

Babel

Version 3.31
2019/05/04

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 lualatex	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	25
1.16	Creating a language	26
1.17	Digits	29
1.18	Getting the current language name	29
1.19	Hyphenation and line breaking	29
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	37
1.25	Tips, workarounds, know issues and notes	38
1.26	Current and future work	40
1.27	Tentative and experimental code	40
2	Loading languages with language.dat	40
2.1	Format	41
3	The interface between the core of babel and the language definition files	41
3.1	Guidelines for contributed languages	43
3.2	Basic macros	43
3.3	Skeleton	44
3.4	Support for active characters	45
3.5	Support for saving macro definitions	46
3.6	Support for extending macros	46
3.7	Macros common to a number of languages	46
3.8	Encoding-dependent strings	47
4	Changes	50
4.1	Changes in babel version 3.9	50
II	Source code	51
5	Identification and loading of required files	51
6	locale directory	51

7	Tools	52
7.1	Multiple languages	55
8	The Package File (\LaTeX, babel.sty)	56
8.1	base	57
8.2	key=value options and other general option	59
8.3	Conditional loading of shorthands	60
8.4	Language options	61
9	The kernel of Babel (babel.def, common)	64
9.1	Tools	64
9.2	Hooks	67
9.3	Setting up language files	69
9.4	Shorthands	70
9.5	Language attributes	80
9.6	Support for saving macro definitions	82
9.7	Short tags	83
9.8	Hyphens	83
9.9	Multiencoding strings	85
9.10	Macros common to a number of languages	91
9.11	Making glyphs available	91
9.11.1	Quotation marks	91
9.11.2	Letters	92
9.11.3	Shorthands for quotation marks	93
9.11.4	Umlauts and tremas	94
9.12	Layout	95
9.13	Load engine specific macros	96
9.14	Creating languages	96
10	The kernel of Babel (babel.def, only \LaTeX)	106
10.1	The redefinition of the style commands	106
10.2	Cross referencing macros	107
10.3	Marks	110
10.4	Preventing clashes with other packages	111
10.4.1	ifthen	111
10.4.2	varioref	112
10.4.3	hhline	112
10.4.4	hyperref	113
10.4.5	fancyhdr	113
10.5	Encoding and fonts	113
10.6	Basic bidi support	115
10.7	Local Language Configuration	118
11	Multiple languages (switch.def)	119
11.1	Selecting the language	120
11.2	Errors	128
12	Loading hyphenation patterns	130
13	Font handling with fontspec	135

14	Hooks for XeTeX and LuaTeX	139
14.1	XeTeX	139
14.2	Layout	141
14.3	LuaTeX	144
14.4	Southeast Asian scripts	149
14.5	CJK line breaking	152
14.6	Layout	152
14.7	Auto bidi with basic and basic-r	154
15	Data for CJK	165
16	The ‘nil’ language	165
17	Support for Plain TeX (plain.def)	165
17.1	Not renaming hyphen.tex	165
17.2	Emulating some L ^A T _E X features	167
17.3	General tools	167
17.4	Encoding related macros	171
18	Acknowledgements	173

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 on <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 on <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 several options. The last one is considered the main language, activated by default. Sometimes, the main language changes the document layout (eg, spanish and french).

EXAMPLE In \LaTeX , the preamble of the document:

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

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

You can also set the main language explicitly:

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

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

To switch the language there are two basic macros, 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 L^AT_EX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to lmrroman. 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 L^AT_EX (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` $\langle\textit{language}\rangle\langle\textit{text}\rangle$

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

1.8 Auxiliary language selectors

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

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

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

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

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

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

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

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

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

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

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

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

NOTE Note the following:

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

A typical error when using shorthands is the following:

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

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

`\shorthandon` \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.

`\useshorthands` `*{\char}`

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

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

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

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

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

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

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

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

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

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

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

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, `\aliashorthands` is ignored).

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

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

WARNING Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand 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` nothing happens.

\languageshorthands $\langle language \rangle$

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

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

(You may also need to activate them with, for example, `\usesshorthands`.) Very often, this is a more convenient way to deactivate shorthands than `\shorthandoff`, as for example if you want to define a macro to easy typing phonetic characters with tipa:

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

`\babelshorthand` $\langle shorthand \rangle$

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

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

Languages with no shorthands Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh
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 `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** New 3.9l No warnings and no *infos* are written to the log file.⁹
- strings=** generic | unicode | encoded | $\langle label \rangle$ | $\langle font\ encoding \rangle$
 Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional T_EX, LICR and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal L^AT_EX 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 `\begin{document}`), but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated;¹¹
select sets it only at `\selectlanguage`;
other also sets it at `otherlanguage`;
other* also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.¹²
- bidi=** default | basic | 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 $\langle code \rangle$ when the file loaded by the corresponding package option is finished (at $\backslash ldf@finish$). 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 $\langle option-name \rangle$ is the same as $\backslash CurrentOption$ (which could not be the same as the option name as set in $\backslash usepackage!$).

EXAMPLE Consider two languages foo and bar defining the same $\backslash macro$ with $\backslash newcommand$. 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 $\backslash babelprovide$), but a higher interface, based on package options, is under development (in other words, $\backslash babelprovide$ 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:

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

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

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

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

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

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{1໐ 1໑ 1 1໒ 1໓ 1໔ 1໕} % Random
```

Khmer clusters are rendered wrongly.

East Asia scripts Internal inconsistencies in script and language names must be sorted out, so you may need to set them explicitly in \babelfont, as well as CJKShape. luatex does basic line breaking, but currently xetex does not (you may load zhspacing). Anyway, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX...), although for a few words and shorts texts babel should be fine.

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

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

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 ^l	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 ^l
hy	Armenian	ms-SG	Malay ^l
ia	Interlingua ^{ul}	ms	Malay ^{ul}
id	Indonesian ^{ul}	mt	Maltese
ig	Igbo	mua	Mundang
ii	Sichuan Yi	my	Burmese
is	Icelandic ^{ul}	mzn	Mazanderani
it	Italian ^{ul}	naq	Nama
ja	Japanese	nb	Norwegian Bokmål ^{ul}
jgo	Ngomba	nd	North Ndebele
jmc	Machame	ne	Nepali

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

In some contexts (currently `\babel font`) an `ini` file may be loaded by its name. Here is the list of the names currently supported. With these languages, `\babel font` 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	chinese-hans-hk
akan	chinese-hans-mo
albanian	chinese-hans-sg
american	chinese-hans
amharic	chinese-hant-hk
arabic	chinese-hant-mo
arabic-algeria	chinese-hant
arabic-DZ	chinese-simplified-hongkongsarchina
arabic-morocco	chinese-simplified-macausarchina
arabic-MA	chinese-simplified-singapore
arabic-syria	chinese-simplified
arabic-SY	chinese-traditional-hongkongsarchina
armenian	chinese-traditional-macausarchina
assamese	chinese-traditional
asturian	chinese
asu	cognian
australian	cornish
austrian	croatian
azerbaijani-cyrillic	czech
azerbaijani-cyrl	danish
azerbaijani-latin	duala
azerbaijani-latn	dutch
azerbaijani	dzongkha
bafia	embu
bambara	english-au
basaa	english-australia
basque	english-ca
belarusian	english-canada
bemba	english-gb
bena	english-newzealand
bengali	english-nz
bodo	english-unitedkingdom
bosnian-cyrillic	english-unitedstates
bosnian-cyrl	english-us
bosnian-latin	english
bosnian-latn	esperanto
bosnian	estonian
brazilian	ewe
breton	ewondo
british	faroes
bulgarian	filipino
burmese	finnish
canadian	french-be
cantonese	french-belgium
catalan	french-ca
centralatlastamazight	french-canada
centralkurdish	french-ch
chechen	french-lu
cherokee	french-luxembourg
chiga	french-switzerland

french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian

lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
portuguese-br

portuguese-brazil	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
serbian-latin-bosniaherzegovina	uzbek
serbian-latin-kosovo	vai-latin
serbian-latin-montenegro	vai-latn
serbian-latin	vai-vai
serbian-latn-ba	vai-vaii
serbian-latn-me	vai
serbian-latn-xk	vietnam
serbian-latn	vietnamese
serbian	vunjo
shambala	walser

welsh	yoruba
westernfrisian	zarma
yangben	zulu afrikaans
yiddish	

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}
```

`\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:

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

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

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.

1.15 Modifying a language

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

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

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

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

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

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

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

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

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

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

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

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= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

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

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

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

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

script= *<script-name>*

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

language= *<language-name>*

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

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

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.

intraspace= *<base> <shrink> <stretch>*

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= *<penalty>*

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

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

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

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

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

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

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

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

1.19 Hyphenation and line breaking

`\babelhyphen` *{<type>}
`\babelhyphen` *{<text>}

New 3.9a It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in \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{<text>}` is a hard “hyphen” using `<text>` instead. A typical case is `\babelhyphen{/}`.

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

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

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

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

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

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

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

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 3.27 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 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` {*⟨text⟩*}

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

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

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.

¹⁹But still defined for backwards compatibility.

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 very likely until (Northern) Winter. This applies to text, but **graphical** elements, including the `picture` environment and PDF or PS based graphics, are not yet correctly handled (far from trivial). Also, indexes and the like are under study, as well as math.

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-l

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 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 now limited). (They are named `basic` mainly because they only consider the intrinsic direction of scripts and weak directionality.)

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

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

EXAMPLE The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember `basic-r` is available in `luatex` only.²⁰

```
\documentclass{article}

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

²⁰At the time of this writing some Arabic fonts are not rendered correctly by the default `luatex` font loader, with misplaced kerns inside some words, so double check the resulting text. Have a look at the workaround available on GitHub, under `/required/babel/samples`

```
\babelprovide[import, main]{arabic}
```

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{فصحى العصر} \textit{fuṣḥā l-‘aṣr} (MSA) and
\textit{فصحى التراث} \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (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, $\langle subsection \rangle . \langle section \rangle$); 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.²¹

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 `\par shape` 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).

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** .

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

²¹Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

Digits in pdfTeX must be marked up explicitly (unlike LaTeX with `bidi=basic` or `bidi=basic-r` and, usually, XeTeX). This command is provided to set `{\lr-text}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `r1` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. 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}{\languagename}{{}}}
```

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

```
\footnote{(\foreignlanguage{\languagename}{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}{\languagename}{{}}%
\BabelFootnote{\localfootnote}{\languagename}{{}}%
\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 `{<name>}{<event>}{<code>}`

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{<name>}`, `\DisableBabelHook{<name>}`. Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`). Current events are the following; in some of them you can use one to three `TEX` parameters (`#1`, `#2`, `#3`), with the meaning given:

- 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}{\languageshortands{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 uppersorbian
Welsh welsh

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

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

```

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

Then you preprocess it with `devnag <file>`, which creates `<file>.tex`; you can then typeset the latter with \LaTeX .

NOTE Please, for info about the support in `luatex` for some complex scripts, see the wiki, on <https://github.com/latex3/latex2e/wiki/Babel:-Remarks-on-the-luatex-support-for-some-scripts>.

