
4463         temp = temp % 3
4464         save_outer = (temp == 0 and 'l') or
4465                     (temp == 1 and 'r') or
4466                     (temp == 2 and 'al')
4467     elseif ispar then -- Or error? Shouldn't happen

```

```

4468     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
4469 else                                     -- Or error? Shouldn't happen
4470     save_outer = ('TRT' == hdir) and 'r' or 'l'
4471 end
4472 -- when the callback is called, we are just _after_ the box,
4473 -- and the textdir is that of the surrounding text
4474 -- if not ispar and hdir ~= tex.textdir then
4475 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
4476 -- end
4477 local outer = save_outer
4478 local last = outer
4479 -- 'al' is only taken into account in the first, current loop
4480 if save_outer == 'al' then save_outer = 'r' end
4481
4482 local fontmap = Babel.fontmap
4483
4484 for item in node.traverse(head) do
4485
4486     -- In what follows, #node is the last (previous) node, because the
4487     -- current one is not added until we start processing the neutrals.
4488
4489     -- three cases: glyph, dir, otherwise
4490     if item.id == GLYPH
4491         or (item.id == 7 and item.subtype == 2) then
4492
4493         local d_font = nil
4494         local item_r
4495         if item.id == 7 and item.subtype == 2 then
4496             item_r = item.replace      -- automatic discs have just 1 glyph
4497         else
4498             item_r = item
4499         end
4500         local chardata = characters[item_r.char]
4501         d = chardata and chardata.d or nil
4502         if not d or d == 'nsm' then
4503             for nn, et in ipairs(ranges) do
4504                 if item_r.char < et[1] then
4505                     break
4506                 elseif item_r.char <= et[2] then
4507                     if not d then d = et[3]
4508                     elseif d == 'nsm' then d_font = et[3]
4509                     end
4510                     break
4511                 end
4512             end
4513         end
4514         d = d or 'l'
4515
4516         -- A short 'pause' in bidi for mapfont
4517         d_font = d_font or d
4518         d_font = (d_font == 'l' and 0) or
4519             (d_font == 'nsm' and 0) or
4520             (d_font == 'r' and 1) or
4521             (d_font == 'al' and 2) or
4522             (d_font == 'an' and 2) or nil
4523         if d_font and fontmap and fontmap[d_font][item_r.font] then
4524             item_r.font = fontmap[d_font][item_r.font]
4525         end
4526

```

```

4527     if new_d then
4528         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4529     if inmath then
4530         attr_d = 0
4531     else
4532         attr_d = node.get_attribute(item, ATDIR)
4533         attr_d = attr_d % 3
4534     end
4535     if attr_d == 1 then
4536         outer_first = 'r'
4537         last = 'r'
4538     elseif attr_d == 2 then
4539         outer_first = 'r'
4540         last = 'al'
4541     else
4542         outer_first = 'l'
4543         last = 'l'
4544     end
4545     outer = last
4546     has_en = false
4547     first_et = nil
4548     new_d = false
4549 end
4550
4551 if glue_d then
4552     if (d == 'l' and 'l' or 'r') ~= glue_d then
4553         table.insert(nodes, {glue_i, 'on', nil})
4554     end
4555     glue_d = nil
4556     glue_i = nil
4557 end
4558
4559 elseif item.id == DIR then
4560     d = nil
4561     new_d = true
4562
4563 elseif item.id == node.id'glue' and item.subtype == 13 then
4564     glue_d = d
4565     glue_i = item
4566     d = nil
4567
4568 elseif item.id == node.id'math' then
4569     inmath = (item.subtype == 0)
4570
4571 else
4572     d = nil
4573 end
4574
4575 -- AL <= EN/ET/ES      -- W2 + W3 + W6
4576 if last == 'al' and d == 'en' then
4577     d = 'an'          -- W3
4578 elseif last == 'al' and (d == 'et' or d == 'es') then
4579     d = 'on'          -- W6
4580 end
4581
4582 -- EN + CS/ES + EN      -- W4
4583 if d == 'en' and #nodes >= 2 then
4584     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4585         and nodes[#nodes-1][2] == 'en' then

```