1.25 Tips, workarounds, know issues and notes

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

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

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

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

```
\addto\extrasfrench{\inputencoding{latin1}}  
\addto\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.²² 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 `\usesshorthands` to activate ' and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the .aux file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make T_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.

²²This 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.

1.26 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²³ 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.

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.^{er} 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).

1.27 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 “Oth” language, typically english, which is preloaded always).²⁴ Until

²³See 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.

²⁴This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

3.9n, this task was delegated to the package `luatex-hyphen`, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).²⁵

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 stored²⁶. 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.²⁷ 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

²⁵The 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`.

²⁶This is because different operating systems sometimes use *very* different file-naming conventions.

²⁷This is not a new feature, but in former versions it didn't work correctly.

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\langle lang \rangle hyphenmins`, `\langle lang \rangle captions`, `\langle lang \rangle date`, `\langle lang \rangle extras` and `\langle lang \rangle noextras` (the last two may be left empty); where `\langle lang \rangle` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date\langle lang \rangle` but not `\captions\langle lang \rangle` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@⟨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 \LaTeX (quotes are entered as `` `` and `' '`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras\langle lang \rangle` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low level) or the language (high level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁸
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

²⁸But not removed, for backward compatibility.

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

3.1 Guidelines for contributed languages

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

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

3.3 Skeleton

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

```

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

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

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

```

```

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

3.5 Support for saving macro definitions

Language definition files may want to *redefine* macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this²⁹.

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

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

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

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

`\set@low@box` For some languages, quotes need to be lowered to the baseline. For this purpose the macro

²⁹This mechanism was introduced by Bernd Raichle.

`\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.

`\save@sf@q`

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

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

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

3.8 Encoding-dependent strings

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

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

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

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

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

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

Encoding info is `charset=` followed by a `charset`, which if given sets how the strings should be 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 a encoded way).

The $\langle category \rangle$ is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.³⁰ It may be empty, too, but in such a case using \setstring is an error (but not \setcase).

```

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

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

\EndBabelCommands

```

A real example is:

```

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

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

\StartBabelCommands{austrian}{date}
  \SetString\monthinname{J\{a}nner}

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

\StartBabelCommands{german,austrian}{date}
  \SetString\monthiiiname{Februar}
  \SetString\monthiiiname{M\{a}rz}
  \SetString\monthivname{April}
  \SetString\monthvname{Mai}
  \SetString\monthviname{Juni}
  \SetString\monthviiname{Juli}
  \SetString\monthviiname{August}
  \SetString\monthixname{September}
  \SetString\monthxiname{Oktober}
  \SetString\monthxiiname{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 $\date \langle language \rangle$ exists).

³⁰In future releases further categories may be added.

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.³¹

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

`\AfterBabelCommands` `{⟨code⟩}`

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

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

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

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

`\SetStringLoop` `{⟨macro-name⟩}{⟨string-list⟩}`

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

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

#1 is replaced by the roman numeral.

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

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

³¹This replaces in 3.9g a short-lived `\UseStrings` which has been removed because it did not work.

```

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

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

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

New 3.9g Case mapping serves in 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"11F}{2}{\lccode"101}}

```

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

4 Changes

4.1 Changes in `babel` version 3.9

Most of changes in version 3.9 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 LaTeX package, which sets options and loads language styles.

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

hyphen.cfg is the file to be used when generating the formats to load hyphenation patterns. By default it also loads switch.def.

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

6 locale directory

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

Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and

polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.31>>
2 <<date=2019/05/04>>
```

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` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³². These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

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

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

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

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

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

³²This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

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

```
59 \def\bbl@ifblank#1{%
60 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
61 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
```

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

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

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

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

`\bbl@replace`

```
82 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
83 \toks@{}}
84 \def\bbl@replace@aux##1#2##2#2{%
85 \ifx\bbl@nil##2%
86 \toks@\expandafter{\the\toks@##1}%
87 \else
88 \toks@\expandafter{\the\toks@##1#3}%
89 \bbl@afterfi
90 \bbl@replace@aux##2#2%
91 \fi}%
92 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
93 \edef#1{\the\toks@}}
```

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

7.1 Multiple languages

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

allocating a counter. The following block is used in `switch.def` and `hyphen.cfg`; the latter may seem redundant, but remember `babel` doesn't require loading `switch.def` in the format.

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

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

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

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

```
147 <<*Define core switching macros>> ≡
148 \ifx\newlanguage\undefined
149   \csname newcount\endcsname\last@language
150   \def\addlanguage#1{%
151     \global\advance\last@language\ne
152     \ifnum\last@language<\ccclvi
153       \else
154         \errmessage{No room for a new \string\language!}%
155       \fi
156       \global\chardef#1\last@language
157       \wlog{\string#1 = \string\language\the\last@language}}
158   \else
159     \countdef\last@language=19
160     \def\addlanguage{\alloc@9\language\chardef\ccclvi}
161   \fi
162 <</Define core switching macros>>
```

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

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

8 The Package File (L^AT_EX, `babel.sty`)

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

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

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

8.1 base

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

```

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

```

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

```

179 \ifx\bbl@languages\undefined\else
180   \begingroup
181     \catcode\^^I=12
182     \@ifpackagewith{babel}{showlanguages}{%
183       \begingroup
184         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
185         \wlog{<*languages>}%
186         \bbl@languages
187         \wlog{</languages>}%
188       \endgroup}{%
189     \endgroup
190     \def\bbl@elt#1#2#3#4{%
191       \ifnum#2=\z@
192         \gdef\bbl@nulllanguage{#1}%
193         \def\bbl@elt##1##2##3##4{}%
194       \fi}%
195     \bbl@languages
196 \fi
197 \ifodd\bbl@engine
198   \def\bbl@activate@preotf{%
199     \let\bbl@activate@preotf\relax % only once
200     \directlua{
201       Babel = Babel or {}
202       %
203       function Babel.pre_otfload_v(head)
204         if Babel.numbers and Babel.digits_mapped then
205           head = Babel.numbers(head)

```

```

206     end
207     if Babel.bidi_enabled then
208         head = Babel.bidi(head, false, dir)
209     end
210     return head
211 end
212 %
213 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
214     if Babel.numbers and Babel.digits_mapped then
215         head = Babel.numbers(head)
216     end
217     if Babel.fixboxdirs then          % Temporary!
218         head = Babel.fixboxdirs(head)
219     end
220     if Babel.bidi_enabled then
221         head = Babel.bidi(head, false, dir)
222     end
223     return head
224 end
225 %
226 luatexbase.add_to_callback('pre_linebreak_filter',
227     Babel.pre_otfload_v,
228     'Babel.pre_otfload_v',
229     luatexbase.priority_in_callback('pre_linebreak_filter',
230         'luaotfload.node_processor') or nil)
231 %
232 luatexbase.add_to_callback('hpack_filter',
233     Babel.pre_otfload_h,
234     'Babel.pre_otfload_h',
235     luatexbase.priority_in_callback('hpack_filter',
236         'luaotfload.node_processor') or nil)
237 }}
238 \let\bbl@tempa\relax
239 \@ifpackagewith{babel}{bidi=basic}%
240     {\def\bbl@tempa{basic}}%
241     {\@ifpackagewith{babel}{bidi=basic-r}%
242         {\def\bbl@tempa{basic-r}}%
243     {}
244 \ifx\bbl@tempa\relax\else
245     \let\bbl@beforeforeign\leavevmode
246     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
247     \RequirePackage{luatexbase}%
248     \directlua{
249         require('babel-data-bidi.lua')
250         require('babel-bidi-\bbl@tempa.lua')
251     }
252     \bbl@activate@preotf
253 \fi
254 \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.

```

255 \bbl@trace{Defining option 'base'}
256 \@ifpackagewith{babel}{base}{%
257     \ifx\directlua\undefined
258         \DeclareOption*{\bbl@patterns{\CurrentOption}}%
259     \else
260         \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
261     \fi

```

```

262 \DeclareOption{base}{}%
263 \DeclareOption{showlanguages}{}%
264 \ProcessOptions
265 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
266 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
267 \global\let@ifl@ter@\@ifl@ter
268 \def\@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\@ifl@ter@}%
269 \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.

```

270 \bbl@trace{key=value and another general options}
271 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
272 \def\bbl@tempb#1.#2{%
273   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
274 \def\bbl@tempd#1.#2@\nnil{%
275   \ifx\@empty#2%
276     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
277   \else
278     \in{=}{#1}\ifin@
279     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
280   \else
281     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
282   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
283   \fi
284 \fi}
285 \let\bbl@tempc\@empty
286 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty@\nnil}
287 \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.

```

288 \DeclareOption{KeepShorthandsActive}{}
289 \DeclareOption{activeacute}{}
290 \DeclareOption{activegrave}{}
291 \DeclareOption{debug}{}
292 \DeclareOption{noconfigs}{}
293 \DeclareOption{showlanguages}{}
294 \DeclareOption{silent}{}
295 \DeclareOption{mono}{}
296 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
297 <<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.

```

298 \let\bbl@opt@shorthands@\nnil
299 \let\bbl@opt@config@\nnil

```

```

300 \let\bbl@opt@main\@nnil
301 \let\bbl@opt@headfoot\@nnil
302 \let\bbl@opt@layout\@nnil

```

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

```

303 \def\bbl@tempa#1=#2\bbl@tempa{%
304   \bbl@csarg\ifx{opt@#1}\@nnil
305     \bbl@csarg\edef{opt@#1}{#2}%
306   \else
307     \bbl@error{%
308       Bad option `#1=#2'. Either you have misspelled the\\%
309       key or there is a previous setting of `#1'}{%
310       Valid keys are `shorthands', `config', `strings', `main',\\%
311       `headfoot', `safe', `math', among others.}
312   \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.

```

313 \let\bbl@language@opts\@empty
314 \DeclareOption*{%
315   \bbl@xin@{\string=}{\CurrentOption}%
316   \ifin@
317     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
318   \else
319     \bbl@add@list\bbl@language@opts{\CurrentOption}%
320   \fi}

```

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

```

321 \ProcessOptions*

```

8.3 Conditional loading of shorthands

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

```

322 \bbl@trace{Conditional loading of shorthands}
323 \def\bbl@sh@string#1{%
324   \ifx#1\@empty\else
325     \ifx#1t\string~%
326     \else\ifx#1c\string,%
327     \else\string#1%
328     \fi\fi
329   \expandafter\bbl@sh@string
330   \fi}
331 \ifx\bbl@opt@shorthands\@nnil
332   \def\bbl@ifshorthand#1#2#3{#2}%
333 \else\ifx\bbl@opt@shorthands\@empty
334   \def\bbl@ifshorthand#1#2#3{#3}%
335 \else

```

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

```

336 \def\bbl@ifshorthand#1{%
337   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
338   \ifin@

```

```

339     \expandafter\@firstoftwo
340     \else
341     \expandafter\@secondoftwo
342     \fi}

```

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

```

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

```

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

```

345 \bbl@ifshorthand{'}%
346     {\PassOptionsToPackage{activeacute}{babel}}{}
347 \bbl@ifshorthand{`}%
348     {\PassOptionsToPackage{activegrave}{babel}}{}
349 \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.

```

350 \ifx\bbl@opt@headfoot\@nnil\else
351     \g@addto@macro\@resetactivechars{%
352         \set@typeset@protect
353         \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
354         \let\protect\noexpand}
355 \fi

```

For the option `safe` we use a different approach – `\bbl@opt@safe` says which macros are redefined (B for hibs and R for refs). By default, both are set.

```

356 \ifx\bbl@opt@safe\@undefined
357     \def\bbl@opt@safe{BR}
358 \fi
359 \ifx\bbl@opt@main\@nnil\else
360     \edef\bbl@language@opts{%
361         \ifx\bbl@language@opts\@empty\else\bbl@language@opts, \fi
362         \bbl@opt@main}
363 \fi

```

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

```

364 \bbl@trace{Defining IfBabelLayout}
365 \ifx\bbl@opt@layout\@nnil
366     \newcommand\IfBabelLayout[3]{#3}%
367 \else
368     \newcommand\IfBabelLayout[1]{%
369         \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
370         \ifin@
371             \expandafter\@firstoftwo
372         \else
373             \expandafter\@secondoftwo
374         \fi}
375 \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).