```

4586         nodes[#nodes][2] = 'en'
4587     end
4588 end
4589
4590 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
4591 if d == 'an' and #nodes >= 2 then
4592     if (nodes[#nodes][2] == 'cs')
4593         and nodes[#nodes-1][2] == 'an' then
4594         nodes[#nodes][2] = 'an'
4595     end
4596 end
4597
4598 -- ET/EN                  -- W5 + W7->l / W6->on
4599 if d == 'et' then
4600     first_et = first_et or (#nodes + 1)
4601 elseif d == 'en' then
4602     has_en = true
4603     first_et = first_et or (#nodes + 1)
4604 elseif first_et then      -- d may be nil here !
4605     if has_en then
4606         if last == 'l' then
4607             temp = 'l'    -- W7
4608         else
4609             temp = 'en'   -- W5
4610         end
4611     else
4612         temp = 'on'       -- W6
4613     end
4614     for e = first_et, #nodes do
4615         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4616     end
4617     first_et = nil
4618     has_en = false
4619 end
4620
4621 if d then
4622     if d == 'al' then
4623         d = 'r'
4624         last = 'al'
4625     elseif d == 'l' or d == 'r' then
4626         last = d
4627     end
4628     prev_d = d
4629     table.insert(nodes, {item, d, outer_first})
4630 end
4631
4632 outer_first = nil
4633
4634 end
4635
4636 -- TODO -- repeated here in case EN/ET is the last node. Find a
4637 -- better way of doing things:
4638 if first_et then      -- dir may be nil here !
4639     if has_en then
4640         if last == 'l' then
4641             temp = 'l'    -- W7
4642         else
4643             temp = 'en'   -- W5
4644         end

```

```

4645     else
4646         temp = 'on'      -- W6
4647     end
4648     for e = first_et, #nodes do
4649         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4650     end
4651 end
4652
4653 -- dummy node, to close things
4654 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4655
4656 ----- NEUTRAL -----
4657
4658 outer = save_outer
4659 last = outer
4660
4661 local first_on = nil
4662
4663 for q = 1, #nodes do
4664     local item
4665
4666     local outer_first = nodes[q][3]
4667     outer = outer_first or outer
4668     last = outer_first or last
4669
4670     local d = nodes[q][2]
4671     if d == 'an' or d == 'en' then d = 'r' end
4672     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4673
4674     if d == 'on' then
4675         first_on = first_on or q
4676     elseif first_on then
4677         if last == d then
4678             temp = d
4679         else
4680             temp = outer
4681         end
4682         for r = first_on, q - 1 do
4683             nodes[r][2] = temp
4684             item = nodes[r][1]      -- MIRRORING
4685             if Babel.mirroring_enabled and item.id == GLYPH and temp == 'r' then
4686                 item.char = characters[item.char].m or item.char
4687             end
4688         end
4689         first_on = nil
4690     end
4691
4692     if d == 'r' or d == 'l' then last = d end
4693 end
4694
4695 ----- IMPLICIT, REORDER -----
4696
4697 outer = save_outer
4698 last = outer
4699
4700 local state = {}
4701 state.has_r = false
4702
4703 for q = 1, #nodes do

```