```

376 \bbl@trace{Language options}
377 \let\bbl@afterlang\relax
378 \let\BabelModifiers\relax
379 \let\bbl@loaded@empty
380 \def\bbl@load@language#1{%
381   \InputIfFileExists{#1.ldf}%
382   {\edef\bbl@loaded{\CurrentOption
383     \ifx\bbl@loaded\empty\else,\bbl@loaded\fi}%
384     \expandafter\let\expandafter\bbl@afterlang
385       \csname\CurrentOption.ldf-h@k\endcsname
386     \expandafter\let\expandafter\BabelModifiers
387       \csname bbl@mod@\CurrentOption\endcsname}%
388   {\bbl@error{%
389     Unknown option '\CurrentOption'. Either you misspelled it\\%
390     or the language definition file \CurrentOption.ldf was not found}{%
391     Valid options are: shorthands=, KeepShorthandsActive,\\%
392     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
393     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

394 \def\bbl@try@load@lang#1#2#3{%
395   \IfFileExists{\CurrentOption.ldf}%
396   {\bbl@load@language{\CurrentOption}}%
397   {#1\bbl@load@language{#2}#3}}
398 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}}
399 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}}
400 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}}
401 \DeclareOption{hebrew}{%
402   \input{r\babel.def}%
403   \bbl@load@language{hebrew}}
404 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}
405 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}
406 \DeclareOption{nyorsk}{\bbl@try@load@lang{}{norsk}}
407 \DeclareOption{polutonikogreek}{%
408   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
409 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}}
410 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}
411 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}}
412 \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.

```

413 \ifx\bbl@opt@config@nil
414   \@ifpackagewith{babel}{noconfigs}}%
415   {\InputIfFileExists{bblopts.cfg}%
416     {\typeout{*****^^J%
417       * Local config file bblopts.cfg used^^J%
418       *}}%
419     {}}%
420 \else
421   \InputIfFileExists{\bbl@opt@config.cfg}%
422   {\typeout{*****^^J%
423     * Local config file \bbl@opt@config.cfg used^^J%

```

```

424         *}}%
425     {\bbl@error{%
426         Local config file '\bbl@opt@config.cfg' not found}{%
427         Perhaps you misspelled it.}}%
428 \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.

```

429 \bbl@for\bbl@tempa\bbl@language@opts{%
430     \bbl@ifunset{ds@\bbl@tempa}%
431     {\edef\bbl@tempb{%
432         \noexpand\DeclareOption
433         {\bbl@tempa}%
434         {\noexpand\bbl@load@language{\bbl@tempa}}}%
435     \bbl@tempb}%
436     \@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.

```

437 \bbl@foreach\@classoptionslist{%
438     \bbl@ifunset{ds@#1}%
439     {\IfFileExists{#1.ldf}%
440     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
441     {}}%
442     {}}

```

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

```

443 \ifx\bbl@opt@main\@nnil\else
444     \expandafter
445     \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
446     \DeclareOption{\bbl@opt@main}{}
447 \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):

```

448 \def\AfterBabelLanguage#1{%
449     \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
450 \DeclareOption*{}
451 \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`.

```

452 \ifx\bbl@opt@main\@nnil
453     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
454     \let\bbl@tempc\@empty
455     \bbl@for\bbl@tempb\bbl@tempa{%
456         \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
457         \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
458     \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}

```



```

459 \expandafter\bbl@tempa\bbl@loaded,\@nnil
460 \ifx\bbl@tempb\bbl@tempc\else
461   \bbl@warning{%
462     Last declared language option is `bbl@tempc',\%
463     but the last processed one was `bbl@tempb'.\%
464     The main language cannot be set as both a global\%
465     and a package option. Use `main=bbl@tempc' as\%
466     option. Reported}%
467 \fi
468 \else
469 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
470 \ExecuteOptions{\bbl@opt@main}
471 \DeclareOption*{}
472 \ProcessOptions*
473 \fi
474 \def\AfterBabelLanguage{%
475   \bbl@error
476   {Too late for \string\AfterBabelLanguage}%
477   {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.

```

478 \ifx\bbl@main@language\undefined
479   \bbl@info{%
480     You haven't specified a language. I'll use 'nil'\%
481     as the main language. Reported}
482   \bbl@load@language{nil}
483 \fi
484 \</package>
485 \<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

```

486 \ifx\ldf@quit\undefined
487 \else
488   \expandafter\endinput
489 \fi
490 \<<Make sure ProvidesFile is defined>>

```

```

491 \ProvidesFile{babel.def}[<<date>> <<version>> Babel common definitions]
492 <<Load macros for plain if not LaTeX>>

```

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

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

```

493 \ifx\bbl@ifshorthand\@undefined
494   \let\bbl@opt@shorthands\@nnil
495   \def\bbl@ifshorthand#1#2#3{#2}%
496   \let\bbl@language@opts\@empty
497   \ifx\babeloptionstrings\@undefined
498     \let\bbl@opt@strings\@nnil
499   \else
500     \let\bbl@opt@strings\babeloptionstrings
501   \fi
502   \def\BabelStringsDefault{generic}
503   \def\bbl@tempa{normal}
504   \ifx\babeloptionmath\bbl@tempa
505     \def\bbl@mathnormal{\noexpand\textormath}
506   \fi
507   \def\AfterBabelLanguage#1#2{}
508   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
509   \let\bbl@afterlang\relax
510   \def\bbl@opt@safe{BR}
511   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
512   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
513 \fi

```

And continue.

```

514 \ifx\bbl@switchflag\@undefined % Prevent double input
515   \let\bbl@switchflag\relax
516   \input switch.def\relax
517 \fi
518 \bbl@trace{Compatibility with language.def}
519 \ifx\bbl@languages\@undefined
520   \ifx\directlua\@undefined
521     \openin1 = language.def
522     \ifeof1
523       \closein1
524       \message{I couldn't find the file language.def}
525     \else
526       \closein1
527       \begingroup
528         \def\addlanguage#1#2#3#4#5{%
529           \expandafter\ifx\csname lang@#1\endcsname\relax\else
530             \global\expandafter\let\csname l@#1\endcsname
531               \csname lang@#1\endcsname
532           \fi}%
533         \def\uselanguage#1{%
534           \input language.def
535         \endgroup
536       \fi
537     \fi
538   \chardef\l@english\z@
539 \fi
540 <<Load patterns in luatex>>

```

541 <<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.

```
542 \def\addto#1#2{%
543   \ifx#1\undefined
544     \def#1{#2}%
545   \else
546     \ifx#1\relax
547       \def#1{#2}%
548     \else
549       {\toks@\expandafter{#1#2}%
550        \xdef#1{\the\toks@}}%
551   \fi
552 }
```

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.

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

`\bbl@redefine` To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don’t want to redefine the L^AT_EX 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@...`

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

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

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

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

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

`\bbl@redefineroast` 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`.

```

567 \def\bbl@redefineroobust#1{%
568   \edef\bbl@tempa{\bbl@stripslash#1}%
569   \bbl@ifunset{\bbl@tempa\space}%
570     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
571       \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
572     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
573   \@namedef{\bbl@tempa\space}}

```

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

```
574 \@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 vety little to catch errors, but it is intended for developpers, after all. `\bbl@usehooks` is the commands used by babel to execute hooks defined for an event.

```

575 \bbl@trace{Hooks}
576 \def\AddBabelHook#1#2{%
577   \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}}%
578   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
579   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
580   \bbl@ifunset{bbl@ev@#1@#2}%
581     {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}}%
582     \bbl@csarg\newcommand}%
583     {\bbl@csarg\let{ev@#1@#2}\relax
584     \bbl@csarg\newcommand}%
585     {ev@#1@#2}[\bbl@tempb]}
586 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
587 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
588 \def\bbl@usehooks#1#2{%
589   \def\bbl@elt##1{%
590     \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}%
591   \@nameuse{bbl@ev@#1}}

```

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

```

592 \def\bbl@evargs{,% <- don't delete this comma
593   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
594   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
595   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
596   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0}

```

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

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

```

597 \bbl@trace{Defining babelensure}
598 \newcommand\babelensure[2][]{% TODO - revise test files
599   \AddBabelHook{babel-ensure}{afterextras}{%
600     \ifcase\bbl@select@type
601       \nameuse{bbl@e@\languagename}%
602     \fi}%
603 \begingroup
604   \let\bbl@ens@include\@empty
605   \let\bbl@ens@exclude\@empty
606   \def\bbl@ens@fontenc{\relax}%
607   \def\bbl@tempb##1{%
608     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
609   \edef\bbl@tempa{\bbl@tempb##1\@empty}%
610   \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
611   \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
612   \def\bbl@tempc{\bbl@ensure}%
613   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
614     \expandafter{\bbl@ens@include}}%
615   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
616     \expandafter{\bbl@ens@exclude}}%
617   \toks@\expandafter{\bbl@tempc}%
618   \bbl@exp{%
619 \endgroup
620 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
621 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
622 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
623   \ifx##1\@empty\else
624     \in@{##1}{#2}%
625   \ifin@ \else
626     \bbl@ifunset{bbl@ensure@\languagename}%
627     {\bbl@exp{%
628       \\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
629         \\foreignlanguage{\languagename}%
630         {\ifx\relax#3\else
631           \\fontencoding{#3}\\selectfont
632           \fi
633           #####1}}}}%
634     }%
635     \toks@\expandafter{##1}%
636   \edef##1{%
637     \bbl@csarg\noexpand{ensure@\languagename}%
638     {\the\toks@}}%
639   \fi
640   \expandafter\bbl@tempb
641 \fi}%
642 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
643 \def\bbl@tempa##1{% elt for include list
644   \ifx##1\@empty\else
645     \bbl@csarg\in@{ensure@\languagename}\expandafter}\expandafter{##1}%
646   \ifin@ \else
647     \bbl@tempb##1\@empty
648   \fi
649   \expandafter\bbl@tempa
650 \fi}%
651 \bbl@tempa#1\@empty}
652 \def\bbl@captionslist{%
653 \prefacename\refname\abstractname\bibname\chaptername\appendixname
654 \contentsname\listfigurename\listtablename\indexname\figurename
655 \tablename\partname\enc1name\ccname\headtoname\pagename\seename

```

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 `@`-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`.

```

657 \bbl@trace{Macros for setting language files up}
658 \def\bbl@ldfinit{%
659   \let\bbl@screset\@empty
660   \let\BabelStrings\bbl@opt@string
661   \let\BabelOptions\@empty
662   \let\BabelLanguages\relax
663   \ifx\originalTeX\@undefined
664     \let\originalTeX\@empty
665   \else
666     \originalTeX
667   \fi}
668 \def\LdfInit#1#2{%
669   \chardef\atcatcode=\catcode`\@
670   \catcode`\@=11\relax
671   \chardef\eqcatcode=\catcode`\=
672   \catcode`\==12\relax
673   \expandafter\if\expandafter\@backslashchar
674     \expandafter\@car\string#2\@nil
675     \ifx#2\@undefined\else
676       \ldf@quit{#1}%
677     \fi
678   \else
679     \expandafter\ifx\csname#2\endcsname\relax\else
680       \ldf@quit{#1}%
681     \fi
682   \fi
683   \bbl@ldfinit}

```

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

```

684 \def\ldf@quit#1{%
685   \expandafter\main@language\expandafter{#1}%
686   \catcode`\@=\atcatcode \let\atcatcode\relax

```

```

687 \catcode`\==\eqcatcode \let\eqcatcode\relax
688 \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.

```

689 \def\bbl@afterldf#1{%
690 \bbl@afterlang
691 \let\bbl@afterlang\relax
692 \let\BabelModifiers\relax
693 \let\bbl@screset\relax}%
694 \def\ldf@finish#1{%
695 \loadlocalcfg{#1}%
696 \bbl@afterldf{#1}%
697 \expandafter\main@language\expandafter{#1}%
698 \catcode`\@=\atcatcode \let\atcatcode\relax
699 \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 .

```

700 \@onlypreamble\LdfInit
701 \@onlypreamble\ldf@quit
702 \@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.

`\bbl@main@language`

```

703 \def\main@language#1{%
704 \def\bbl@main@language{#1}%
705 \let\language\bbl@main@language
706 \bbl@id@assign
707 \chardef\localeid\@nameuse{\bbl@id@\language}%
708 \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`.

```

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

```

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

```

712 \def\select@language@x#1{%
713 \ifcase\bbl@select@type
714 \bbl@ifsamestring\language{#1}{\select@language{#1}}%
715 \else
716 \select@language{#1}%
717 \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.

```

718 \bbl@trace{Shorhands}
719 \def\bbl@add@special#1{% 1:a macro like ", \?, etc.
720 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
721 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
722 \ifx\nfss@catcodes\undefined\else % TODO - same for above
723 \begingroup
724 \catcode`#1\active
725 \nfss@catcodes
726 \ifnum\catcode`#1=\active
727 \endgroup
728 \bbl@add\nfss@catcodes{\@makeother#1}%
729 \else
730 \endgroup
731 \fi
732 \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.

```

733 \def\bbl@remove@special#1{%
734 \begingroup
735 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
736 \else\noexpand##1\noexpand##2\fi}%
737 \def\do{\x\do}%
738 \def\@makeother{\x\@makeother}%
739 \edef\x{\endgroup
740 \def\noexpand\dospecials{\dospecials}%
741 \expandafter\ifx\csname @sanitize\endcsname\relax\else
742 \def\noexpand\@sanitize{\@sanitize}%
743 \fi}%
744 \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` $\langle char \rangle$ to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char` $\langle char \rangle$ by default ($\langle char \rangle$ being the character to be made active). Later its definition can be changed to expand to `\active@char` $\langle char \rangle$ by calling `\bbl@activate` $\langle char \rangle$.

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).

```

745 \def\bbl@active@def#1#2#3#4{%

```



```

746 \@namedef{#3#1}{%
747   \expandafter\ifx\csname#2@sh@#1@endcsname\relax
748   \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
749   \else
750   \bbl@afterfi\csname#2@sh@#1@endcsname
751   \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.

```

752 \long\@namedef{#3@arg#1}##1{%
753   \expandafter\ifx\csname#2@sh@#1@string##1@endcsname\relax
754   \bbl@afterelse\csname#4#1@endcsname##1%
755   \else
756   \bbl@afterfi\csname#2@sh@#1@string##1@endcsname
757   \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.

```

758 \def\initiate@active@char#1{%
759   \bbl@ifunset{active@char\string#1}%
760   {\bbl@withactive
761     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
762   {}}

```

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`).

```

763 \def\@initiate@active@char#1#2#3{%
764   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
765   \ifx#1\@undefined
766     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
767   \else
768     \bbl@csarg\let{oridef@#2}#1%
769     \bbl@csarg\edef{oridef@#2}{%
770       \let\noexpand#1%
771       \expandafter\noexpand\csname bbl@oridef@#2@endcsname}%
772   \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*").

```

773 \ifx#1#3\relax
774   \expandafter\let\csname normal@char#2@endcsname#3%
775 \else
776   \bbl@info{Making #2 an active character}%
777   \ifnum\mathcode`#2="8000
778     \@namedef{normal@char#2}{%
779       \textormath{#3}{\csname bbl@oridef@#2@endcsname}}%
780   \else
781     \@namedef{normal@char#2}{#3}%
782   \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).

```

783 \bbl@restoreactive{#2}%
784 \AtBeginDocument{%
785   \catcode`#2\active
786   \if@filesw
787     \immediate\write\@mainaux{\catcode`\string#2\active}%
788   \fi}%
789 \expandafter\bbl@add@special\csname#2\endcsname
790 \catcode`#2\active
791 \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⟩`).

```

792 \let\bbl@tempa\@firstoftwo
793 \if\string^#2%
794   \def\bbl@tempa{\noexpand\textormath}%
795 \else
796   \ifx\bbl@mathnormal\@undefined\else
797     \let\bbl@tempa\bbl@mathnormal
798   \fi
799 \fi
800 \expandafter\edef\csname active@char#2\endcsname{%
801   \bbl@tempa
802     {\noexpand\if@safe@actives
803       \noexpand\expandafter
804         \expandafter\noexpand\csname normal@char#2\endcsname
805       \noexpand\else
806         \noexpand\expandafter
807         \expandafter\noexpand\csname bbl@doactive#2\endcsname
808       \noexpand\fi}%
809   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
810 \bbl@csarg\edef{doactive#2}{%
811   \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

$$\backslash\text{active@prefix}\langle\text{char}\rangle\backslash\text{normal@char}\langle\text{char}\rangle$$

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

```

812 \bbl@csarg\edef{active@#2}{%
813   \noexpand\active@prefix\noexpand#1%
814   \expandafter\noexpand\csname active@char#2\endcsname}%
815 \bbl@csarg\edef{normal@#2}{%
816   \noexpand\active@prefix\noexpand#1%
817   \expandafter\noexpand\csname normal@char#2\endcsname}%
818 \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.

```

819 \bbl@active@def#2\user@group{user@active}{language@active}%
820 \bbl@active@def#2\language@group{language@active}{system@active}%
821 \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.

```

822 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
823   {\expandafter\noexpand\csname normal@char#2\endcsname}%
824 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
825   {\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.

```

826 \if\string'#2%
827   \let\prim@s\bbl@prim@s
828   \let\active@math@prime#1%
829 \fi
830 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

831 <<(*More package options)>> ≡
832 \DeclareOption{math=active}{}
833 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}
834 <</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.

```

835 \@ifpackagewith{babel}{KeepShorthandsActive}%
836   {\let\bbl@restoreactive@gobble}%
837   {\def\bbl@restoreactive#1{%
838     \bbl@exp{%
839       \\AfterBabelLanguage\\CurrentOption
840       {\catcode`#1=\the\catcode`#1\relax}%
841       \\AtEndOfPackage
842       {\catcode`#1=\the\catcode`#1\relax}}}%
843   \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.

```

844 \def\bbl@sh@select#1#2{%
845   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
846     \bbl@afterelse\bbl@scndcs
847   \else
848     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
849   \fi}

```

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

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

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

```
853   \ifx\protect\@unexpandable@protect
854   \noexpand#1%
855   \else
856   \protect#1%
857   \fi
858   \expandafter\@gobble
859 \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*).

```
860 \newif\if@safe@actives
861 \@safe@activesfalse
```

`\bbl@restore@actives` When the output routine kicks in while the active characters were made “safe” this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them “unsafe” again.

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

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

```
863 \def\bbl@activate#1{%
864   \bbl@withactive{\expandafter\let\expandafter}#1%
865   \csname bbl@active@\string#1\endcsname}
866 \def\bbl@deactivate#1{%
867   \bbl@withactive{\expandafter\let\expandafter}#1%
868   \csname bbl@normal@\string#1\endcsname}
```

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

```
869 \def\bbl@firstcs#1#2{\csname#1\endcsname}
870 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

`\declare@shorthand` The command `\declare@shorthand` is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. `~` or `"a`;
3. the code to be executed when the shorthand is encountered.

```

871 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
872 \def\@decl@short#1#2#3\@nil#4{%
873   \def\bb1@tempa{#3}%
874   \ifx\bb1@tempa\@empty
875     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@scndcs
876     \bb1@ifunset{#1@sh@\string#2@}\{}%
877     {\def\bb1@tempa{#4}%
878       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bb1@tempa
879       \else
880         \bb1@info
881         {Redefining #1 shorthand \string#2\%
882         in language \CurrentOption}%
883       \fi}%
884     \@namedef{#1@sh@\string#2@}{#4}%
885   \else
886     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@firstcs
887     \bb1@ifunset{#1@sh@\string#2@\string#3@}\{}%
888     {\def\bb1@tempa{#4}%
889       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bb1@tempa
890       \else
891         \bb1@info
892         {Redefining #1 shorthand \string#2\string#3\%
893         in language \CurrentOption}%
894       \fi}%
895     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
896   \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.

```

897 \def\textormath{%
898   \ifmmode
899     \expandafter\@secondoftwo
900   \else
901     \expandafter\@firstoftwo
902   \fi}

```

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

```

903 \def\user@group{user}
904 \def\language@group{english}
905 \def\system@group{system}

```

`\usesshorthands` 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.

```

906 \def\usesshorthands{%
907   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
908 \def\bb1@usesh@s#1{%
909   \bb1@usesh@x
910   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
911   {#1}}
912 \def\bb1@usesh@x#1#2{%
913   \bb1@ifshorthand{#2}%

```

```

914 {\def\user@group{user}%
915 \initiate@active@char{#2}%
916 #1%
917 \bbl@activate{#2}}%
918 {\bbl@error
919 {Cannot declare a shorthand turned off (\string#2)}
920 {Sorry, but you cannot use shorthands which have been\%
921 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.

```

922 \def\user@language@group{user@\language@group}
923 \def\bbl@set@user@generic#1#2{%
924 \bbl@ifunset{user@generic@active#1}%
925 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
926 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
927 \expandafter\edef\csname#2@sh@#1@@\endcsname{%
928 \expandafter\noexpand\csname normal@char#1\endcsname}%
929 \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
930 \expandafter\noexpand\csname user@active#1\endcsname}}%
931 \@empty}
932 \newcommand\defineshorthand[3][user]{%
933 \edef\bbl@tempa{\zap@space#1 \@empty}%
934 \bbl@for\bbl@tempb\bbl@tempa{%
935 \if*\expandafter\@car\bbl@tempb\@nil
936 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
937 \@expandtwoargs
938 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
939 \fi
940 \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.

```

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

```

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

```

942 \def\aliasshorthand#1#2{%
943 \bbl@ifshorthand{#2}%
944 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
945 \ifx\document\@notprerr
946 \@notshorthand{#2}%
947 \else
948 \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`.