```

4704
4705     local item = nodes[q][1]
4706
4707     outer = nodes[q][3] or outer
4708
4709     local d = nodes[q][2]
4710
4711     if d == 'nsm' then d = last end           -- W1
4712     if d == 'en' then d = 'an' end
4713     local isdir = (d == 'r' or d == 'l')
4714
4715     if outer == 'l' and d == 'an' then
4716         state.san = state.san or item
4717         state.ean = item
4718     elseif state.san then
4719         head, state = insert_numeric(head, state)
4720     end
4721
4722     if outer == 'l' then
4723         if d == 'an' or d == 'r' then        -- im -> implicit
4724             if d == 'r' then state.has_r = true end
4725             state.sim = state.sim or item
4726             state.eim = item
4727         elseif d == 'l' and state.sim and state.has_r then
4728             head, state = insert_implicit(head, state, outer)
4729         elseif d == 'l' then
4730             state.sim, state.eim, state.has_r = nil, nil, false
4731         end
4732     else
4733         if d == 'an' or d == 'l' then
4734             if nodes[q][3] then -- nil except after an explicit dir
4735                 state.sim = item -- so we move sim 'inside' the group
4736             else
4737                 state.sim = state.sim or item
4738             end
4739             state.eim = item
4740         elseif d == 'r' and state.sim then
4741             head, state = insert_implicit(head, state, outer)
4742         elseif d == 'r' then
4743             state.sim, state.eim = nil, nil
4744         end
4745     end
4746
4747     if isdir then
4748         last = d           -- Don't search back - best save now
4749     elseif d == 'on' and state.san then
4750         state.san = state.san or item
4751         state.ean = item
4752     end
4753
4754 end
4755
4756 return node.prev(head) or head
4757 end
4758 </basic>

```

15 Data for CJK

It is a boring file and it's not shown here. See the generated file.

16 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the @ sign, etc.

```
4759 ⟨*nil⟩
4760 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
4761 \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.

```
4762 \ifx\l@nil\@undefined
4763   \newlanguage\l@nil
4764   \@namedef{bbl@hyphendata@the\l@nil}{{}}{}% Remove warning
4765 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
4766 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 4767 \let\captionnil\@empty
4768 \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.

```
4769 \ldf@finish{nil}
4770 ⟨/nil⟩
```

17 Support for Plain T_EX (plain.def)

17.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_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 `locallyhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `lplain.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 `lplain.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 \LaTeX sees, we need to set some category codes just to be able to change the definition of \input

```
4771 <*bplain | bplain>
4772 \catcode`\{=1 % left brace is begin-group character
4773 \catcode`\}=2 % right brace is end-group character
4774 \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...

```
4775 \openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
4776 \ifeof0
4777 \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).

```
4778 \let\input
```

Then \input is defined to forget about its argument and load `hyphen.cfg` instead.

```
4779 \def\input #1 {%
4780   \let\input\input
4781   \input hyphen.cfg
```

Once that's done the original meaning of \input can be restored and the definition of \input can be forgotten.

```
4782   \let\input\undefined
4783 }
4784 \fi
4785 </bplain | bplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
4786 <bplain>\input plain.tex
4787 <bplain>\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.

```
4788 <bplain>\def\fmtname{babel-plain}
4789 <bplain>\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

17.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of $\text{\LaTeX} 2_{\epsilon}$ that are needed for `babel`.

```
4790 <*plain>
4791 \def\@empty{}
4792 \def\loadlocalcfg#1{%
4793   \openin0#1.cfg
4794   \ifeof0
4795     \closein0
4796   \else
```

```

4797 \closein0
4798 {\immediate\write16{*****}%
4799 \immediate\write16{* Local config file #1.cfg used}%
4800 \immediate\write16{*}%
4801 }
4802 \input #1.cfg\relax
4803 \fi
4804 \@endoflfd}

```

17.3 General tools

A number of \LaTeX macro's that are needed later on.