```

949 \expandafter\let\csname active@char\string#2\expandafter\endcsname
950 \csname active@char\string#1\endcsname
951 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
952 \csname normal@char\string#1\endcsname
953 \bbl@activate{#2}%
954 \fi

```

```

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

```

\@notshorthand

```

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

```

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

\shorthandoff

```

967 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
968 \DeclareRobustCommand*\shorthandoff{%
969   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
970 \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.

```

971 \def\bbl@switch@sh#1#2{%
972   \ifx#2\@nnil\else
973     \bbl@ifunset{\bbl@active@\string#2}%
974     {\bbl@error
975       {I cannot switch `\'string#2' on or off--not a shorthand}%
976       {This character is not a shorthand. Maybe you made\\%
977         a typing mistake? I will ignore your instruction}}}%
978     {\ifcase#1%
979       \catcode`#212\relax
980     \or
981       \catcode`#2\active
982     \or
983       \csname bbl@oricat@\string#2\endcsname
984       \csname bbl@oridef@\string#2\endcsname
985     \fi}%
986     \bbl@afterfi\bbl@switch@sh#1%
987   \fi}

```

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

```

988 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
989 \def\bbl@putsh#1{%
990   \bbl@ifunset{\bbl@active@\string#1}%
991   {\bbl@putsh@i#1\@empty\@nnil}%
992   {\csname bbl@active@\string#1\endcsname}}
993 \def\bbl@putsh@i#1#2\@nnil{%

```

```

994 \csname\languagename @sh@\string#1@%
995 \ifx\@empty#2\else\string#2@\fi\endcsname}
996 \ifx\bbbl@opt@shorthands\@nnil\else
997 \let\bbbl@s@initiate@active@char\initiate@active@char
998 \def\initiate@active@char#1{%
999 \bbbl@ifshorthand{#1}{\bbbl@s@initiate@active@char{#1}}{}}
1000 \let\bbbl@s@switch@sh\bbbl@switch@sh
1001 \def\bbbl@switch@sh#1#2{%
1002 \ifx#2\@nnil\else
1003 \bbbl@afterfi
1004 \bbbl@ifshorthand{#2}{\bbbl@s@switch@sh#1{#2}}{\bbbl@switch@sh#1}%
1005 \fi}
1006 \let\bbbl@s@activate\bbbl@activate
1007 \def\bbbl@activate#1{%
1008 \bbbl@ifshorthand{#1}{\bbbl@s@activate{#1}}{}}
1009 \let\bbbl@s@deactivate\bbbl@deactivate
1010 \def\bbbl@deactivate#1{%
1011 \bbbl@ifshorthand{#1}{\bbbl@s@deactivate{#1}}{}}
1012 \fi

```

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

```

1013 \newcommand\ifbabelshorthand[3]{\bbbl@ifunset{bbbl@active@\string#1}{#3}{#2}}

```

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

```

1014 \def\bbbl@prim@s{%
1015 \prime\futurelet\@let@token\bbbl@pr@m@s}
1016 \def\bbbl@if@primes#1#2{%
1017 \ifx#1\@let@token
1018 \expandafter\@firstoftwo
1019 \else\ifx#2\@let@token
1020 \bbbl@afterelse\expandafter\@firstoftwo
1021 \else
1022 \bbbl@afterfi\expandafter\@secondoftwo
1023 \fi\fi}
1024 \begingroup
1025 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^^
1026 \catcode`\'=12 \catcode`\"=\active \lccode`\"='\ '
1027 \lowercase{%
1028 \gdef\bbbl@pr@m@s{%
1029 \bbbl@if@primes" '%
1030 \pr@@@s
1031 {\bbbl@if@primes*\^*\pr@@@t\egroup}}
1032 \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).

```

1033 \initiate@active@char{~}
1034 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1035 \bbbl@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.

```
1036 \expandafter\def\csname OT1dqpos\endcsname{127}
1037 \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

```
1038 \ifx\f@encoding\undefined
1039 \def\f@encoding{OT1}
1040 \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.

```
1041 \bbl@trace{Language attributes}
1042 \newcommand\languageattribute[2]{%
1043 \def\bbl@tempc{#1}%
1044 \bbl@fixname\bbl@tempc
1045 \bbl@iflanguage\bbl@tempc{%
1046 \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.

```
1047 \ifx\bbl@known@attribs\undefined
1048 \in@false
1049 \else
```

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

```
1050 \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,}%
1051 \fi
```

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

```
1052 \ifin@
1053 \bbl@warning{%
1054 You have more than once selected the attribute '##1'\%
1055 for language #1. Reported}%
1056 \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.

```
1057 \bbl@exp{%
1058 \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}%
1059 \edef\bbl@tempa{\bbl@tempc-##1}%
1060 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1061 {\csname\bbl@tempc @attr##1\endcsname}%
1062 {\@attrerr{\bbl@tempc}{##1}}%
1063 \fi}}
```

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

```
1064 \@onlypreamble\languageattribute
```

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

```
1065 \newcommand*{\@attrerr}[2]{%
1066   \bbl@error
1067   {The attribute #2 is unknown for language #1.}%
1068   {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}`.

```
1069 \def\bbl@declare@ttribute#1#2#3{%
1070   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1071   \ifin@
1072     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1073   \fi
1074   \bbl@add@list\bbl@attributes{#1-#2}%
1075   \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.

```
1076 \def\bbl@ifattributeset#1#2#3#4{%
    First we need to find out if any attributes were set; if not we're done.
1077   \ifx\bbl@known@attribs\undefined
1078     \in@false
1079   \else
```

The we need to check the list of known attributes.

```
1080   \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1081   \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`'.

```
1082   \ifin@
1083     \bbl@afterelse#3%
1084   \else
1085     \bbl@afterfi#4%
1086   \fi
1087 }
```

`\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.

```
1088 \def\bbl@ifknown@ttrib#1#2{%
    We first assume the attribute is unknown.
1089   \let\bbl@tempa\@secondoftwo
    Then we loop over the list of known attributes, trying to find a match.
1090   \bbl@loopx\bbl@tempb{#2}{%
1091     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1092     \ifin@
```

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

```
1093 \let\bbl@tempa\@firstoftwo
1094 \else
1095 \fi}%
```

Finally we execute `\bbl@tempa`.

```
1096 \bbl@tempa
1097 }
```

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

```
1098 \def\bbl@clear@ttribs{%
1099 \ifx\bbl@attributes\undefined\else
1100 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1101 \expandafter\bbl@clear@ttrib\bbl@tempa.
1102 }%
1103 \let\bbl@attributes\undefined
1104 \fi}
1105 \def\bbl@clear@ttrib#1-#2.{%
1106 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1107 \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`

```
1108 \bbl@trace{Macros for saving definitions}
1109 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1110 \newcount\babel@savecnt
1111 \babel@beginsave
```

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX`³³. 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.

```
1112 \def\babel@save#1{%
1113 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1114 \toks@\expandafter{\originalTeX\let#1=}%
1115 \bbl@exp{%
1116 \def\\\originalTeX{\the\toks@<babel@number\babel@savecnt>\relax}}%
1117 \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.

```
1118 \def\babel@savevariable#1{%
1119 \toks@\expandafter{\originalTeX #1}%
1120 \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

³³`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\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.

```
1121 \def\bbl@frenchspacing{%
1122   \ifnum\the\sfcode`\.=\@m
1123     \let\bbl@nonfrenchspacing\relax
1124   \else
1125     \frenchspacing
1126     \let\bbl@nonfrenchspacing\nonfrenchspacing
1127   \fi}
1128 \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.

```
1129 \bbl@trace{Short tags}
1130 \def\babeltags#1{%
1131   \edef\bbl@tempa{\zap@space#1 \@empty}%
1132   \def\bbl@tempb##1=##2\@{@%
1133     \edef\bbl@tempc{%
1134       \noexpand\newcommand
1135       \expandafter\noexpand\csname ##1\endcsname{%
1136         \noexpand\protect
1137         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1138       \noexpand\newcommand
1139       \expandafter\noexpand\csname text##1\endcsname{%
1140         \noexpand\foreignlanguage{##2}}
1141     \bbl@tempc}%
1142   \bbl@for\bbl@tempa\bbl@tempa{%
1143     \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.

```
1144 \bbl@trace{Hyphens}
1145 \@onlypreamble\babelhyphenation
1146 \AtEndOfPackage{%
1147   \newcommand\babelhyphenation[2][\@empty]{%
1148     \ifx\bbl@hyphenation@\relax
1149       \let\bbl@hyphenation@\@empty
1150     \fi
1151     \ifx\bbl@hyphlist\@empty\else
1152       \bbl@warning{%
1153         You must not intermingle \string\selectlanguage\space and\%
1154         \string\babelhyphenation\space or some exceptions will not\%
1155         be taken into account. Reported}%
1156     \fi
1157     \ifx\@empty#1%
1158       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1159     \else
1160       \bbl@vforeach{#1}{%
```

```

1161     \def\bbl@tempa{##1}%
1162     \bbl@fixname\bbl@tempa
1163     \bbl@iflanguage\bbl@tempa{%
1164         \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1165             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1166                 \@empty
1167                 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1168                 #2}}}%
1169     \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip Opt plus Opt`³⁴.

```

1170 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1171 \def\bbl@t@one{T1}
1172 \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`.

```

1173 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1174 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1175 \def\bbl@hyphen{%
1176     \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1177 \def\bbl@hyphen@i#1#2{%
1178     \bbl@ifunset{bbl@hy@#1#2\@empty}%
1179     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{-}{#2}}}%
1180     {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

```

1181 \def\bbl@usehyphen#1{%
1182     \leavevmode
1183     \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1184     \nobreak\hskip\z@skip}
1185 \def\bbl@@usehyphen#1{%
1186     \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1187 \def\bbl@hyphenchar{%
1188     \ifnum\hyphenchar\font=\m@ne
1189         \babelnullhyphen
1190     \else
1191         \char\hyphenchar\font
1192     \fi}

```

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

```

1193 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{-}}{}}
1194 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{-}}{}}
1195 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1196 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}

```

³⁴TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1197 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1198 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1199 \def\bbl@hy@repeat{%
1200   \bbl@usehyphen{%
1201     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1202 \def\bbl@hy@repeat{%
1203   \bbl@usehyphen{%
1204     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1205 \def\bbl@hy@empty{\hskip\z@skip}
1206 \def\bbl@hy@empty{\discretionary{}{}{}}

```

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

```

1207 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@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.

```

1208 \bbl@trace{Multiencoding strings}
1209 \def\bbl@tglobal#1{\global\let#1#1}
1210 \def\bbl@recatcode#1{%
1211   \@tempcnta="7F
1212   \def\bbl@tempa{%
1213     \ifnum\@tempcnta>"FF\else
1214       \catcode\@tempcnta=#1\relax
1215       \advance\@tempcnta\@ne
1216       \expandafter\bbl@tempa
1217     \fi}%
1218   \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).

```

1219 \@ifpackagewith{babel}{nocase}%
1220   {\let\bbl@patchuclc\relax}%
1221   {\def\bbl@patchuclc{%
1222     \global\let\bbl@patchuclc\relax
1223     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1224     \gdef\bbl@uclc##1{%
1225       \let\bbl@encoded\bbl@encoded@uclc
1226       \bbl@ifunset{\languagenome @bbl@uclc}% and resumes it
1227       {##1}%

```

```

1228     {\let\bbl@tempa##1\relax % Used by LANG@bbl@uc1c
1229     \csname\languagename @bbl@uc1c\endcsname}%
1230     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1231     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1232     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}%
1233 <<(*More package options)>> ≡
1234 \DeclareOption{nocase}{}
1235 <</More package options>>

```

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

```

1236 <<(*More package options)>> ≡
1237 \let\bbl@opt@strings\@nnil % accept strings=value
1238 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1239 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1240 \def\BabelStringsDefault{generic}
1241 <</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.

```

1242 \@onlypreamble\StartBabelCommands
1243 \def\StartBabelCommands{%
1244   \begingroup
1245   \bbl@recatcode{11}%
1246   <⟨Macros local to BabelCommands⟩>
1247   \def\bbl@provstring##1##2{%
1248     \providecommand##1{##2}%
1249     \bbl@tglobal##1}%
1250   \global\let\bbl@scafter\@empty
1251   \let\StartBabelCommands\bbl@startcmds
1252   \ifx\BabelLanguages\relax
1253     \let\BabelLanguages\CurrentOption
1254   \fi
1255   \begingroup
1256   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1257   \StartBabelCommands}
1258 \def\bbl@startcmds{%
1259   \ifx\bbl@screset\@nnil\else
1260     \bbl@usehooks{stopcommands}{}%
1261   \fi
1262   \endgroup
1263   \begingroup
1264   \@ifstar
1265     {\ifx\bbl@opt@strings\@nnil
1266       \let\bbl@opt@strings\BabelStringsDefault
1267     \fi
1268     \bbl@startcmds@i}%
1269   \bbl@startcmds@i}
1270 \def\bbl@startcmds@i#1#2{%
1271   \edef\bbl@L{\zap@space#1 \@empty}%
1272   \edef\bbl@G{\zap@space#2 \@empty}%
1273   \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.

```

1274 \newcommand\bb@startcmds@ii[1][\@empty]{%
1275 \let\SetString@gobbletwo
1276 \let\bb@stringdef@gobbletwo
1277 \let\AfterBabelCommands@gobble
1278 \ifx\@empty#1%
1279 \def\bb@sc@label{generic}%
1280 \def\bb@encstring##1##2{%
1281 \ProvideTextCommandDefault##1{##2}%
1282 \bb@tglobal##1%
1283 \expandafter\bb@tglobal\csname\string?\string##1\endcsname}%
1284 \let\bb@sctest\in@true
1285 \else
1286 \let\bb@sc@charset\space % <- zapped below
1287 \let\bb@sc@fontenc\space % <- " "
1288 \def\bb@tempa##1=##2\@nil{%
1289 \bb@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1290 \bb@vforeach{label=#1}{\bb@tempa##1\@nil}%
1291 \def\bb@tempa##1 ##2{% space -> comma
1292 ##1%
1293 \ifx\@empty##2\else\ifx,##1,\else,\fi\bb@afterfi\bb@tempa##2\fi}%
1294 \edef\bb@sc@fontenc{\expandafter\bb@tempa\bb@sc@fontenc\@empty}%
1295 \edef\bb@sc@label{\expandafter\zap@space\bb@sc@label\@empty}%
1296 \edef\bb@sc@charset{\expandafter\zap@space\bb@sc@charset\@empty}%
1297 \def\bb@encstring##1##2{%
1298 \bb@foreach\bb@sc@fontenc{%
1299 \bb@ifunset{T@###1}%
1300 }%
1301 {\ProvideTextCommand##1{###1}{##2}%
1302 \bb@tglobal##1%
1303 \expandafter
1304 \bb@tglobal\csname###1\string##1\endcsname}}}%
1305 \def\bb@sctest{%
1306 \bb@xin@{\, \bb@opt@strings,}{, \bb@sc@label, \bb@sc@fontenc,}}%
1307 \fi
1308 \ifx\bb@opt@strings\@nnil % ie, no strings key -> defaults
1309 \else\ifx\bb@opt@strings\relax % ie, strings=encoded
1310 \let\AfterBabelCommands\bb@aftercmds
1311 \let\SetString\bb@setstring
1312 \let\bb@stringdef\bb@encstring
1313 \else % ie, strings=value
1314 \bb@sctest
1315 \ifin@
1316 \let\AfterBabelCommands\bb@aftercmds
1317 \let\SetString\bb@setstring
1318 \let\bb@stringdef\bb@provstring
1319 \fi\fi\fi
1320 \bb@scswitch
1321 \ifx\bb@G\@empty
1322 \def\SetString##1##2{%
1323 \bb@error{Missing group for string \string##1}%
1324 {You must assign strings to some category, typically\\%
1325 captions or extras, but you set none}}%
1326 \fi

```



```

1327 \ifx\@empty#1%
1328   \bbl@usehooks{defaultcommands}{}%
1329 \else
1330   \@expandtwoargs
1331   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1332 \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) .

```

1333 \def\bbl@forlang#1#2{%
1334   \bbl@for#1\bbl@L{%
1335     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1336     \ifin@#2\relax\fi}}
1337 \def\bbl@scswitch{%
1338   \bbl@forlang\bbl@tempa{%
1339     \ifx\bbl@G\@empty\else
1340       \ifx\SetString@gobbletwo\else
1341         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1342         \bbl@xin@{\bbl@GL,}{,\bbl@screset,}%
1343         \ifin@\else
1344           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1345           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1346           \fi
1347         \fi
1348       \fi}}
1349 \AtEndOfPackage{%
1350   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1351   \let\bbl@scswitch\relax}
1352 \@onlypreamble\EndBabelCommands
1353 \def\EndBabelCommands{%
1354   \bbl@usehooks{stopcommands}{}%
1355   \endgroup
1356   \endgroup
1357   \bbl@scafter}

```

Now we define commands to be used inside `\StartBabelCommands`.

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

```

1358 \def\bbl@setstring#1#2{%
1359   \bbl@forlang\bbl@tempa{%
1360     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1361     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1362     {\global\expandafter % TODO - con \bbl@exp ?
1363       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1364       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
1365     {}}%
1366   \def\BabelString{\#2}%

```

```

1367 \bbl@usehooks{stringprocess}{}%
1368 \expandafter\bbl@stringdef
1369 \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```

1370 \ifx\bbl@opt@strings\relax
1371 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1372 \bbl@patchuclc
1373 \let\bbl@encoded\relax
1374 \def\bbl@encoded@uclc#1{%
1375 \inmathwarn#1%
1376 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1377 \expandafter\ifx\csname ?\string#1\endcsname\relax
1378 \TextSymbolUnavailable#1%
1379 \else
1380 \csname ?\string#1\endcsname
1381 \fi
1382 \else
1383 \csname\cf@encoding\string#1\endcsname
1384 \fi}
1385 \else
1386 \def\bbl@scset#1#2{\def#1{#2}}
1387 \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.

```

1388 <<*Macros local to BabelCommands>> ≡
1389 \def\SetStringLoop##1##2{%
1390 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1391 \count@z@
1392 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1393 \advance\count@\@ne
1394 \toks@\expandafter{\bbl@tempa}%
1395 \bbl@exp{%
1396 \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1397 \count@=\the\count@\relax}}}%
1398 <</Macros local to BabelCommands>>

```

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

```

1399 \def\bbl@aftercmds#1{%
1400 \toks@\expandafter{\bbl@scafter#1}%
1401 \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.

```

1402 <<*Macros local to BabelCommands>> ≡
1403 \newcommand\SetCase[3][[]]{%
1404 \bbl@patchuclc
1405 \bbl@forlang\bbl@tempa{%
1406 \expandafter\bbl@encstring

```

```

1407     \csname\bb1@tempa @bb1@uclc\endcsname{\bb1@tempa##1}%
1408     \expandafter\bb1@encstring
1409     \csname\bb1@tempa @bb1@uc\endcsname{##2}%
1410     \expandafter\bb1@encstring
1411     \csname\bb1@tempa @bb1@lc\endcsname{##3}}}%
1412 <</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.

```

1413 <<{*Macros local to BabelCommands}>> ≡
1414 \newcommand\SetHyphenMap[1]{%
1415   \bb1@forlang\bb1@tempa{%
1416     \expandafter\bb1@stringdef
1417     \csname\bb1@tempa @bb1@hyphenmap\endcsname{##1}}}%
1418 <</Macros local to BabelCommands>>

```

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

```

1419 \newcommand\BabelLower[2]{% one to one.
1420   \ifnum\lccode#1=#2\else
1421     \babel@savevariable{\lccode#1}%
1422     \lccode#1=#2\relax
1423   \fi}
1424 \newcommand\BabelLowerMM[4]{% many-to-many
1425   \@tempcnta=#1\relax
1426   \@tempcntb=#4\relax
1427   \def\bb1@tempa{%
1428     \ifnum\@tempcnta>#2\else
1429       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1430       \advance\@tempcnta#3\relax
1431       \advance\@tempcntb#3\relax
1432       \expandafter\bb1@tempa
1433     \fi}%
1434   \bb1@tempa}
1435 \newcommand\BabelLowerM0[4]{% many-to-one
1436   \@tempcnta=#1\relax
1437   \def\bb1@tempa{%
1438     \ifnum\@tempcnta>#2\else
1439       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1440       \advance\@tempcnta#3
1441       \expandafter\bb1@tempa
1442     \fi}%
1443   \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1444 <<{*More package options}>> ≡
1445 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
1446 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
1447 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
1448 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
1449 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1450 <</More package options>>

```

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

```

1451 \AtEndOfPackage{%
1452   \ifx\bb1@opt@hyphenmap\undefined
1453     \bb1@xin@{,}{\bb1@language@opts}%
1454     \chardef\bb1@opt@hyphenmap\ifin@4\else\@ne\fi
1455   \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.

```
1456 \bbl@trace{Macros related to glyphs}
1457 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1458   \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
1459   \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.

```
1460 \def\save@sf@q#1{\leavevmode
1461   \begingroup
1462   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1463   \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.

```
1464 \ProvideTextCommand{\quotedblbase}{OT1}{%
1465   \save@sf@q{\set@low@box{\textquotedblright\}}%
1466   \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
1467 \ProvideTextCommandDefault{\quotedblbase}{%
1468   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1469 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1470   \save@sf@q{\set@low@box{\textquoteright\}}%
1471   \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
1472 \ProvideTextCommandDefault{\quotesinglbase}{%
1473   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
\guillemotright 1474 \ProvideTextCommand{\guillemotleft}{OT1}{%
1475   \ifmmode
1476     \ll
1477   \else
1478     \save@sf@q{\nobreak
1479       \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
1480   \fi}
1481 \ProvideTextCommand{\guillemotright}{OT1}{%
1482   \ifmmode
1483     \gg
1484   \else
```

```

1485 \save@sff@q{\nobreak
1486 \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
1487 \fi}

```

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

```

1488 \ProvideTextCommandDefault{\guillemotleft}{%
1489 \UseTextSymbol{OT1}{\guillemotleft}}
1490 \ProvideTextCommandDefault{\guillemotright}{%
1491 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

\guilsinglright 1492 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1493 \ifmmode
1494 <%
1495 \else
1496 \save@sff@q{\nobreak
1497 \raise.2ex\hbox{\scriptscriptstyle<}\bbl@allowhyphens}%
1498 \fi}
1499 \ProvideTextCommand{\guilsinglright}{OT1}{%
1500 \ifmmode
1501 >%
1502 \else
1503 \save@sff@q{\nobreak
1504 \raise.2ex\hbox{\scriptscriptstyle>}\bbl@allowhyphens}%
1505 \fi}

```

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

```

1506 \ProvideTextCommandDefault{\guilsinglleft}{%
1507 \UseTextSymbol{OT1}{\guilsinglleft}}
1508 \ProvideTextCommandDefault{\guilsinglright}{%
1509 \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
`\IJ` encoded fonts. Therefore we fake it for the OT1 encoding.

```

1510 \DeclareTextCommand{\ij}{OT1}{%
1511 i\kern-0.02em\bbl@allowhyphens j}
1512 \DeclareTextCommand{\IJ}{OT1}{%
1513 I\kern-0.02em\bbl@allowhyphens J}
1514 \DeclareTextCommand{\ij}{T1}{\char188}
1515 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

1516 \ProvideTextCommandDefault{\ij}{%
1517 \UseTextSymbol{OT1}{\ij}}
1518 \ProvideTextCommandDefault{\IJ}{%
1519 \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).