```

4805 \long\def\@firstofone#1{#1}
4806 \long\def\@firstoftwo#1#2{#1}
4807 \long\def\@secondoftwo#1#2{#2}
4808 \def\@nnil{\@nil}
4809 \def\@gobbletwo#1#2{}
4810 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4811 \def\@star@or@long#1{%
4812 \@ifstar
4813 {\let\l@ngrel@x\relax#1}%
4814 {\let\l@ngrel@x\long#1}}
4815 \let\l@ngrel@x\relax
4816 \def\@car#1#2\@nil{#1}
4817 \def\@cdr#1#2\@nil{#2}
4818 \let\@typeset@protect\relax
4819 \let\protected@edef\edef
4820 \long\def\@gobble#1{}
4821 \edef\@backslashchar{\expandafter\@gobble\string\}
4822 \def\strip@prefix#1>{}
4823 \def\g@addto@macro#1#2{%
4824 \toks@\expandafter{#1#2}%
4825 \xdef#1{\the\toks@}}
4826 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4827 \def\@nameuse#1{\csname #1\endcsname}
4828 \def\@ifundefined#1{%
4829 \expandafter\ifx\csname#1\endcsname\relax
4830 \expandafter\@firstoftwo
4831 \else
4832 \expandafter\@secondoftwo
4833 \fi}
4834 \def\@expandtwoargs#1#2#3{%
4835 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4836 \def\zap@space#1 #2{%
4837 #1%
4838 \ifx#2\@empty\else\expandafter\zap@space\fi
4839 #2}

```

$\text{\LaTeX} 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

4840 \ifx\@preamblecmds\@undefined
4841 \def\@preamblecmds{}
4842 \fi
4843 \def\@onlypreamble#1{%
4844 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4845 \@preamblecmds\do#1}}
4846 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

4847 \def\begindocument{%
4848   \@begindocumenthook
4849   \global\let\@begindocumenthook\@undefined
4850   \def\do##1{\global\let##1\@undefined}%
4851   \@preamblecmds
4852   \global\let\do\noexpand}

4853 \ifx\@begindocumenthook\@undefined
4854   \def\@begindocumenthook{}
4855 \fi
4856 \@onlypreamble\@begindocumenthook
4857 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

4858 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
4859 \@onlypreamble\AtEndOfPackage
4860 \def\@endoflfd{}
4861 \@onlypreamble\@endoflfd
4862 \let\bbbl@afterlang\@empty
4863 \chardef\bbbl@opt@hyphenmap\z@

```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

4864 \ifx@if@files\@undefined
4865   \expandafter\let\csname if@files\expandafter\endcsname
4866   \csname iffalse\endcsname
4867 \fi

```

Mimick \LaTeX 's commands to define control sequences.

```

4868 \def\newcommand{\@star@or@long\new@command}
4869 \def\new@command#1{%
4870   \@testopt{\@newcommand#1}0}
4871 \def\@newcommand#1[#2]{%
4872   \@ifnextchar [{\@xargdef#1[#2]}%
4873   {\@argdef#1[#2]}}
4874 \long\def\@argdef#1[#2]#3{%
4875   \@yargdef#1\@ne{#2}{#3}}
4876 \long\def\@xargdef#1[#2][#3]#4{%
4877   \expandafter\def\expandafter#1\expandafter{%
4878     \expandafter\@protected@testopt\expandafter #1%
4879     \csname\string#1\expandafter\endcsname{#3}}%
4880   \expandafter\@yargdef \csname\string#1\endcsname
4881   \tw@{#2}{#4}}
4882 \long\def\@yargdef#1#2#3{%
4883   \@tempcnta#3\relax
4884   \advance \@tempcnta \@ne
4885   \let\@hash@\relax
4886   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4887   \@tempcntb #2%
4888   \@whilenum\@tempcntb <\@tempcnta
4889     \do{%
4890       \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
4891       \advance\@tempcntb \@ne}%
4892   \let\@hash@###
4893   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
4894 \def\providecommand{\@star@or@long\provide@command}

```

```

4895 \def\provide@command#1{%
4896   \begingroup
4897     \escapechar\m@ne\xdef\gtempa{\string#1}%
4898   \endgroup
4899   \expandafter\ifundefined\gtempa
4900     {\def\reserved@a{\new@command#1}}%
4901     {\let\reserved@a\relax
4902      \def\reserved@a{\new@command\reserved@a}}%
4903   \reserved@a}%

4904 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4905 \def\declare@robustcommand#1{%
4906   \edef\reserved@a{\string#1}%
4907   \def\reserved@b{#1}%
4908   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4909   \edef#1{%
4910     \ifx\reserved@a\reserved@b
4911       \noexpand\x@protect
4912       \noexpand#1%
4913     \fi
4914     \noexpand\protect
4915     \expandafter\noexpand\csname
4916       \expandafter\@gobble\string#1 \endcsname
4917   }%
4918   \expandafter\new@command\csname
4919     \expandafter\@gobble\string#1 \endcsname
4920 }
4921 \def\x@protect#1{%
4922   \ifx\protect\@typeset@protect\else
4923     \@x@protect#1%
4924   \fi
4925 }
4926 \def\@x@protect#1\fi#2#3{%
4927   \fi\protect#1%
4928 }

```

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`.