```

1520 \def\crrtic@{\hrule height0.1ex width0.3em}

```

```

1521 \def\crttic@{\hrule height0.1ex width0.33em}
1522 \def\ddj@{%
1523   \setbox0\hbox{d}\dimen@=\ht0
1524   \advance\dimen@1ex
1525   \dimen@.45\dimen@
1526   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1527   \advance\dimen@ii.5ex
1528   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1529 \def\DDJ@{%
1530   \setbox0\hbox{D}\dimen@=.55\ht0
1531   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1532   \advance\dimen@ii.15ex % correction for the dash position
1533   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1534   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1535   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1536 %
1537 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1538 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

1539 \ProvideTextCommandDefault{\dj}{%
1540   \UseTextSymbol{OT1}{\dj}}
1541 \ProvideTextCommandDefault{\DJ}{%
1542   \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.

```

1543 \DeclareTextCommand{\SS}{OT1}{SS}
1544 \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 1545 \ProvideTextCommandDefault{\glq}{%
1546   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

1547 \ProvideTextCommand{\grq}{T1}{%
1548   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1549 \ProvideTextCommand{\grq}{TU}{%
1550   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1551 \ProvideTextCommand{\grq}{OT1}{%
1552   \save@sf@q{\kern-.0125em
1553     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1554     \kern.07em\relax}}
1555 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq 1556 \ProvideTextCommandDefault{\glqq}{%
1557   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```
1558 \ProvideTextCommand{\grqq}{T1}{%
1559 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1560 \ProvideTextCommand{\grqq}{TU}{%
1561 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1562 \ProvideTextCommand{\grqq}{OT1}{%
1563 \save@sf@q{\kern-.07em
1564 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1565 \kern.07em\relax}}
1566 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 1567 \ProvideTextCommandDefault{\flq}{%
1568 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1569 \ProvideTextCommandDefault{\frq}{%
1570 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 1571 \ProvideTextCommandDefault{\flqq}{%
1572 \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1573 \ProvideTextCommandDefault{\frqq}{%
1574 \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

9.11.4 Umlauts and tremas

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

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

```
1575 \def\umlauthigh{%
1576 \def\bbl@umlauta##1{\leavevmode\bgroup%
1577 \expandafter\accent\csname\fontencoding dqpos\endcsname
1578 ##1\bbl@allowhyphens\egroup}%
1579 \let\bbl@umlaute\bbl@umlauta}
1580 \def\umlautlow{%
1581 \def\bbl@umlauta{\protect\lower@umlaut}}
1582 \def\umlautelow{%
1583 \def\bbl@umlaute{\protect\lower@umlaut}}
1584 \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.

```
1585 \expandafter\ifx\csname U@D\endcsname\relax
1586 \csname newdimen\endcsname\U@D
1587 \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.

```

1588 \def\lower@umlaut#1{%
1589   \leavevmode\bgroup
1590     \U@D 1ex%
1591     {\setbox\z@\hbox{%
1592       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1593       \dimen@ -.45ex\advance\dimen@\ht\z@
1594       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1595     \expandafter\accent\csname\fontencoding dqpos\endcsname
1596     \fontdimen5\font\U@D #1%
1597   \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).

```

1598 \AtBeginDocument{%
1599   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
1600   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
1601   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
1602   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
1603   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
1604   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
1605   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
1606   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
1607   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
1608   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
1609   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%
1610 }

```

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

```

1611 \ifx\l@english\undefined
1612   \chardef\l@english\z@
1613 \fi
1614 \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.

```

1615 \bbl@trace{Bidi layout}
1616 \providecommand\IfBabelLayout[3]{#3}%
1617 \newcommand\BabelPatchSection[1]{%
1618   \@ifundefined{#1}{-}{%
1619     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1620     \@namedef{#1}{%
1621       \@ifstar{\bbl@presec@#1}}%
1622       {\@dblarg{\bbl@presec@x{#1}}}}%
1623 \def\bbl@presec@x#1[#2]#3{%
1624   \bbl@exp{%

```



```

1625   \\select@language@x{\bbl@main@language}%
1626   \\@nameuse{bbl@sspre@#1}%
1627   \\@nameuse{bbl@ss@#1}%
1628   [\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
1629   {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
1630   \\select@language@x{\languagename}}
1631 \def\bbl@presec@s#1#2{%
1632   \bbl@exp{%
1633     \\select@language@x{\bbl@main@language}%
1634     \\@nameuse{bbl@sspre@#1}%
1635     \\@nameuse{bbl@ss@#1})*%
1636     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1637     \\select@language@x{\languagename}}%
1638 \IfBabelLayout{sectioning}%
1639   {\BabelPatchSection{part}%
1640    \BabelPatchSection{chapter}%
1641    \BabelPatchSection{section}%
1642    \BabelPatchSection{subsection}%
1643    \BabelPatchSection{subsubsection}%
1644    \BabelPatchSection{paragraph}%
1645    \BabelPatchSection{subparagraph}}%
1646   \def\babel@toc#1{%
1647     \select@language@x{\bbl@main@language}}{}
1648 \IfBabelLayout{captions}%
1649   {\BabelPatchSection{caption}}{}

```

9.13 Load engine specific macros

```

1650 \bbl@trace{Input engine specific macros}
1651 \ifcase\bbl@engine
1652   \input txtbabel.def
1653 \or
1654   \input luababel.def
1655 \or
1656   \input xebabel.def
1657 \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.

```

1658 \bbl@trace{Creating languages and reading ini files}
1659 \newcommand\babelprovide[2][{}]{%
1660   \let\bbl@savelangname\languagename
1661   \edef\bbl@savelocaleid{\the\localeid}%
1662   % Set name and locale id
1663   \def\languagename{#2}%
1664   \bbl@id@assign
1665   \chardef\localeid\@nameuse{bbl@id@\languagename}%
1666   \let\bbl@KVP@captions\@nil
1667   \let\bbl@KVP@import\@nil
1668   \let\bbl@KVP@main\@nil
1669   \let\bbl@KVP@script\@nil
1670   \let\bbl@KVP@language\@nil
1671   \let\bbl@KVP@dir\@nil
1672   \let\bbl@KVP@hyphenrules\@nil
1673   \let\bbl@KVP@mapfont\@nil

```

```

1674 \let\bb1@KVP@maparabic\@nil
1675 \let\bb1@KVP@mapdigits\@nil
1676 \let\bb1@KVP@intraspace\@nil
1677 \let\bb1@KVP@intrapenalty\@nil
1678 \bb1@forkv{#1}{\bb1@csarg\def{KVP@##1}{##2}}% TODO - error handling
1679 \ifx\bb1@KVP@import\@nil\else
1680   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
1681     {\begingroup
1682       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
1683       \InputIfFileExists{babel-#2.tex}{}}%
1684     \endgroup}%
1685   {}%
1686 \fi
1687 \ifx\bb1@KVP@captions\@nil
1688   \let\bb1@KVP@captions\bb1@KVP@import
1689 \fi
1690 % Load ini
1691 \bb1@ifunset{date#2}%
1692   {\bb1@provide@new{#2}}%
1693   {\bb1@ifblank{#1}%
1694     {\bb1@error
1695       {If you want to modify `#2' you must tell how in\%
1696         the optional argument. See the manual for the\%
1697         available options.}%
1698       {Use this macro as documented}}%
1699     {\bb1@provide@renew{#2}}}%
1700 % Post tasks
1701 \bb1@exp{\bb1@babelensure[exclude=\today]{#2}}%
1702 \bb1@ifunset{\bb1@ensure@\languagename}%
1703   {\bb1@exp{%
1704     \DeclareRobustCommand\<bb1@ensure@\languagename>[1]{%
1705       \foreignlanguage{\languagename}%
1706       {###1}}}%
1707   }%
1708 % At this point all parameters are defined if 'import'. Now we
1709 % execute some code depending on them. But what about if nothing was
1710 % imported? We just load the very basic parameters: ids and a few
1711 % more.
1712 \bb1@ifunset{\bb1@lname#2}%
1713   {\def\BabelBeforeIni##1##2{%
1714     \begingroup
1715     \catcode\` [=12 \catcode\` \]=12 \catcode\` ==12 %
1716     \let\bb1@ini@captions@aux\@gobbletwo
1717     \def\bb1@inidate #####1.#####2.#####3.#####4\relax #####5#####6{%
1718       \bb1@read@ini{##1}%
1719       \bb1@exportkey{chrng}{characters.ranges}{}%
1720       \bb1@exportkey{dgnat}{numbers.digits.native}{}%
1721     \endgroup}% boxed, to avoid extra spaces:
1722     {\setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
1723     }%
1724   % -
1725 % Override script and language names with script= and language=
1726 \ifx\bb1@KVP@script\@nil\else
1727   \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
1728 \fi
1729 \ifx\bb1@KVP@language\@nil\else
1730   \bb1@csarg\edef{lname@#2}{\bb1@KVP@language}%
1731 \fi
1732 % For bidi texts, to switch the language based on direction

```

```

1733 \ifx\bb1@KVP@mapfont\@nil\else
1734   \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}{}%
1735   {\bb1@error{Option `bb1@KVP@mapfont' unknown for\%
1736     mapfont. Use `direction'.%
1737     {See the manual for details.}}}%
1738 \bb1@ifunset{bb1@lsys@language}{\bb1@provide@lsys@language}{}%
1739 \bb1@ifunset{bb1@wdir@language}{\bb1@provide@dirs@language}{}%
1740 \ifx\bb1@mapselect\@undefined
1741   \AtBeginDocument{%
1742     \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}%
1743     {\selectfont}}%
1744   \def\bb1@mapselect{%
1745     \let\bb1@mapselect\relax
1746     \edef\bb1@prefontid{\fontid\font}}%
1747   \def\bb1@mapdir##1{%
1748     {\def\language##1}%
1749     \let\bb1@ifrestoring\@firstoftwo % avoid font warning
1750     \bb1@switchfont
1751     \directlua{Babel.fontmap
1752       [\the\csname bb1@wdir@##1\endcsname]%
1753       [\bb1@prefontid]=\fontid\font}}%
1754   \fi
1755   \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir@language}}%
1756 \fi
1757 % For Southeast Asian, if interspace in ini -- TODO: as hook
1758 \ifcase\bb1@engine\or
1759   \bb1@ifunset{bb1@intsp@language}{}%
1760   {\expandafter\ifx\csname bb1@intsp@language\endcsname\@empty\else
1761     \bb1@xin@{\bb1@cs{sbc@language}}{Hant,Hans,Japan,Kore}%
1762     \ifin@
1763       \bb1@cjkintraspacespace
1764       \directlua{
1765         Babel = Babel or {}
1766         Babel.locale_props = Babel.locale_props or {}
1767         Babel.locale_props[\the\localeid].linebreak = 'c'
1768       }%
1769       \ifx\bb1@KVP@intraspacespace\@nil
1770         \bb1@exp{%
1771           \bb1@intraspacespace\bb1@cs{intsp@language}\@}%
1772         \fi
1773       \ifx\bb1@KVP@intrapenalty\@nil
1774         \bb1@intrapenalty0@@
1775       \fi
1776     \else
1777       \bb1@seaintraspacespace
1778       \ifx\bb1@KVP@intraspacespace\@nil
1779         \bb1@exp{%
1780           \bb1@intraspacespace\bb1@cs{intsp@language}\@}%
1781         \fi
1782       \directlua{
1783         Babel = Babel or {}
1784         Babel.sea_ranges = Babel.sea_ranges or {}
1785         Babel.set_chranges('\bb1@cs{sbc@language}',
1786           '\bb1@cs{chrng@language}')
1787       }%
1788       \ifx\bb1@KVP@intrapenalty\@nil
1789         \bb1@intrapenalty0@@
1790       \fi
1791     \fi