```

4929 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4930 \ifx\in@\@undefined
4931   \def\in@#1#2{%
4932     \def\in@##1#1##2##3\in@{%
4933       \ifx\in@##2\in@false\else\in@true\fi}%
4934     \in@#2#1\in@\in@}
4935 \else
4936   \let\bbl@tempa\@empty
4937 \fi
4938 \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).

```

4939 \def\@ifpackagewith#1#2#3#4{#3}

```

The \LaTeX macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```
4940 \def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\LaTeX 2_\epsilon$ versions; just enough to make things work in plain \TeX environments.

```
4941 \ifx\@tempcnta\@undefined
4942   \csname newcount\endcsname\@tempcnta\relax
4943 \fi
4944 \ifx\@tempcntb\@undefined
4945   \csname newcount\endcsname\@tempcntb\relax
4946 \fi
```

To prevent wasting two counters in $\LaTeX 2.09$ (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
4947 \ifx\bye\@undefined
4948   \advance\count10 by -2\relax
4949 \fi
4950 \ifx\@ifnextchar\@undefined
4951   \def\@ifnextchar#1#2#3{%
4952     \let\reserved@d=#1%
4953     \def\reserved@a{#2}\def\reserved@b{#3}%
4954     \futurelet\@let@token\@ifnch}
4955   \def\@ifnch{%
4956     \ifx\@let@token\@sptoken
4957       \let\reserved@c\@xifnch
4958     \else
4959       \ifx\@let@token\reserved@d
4960         \let\reserved@c\reserved@a
4961       \else
4962         \let\reserved@c\reserved@b
4963       \fi
4964     \fi
4965     \reserved@c}
4966   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
4967   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
4968 \fi
4969 \def\@testopt#1#2{%
4970   \@ifnextchar[#{#1}{#1[#2]}}
4971 \def\@protected@testopt#1{%
4972   \ifx\protect\@typeset@protect
4973     \expandafter\@testopt
4974   \else
4975     \@x@protect#1%
4976   \fi}
4977 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4978   #2\relax}\fi}
4979 \long\def\@iwhilenum#1{\ifnum #1\relax\expandafter\@iwhilenum
4980   \else\expandafter\@gobble\fi{#1}}
```

17.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```
4981 \def\DeclareTextCommand{%
4982   \@dec@text@cmd\providecommand
4983 }
```

```

4984 \def\ProvideTextCommand{%
4985   \@dec@text@cmd\providecommand
4986 }
4987 \def\DeclareTextSymbol#1#2#3{%
4988   \@dec@text@cmd\chardef#1{#2}#3\relax
4989 }
4990 \def\@dec@text@cmd#1#2#3{%
4991   \expandafter\def\expandafter#2%
4992     \expandafter{%
4993       \csname#3-cmd\expandafter\endcsname
4994       \expandafter#2%
4995       \csname#3\string#2\endcsname
4996     }%
4997 %   \let\@ifdefinable\@rc@ifdefinable
4998   \expandafter#1\csname#3\string#2\endcsname
4999 }
5000 \def\@current@cmd#1{%
5001   \ifx\protect\@typeset@protect\else
5002     \noexpand#1\expandafter\@gobble
5003   \fi
5004 }
5005 \def\@changed@cmd#1#2{%
5006   \ifx\protect\@typeset@protect
5007     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
5008       \expandafter\ifx\csname ?\string#1\endcsname\relax
5009         \expandafter\def\csname ?\string#1\endcsname{%
5010           \@changed@x@err{#1}%
5011         }%
5012       \fi
5013       \global\expandafter\let
5014         \csname\cf@encoding \string#1\expandafter\endcsname
5015         \csname ?\string#1\endcsname
5016       \fi
5017       \csname\cf@encoding\string#1%
5018       \expandafter\endcsname
5019     \else
5020       \noexpand#1%
5021     \fi
5022 }
5023 \def\@changed@x@err#1{%
5024   \errhelp{Your command will be ignored, type <return> to proceed}%
5025   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
5026 \def\DeclareTextCommandDefault#1{%
5027   \DeclareTextCommand#1?%
5028 }
5029 \def\ProvideTextCommandDefault#1{%
5030   \ProvideTextCommand#1?%
5031 }
5032 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
5033 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
5034 \def\DeclareTextAccent#1#2#3{%
5035   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
5036 }
5037 \def\DeclareTextCompositeCommand#1#2#3#4{%
5038   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
5039   \edef\reserved@b{\string#1}%
5040   \edef\reserved@c{%
5041     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
5042   \ifx\reserved@b\reserved@c

```

```

5043 \expandafter\expandafter\expandafter\ifx
5044 \expandafter\@car\reserved@a\relax\relax\@nil
5045 \@text@composite
5046 \else
5047 \edef\reserved@b##1{%
5048 \def\expandafter\noexpand
5049 \csname#2\string#1\endcsname####1{%
5050 \noexpand\@text@composite
5051 \expandafter\noexpand\csname#2\string#1\endcsname
5052 ####1\noexpand\@empty\noexpand\@text@composite
5053 {##1}%
5054 }%
5055 }%
5056 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
5057 \fi
5058 \expandafter\def\csname\expandafter\string\csname
5059 #2\endcsname\string#1-\string#3\endcsname{#4}
5060 \else
5061 \errhelp{Your command will be ignored, type <return> to proceed}%
5062 \errmessage{\string\DeclareTextCompositeCommand\space used on
5063 inappropriate command \protect#1}
5064 \fi
5065 }
5066 \def\@text@composite#1#2#3\@text@composite{%
5067 \expandafter\@text@composite@x
5068 \csname\string#1-\string#2\endcsname
5069 }
5070 \def\@text@composite@x#1#2{%
5071 \ifx#1\relax
5072 #2%
5073 \else
5074 #1%
5075 \fi
5076 }
5077 %
5078 \def\@strip@args#1:#2-#3\@strip@args{#2}
5079 \def\DeclareTextComposite#1#2#3#4{%
5080 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
5081 \bgroup
5082 \lccode\@=#4%
5083 \lowercase{%
5084 \egroup
5085 \reserved@a @%
5086 }%
5087 }
5088 %
5089 \def\UseTextSymbol#1#2{%
5090 % \let\@curr@enc\cf@encoding
5091 % \@use@text@encoding{#1}%
5092 #2%
5093 % \@use@text@encoding\@curr@enc
5094 }
5095 \def\UseTextAccent#1#2#3{%
5096 % \let\@curr@enc\cf@encoding
5097 % \@use@text@encoding{#1}%
5098 % #2{\@use@text@encoding\@curr@enc\selectfont#3}%
5099 % \@use@text@encoding\@curr@enc
5100 }
5101 \def\@use@text@encoding#1{%

```

```

5102 % \edef\fontencoding{#1}%
5103 % \xdef\fontname{%
5104 % \csname\curr@fontshape/\fontsize\endcsname
5105 % }%
5106 % \pickup@font
5107 % \fontname
5108 % \@@enc@update
5109 }
5110 \def\DeclareTextSymbolDefault#1#2{%
5111 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
5112 }
5113 \def\DeclareTextAccentDefault#1#2{%
5114 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
5115 }
5116 \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.

```

5117 \DeclareTextAccent{"}{OT1}{127}
5118 \DeclareTextAccent{'}{OT1}{19}
5119 \DeclareTextAccent{^}{OT1}{94}
5120 \DeclareTextAccent`}{OT1}{18}
5121 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```

5122 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
5123 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
5124 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
5125 \DeclareTextSymbol{\textquoteright}{OT1}{''}
5126 \DeclareTextSymbol{\i}{OT1}{16}
5127 \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`.

```

5128 \ifx\scriptsize\@undefined
5129 \let\scriptsize\sevenrm
5130 \fi
5131 </plain>

```

18 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 T_EX*, *TUGboat* 9 (1988) #1, p. 70–72.
- [6] Leslie Lamport, in: T_EXhax Digest, Volume 89, #13, 17 February 1989.
- [7] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national L^AT_EX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [8] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [9] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [10] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [11] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.