```

```

1792     \fi
1793     \ifx\bb1@KVP@intraspace\@nil\else % We may override the ini
1794         \expandafter\bb1@intraspace\bb1@KVP@intraspace\@@
1795     \fi
1796     \ifx\bb1@KVP@intrapenalty\@nil\else
1797         \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
1798     \fi}%
1799 \or
1800     \bb1@xin@\bb1@cs{sbcpr@\languagename}}{Thai,Lao,Khmr}%
1801     \ifin@
1802         \bb1@ifunset{\bb1@intsp@\languagename}}{%
1803             {\expandafter\ifx\csname \bb1@intsp@\languagename\endcsname\@empty\else
1804                 \ifx\bb1@KVP@intraspace\@nil
1805                     \bb1@exp{%
1806                         \\bb1@intraspace\bb1@cs{intsp@\languagename}}\@@}%
1807                 \fi
1808                 \ifx\bb1@KVP@intrapenalty\@nil
1809                     \bb1@intrapenalty0\@@
1810                 \fi
1811             \fi
1812             \ifx\bb1@KVP@intraspace\@nil\else % We may override the ini
1813                 \expandafter\bb1@intraspace\bb1@KVP@intraspace\@@
1814             \fi
1815             \ifx\bb1@KVP@intrapenalty\@nil\else
1816                 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
1817             \fi
1818             \ifx\bb1@ispace\@undefined
1819                 \AtBeginDocument{%
1820                     \expandafter\bb1@add
1821                     \csname selectfont \endcsname{\bb1@ispace}%
1822                     \def\bb1@ispace{\bb1@cs{xeisp@\bb1@cs{sbcpr@\languagename}}}%
1823                 \fi}%
1824     \fi
1825 \fi
1826 % Native digits, if provided in ini (TeX level, xe and lua)
1827 \ifcase\bb1@engine\else
1828     \bb1@ifunset{\bb1@dgnat@\languagename}}{%
1829         {\expandafter\ifx\csname \bb1@dgnat@\languagename\endcsname\@empty\else
1830             \expandafter\expandafter\expandafter
1831             \bb1@setdigits\csname \bb1@dgnat@\languagename\endcsname
1832             \ifx\bb1@KVP@maparabic\@nil\else
1833                 \ifx\bb1@latinarabic\@undefined
1834                     \expandafter\let\expandafter\@arabic
1835                     \csname \bb1@counter@\languagename\endcsname
1836                 \else % ie, if layout=counters, which redefines \@arabic
1837                     \expandafter\let\expandafter\bb1@latinarabic
1838                     \csname \bb1@counter@\languagename\endcsname
1839                 \fi
1840             \fi
1841         \fi}%
1842 \fi
1843 % Native digits (lua level).
1844 \ifodd\bb1@engine
1845     \ifx\bb1@KVP@mapdigits\@nil\else
1846         \bb1@ifunset{\bb1@dgnat@\languagename}}{%
1847             {\RequirePackage{luatexbase}%
1848                 \bb1@activate@preotf
1849                 \directlua{
1850                     Babel = Babel or {} %% -> presets in luababel

```

```

1851     Babel.digits_mapped = true
1852     Babel.digits = Babel.digits or {}
1853     Babel.digits[\the\localeid] =
1854       table.pack(string.utfvalue('\bbl@cs{dgnat@\languagename}'))
1855     if not Babel.numbers then
1856       function Babel.numbers(head)
1857         local LOCALE = luatexbase.registernumber'bbl@attr@locale'
1858         local GLYPH = node.id'glyph'
1859         local inmath = false
1860         for item in node.traverse(head) do
1861           if not inmath and item.id == GLYPH then
1862             local temp = node.get_attribute(item, LOCALE)
1863             if Babel.digits[temp] then
1864               local chr = item.char
1865               if chr > 47 and chr < 58 then
1866                 item.char = Babel.digits[temp][chr-47]
1867               end
1868             end
1869             elseif item.id == node.id'math' then
1870               inmath = (item.subtype == 0)
1871             end
1872           end
1873         return head
1874       end
1875     end
1876   }}
1877   \fi
1878 \fi
1879 % To load or reload the babel-*.tex, if require.babel in ini
1880 \bbl@ifunset{bbl@rqtex@\languagename}{}%
1881   {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
1882     \let\BabelBeforeIni\@gobbletwo
1883     \chardef\atcatcode=\catcode`\@
1884     \catcode`\@=11\relax
1885     \InputIfFileExists{babel-\bbl@cs{rqtex@\languagename}.tex}{}%
1886     \catcode`\@=\atcatcode
1887     \let\atcatcode\relax
1888   \fi}%
1889 \let\languagename\bbl@savelangname
1890 \chardef\localeid\bbl@savelocaleid\relax}

```

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

```

1891 \def\bbl@setdigits#1#2#3#4#5{%
1892   \bbl@exp{%
1893     \def\<\languagename digits>####1{%      ie, \langdigits
1894       \<bbl@digits@\languagename>####1\\\@nil}%
1895     \def\<\languagename counter>####1{%      ie, \langcounter
1896       \\expandafter\<bbl@counter@\languagename>%
1897       \\csname c@####1\endcsname}%
1898     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
1899       \\expandafter\<bbl@digits@\languagename>%
1900       \\number####1\\\@nil}}%
1901 \def\bbl@tempa##1##2##3##4##5{%
1902   \bbl@exp{%      Wow, quite a lot of hashes! :- (
1903     \def\<bbl@digits@\languagename>#####1{%
1904       \\ifx#####1\\\@nil          % ie, \bbl@digits@lang
1905       \\else
1906         \\ifx0#####1#1%

```



```

1963 \EndBabelCommands
1964 \fi
1965 \ifx\babel@KVP@import\@nil\else
1966 \StartBabelCommands*{#1}{date}%
1967 \babel@savetoday
1968 \babel@savedate
1969 \EndBabelCommands
1970 \fi
1971 \babel@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

1972 \def\babel@provide@hyphens#1{%
1973 \let\babel@tempa\relax
1974 \ifx\babel@KVP@hyphenrules\@nil\else
1975 \babel@replace\babel@KVP@hyphenrules{ }{,}%
1976 \babel@foreach\babel@KVP@hyphenrules{%
1977 \ifx\babel@tempa\relax % if not yet found
1978 \babel@ifsamestring{##1}{+}%
1979 {\babel@exp{\addlanguage\<l@##1>}}}%
1980 }%
1981 \babel@ifunset{l@##1}%
1982 }%
1983 {\babel@exp{\let\babel@tempa\<l@##1>}}%
1984 \fi}%
1985 \fi
1986 \ifx\babel@tempa\relax % if no opt or no language in opt found
1987 \ifx\babel@KVP@import\@nil\else % if importing
1988 \babel@exp{%
1989 \babel@ifblank{\@nameuse{babel@hyphr@#1}}%
1990 }%
1991 {\let\babel@tempa\<l@\@nameuse{babel@hyphr@\language}\>}}%
1992 \fi
1993 \fi
1994 \babel@ifunset{babel@tempa}% ie, relax or undefined
1995 {\babel@ifunset{l@#1}% no hyphenrules found - fallback
1996 {\babel@exp{\adddialect\<l@#1>\language}}}%
1997 }% so, l@<lang> is ok - nothing to do
1998 {\babel@exp{\adddialect\<l@#1>\babel@tempa}}% found in opt list or ini

```

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

```

1999 \def\babel@read@ini#1{%
2000 \openin1=babel-#1.ini % FIXME - number must not be hardcoded
2001 \ifeof1
2002 \babel@error
2003 {There is no ini file for the requested language\%
2004 (#1). Perhaps you misspelled it or your installation\%
2005 is not complete.}%
2006 {Fix the name or reinstall babel.}%
2007 \else
2008 \let\babel@section\@empty
2009 \let\babel@savestrings\@empty
2010 \let\babel@savetoday\@empty
2011 \let\babel@savedate\@empty
2012 \let\babel@inireader\babel@iniskip
2013 \babel@info{Importing data from babel-#1.ini for \language}%
2014 \loop
2015 \if T\ifeof1F\fi T\relax % Trick, because inside \loop
2016 \endlinechar\m@ne

```

```

2017     \read1 to \bbl@line
2018     \endlinechar``^^M
2019     \ifx\bbl@line\@empty\else
2020         \expandafter\bbl@iniline\bbl@line\bbl@iniline
2021     \fi
2022     \repeat
2023 \fi}
2024 \def\bbl@iniline#1\bbl@iniline{%
2025     \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inireader}#1\@}% ]

```

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

```

2026 \def\bbl@iniskip#1\@{%          if starts with ;
2027 \def\bbl@inisec[#1]#2\@{%      if starts with opening bracket
2028     \@nameuse{bbl@secpost\bbl@section}% ends previous section
2029 \def\bbl@section{#1}%
2030 \@nameuse{bbl@secpre\bbl@section}% starts current section
2031 \bbl@ifunset{bbl@inikv#1}%
2032     {\let\bbl@inireader\bbl@iniskip}%
2033     {\bbl@exp{\let\\bbl@inireader<\bbl@inikv@#1>}}}

```

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

```

2034 \def\bbl@inikv#1=#2\@{%       key=value
2035     \bbl@trim@def\bbl@tempa{#1}%
2036     \bbl@trim\toks@{#2}%
2037     \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.

```

2038 \def\bbl@exportkey#1#2#3{%
2039     \bbl@ifunset{bbl@kv@#2}%
2040     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2041     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2042         \bbl@csarg\gdef{#1@\languagename}{#3}}%
2043     \else
2044         \bbl@exp{\global\let<bbl@#1@\languagename><bbl@kv@#2>}%
2045     \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.

```

2046 \let\bbl@inikv@identification\bbl@inikv
2047 \def\bbl@secpost@identification{%
2048     \bbl@exportkey{lname}{identification.name.english}{}%
2049     \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
2050     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2051     \bbl@exportkey{sname}{identification.script.name}{}%
2052     \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2053     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
2054 \let\bbl@inikv@typography\bbl@inikv
2055 \let\bbl@inikv@characters\bbl@inikv
2056 \let\bbl@inikv@numbers\bbl@inikv
2057 \def\bbl@after@ini{%
2058     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2059     \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2060     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2061     \bbl@exportkey{intsp}{typography.intraspace}{}%
2062     \bbl@exportkey{jstfy}{typography.justify}{w}%
2063     \bbl@exportkey{chrng}{characters.ranges}{}%

```



```

2064 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2065 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2066 \bbl@xin@{0.5}{\@nameuse{bbl@kv@identification.version}}%
2067 \ifin@
2068   \bbl@warning{%
2069     There are neither captions nor date in ``\languagename'.\%
2070     It may not be suitable for proper typesetting, and it\%
2071     could change. Reported}%
2072 \fi
2073 \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
2074 \ifin@
2075   \bbl@warning{%
2076     The ``\languagename' date format may not be suitable\%
2077     for proper typesetting, and therefore it very likely will\%
2078     change in a future release. Reported}%
2079 \fi
2080 \bbl@toglobal\bbl@savetoday
2081 \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.

```

2082 \ifcase\bbl@engine
2083   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
2084     \bbl@ini@captions@aux{#1}{#2}}
2085 \else
2086   \def\bbl@inikv@captions#1=#2\@@{%
2087     \bbl@ini@captions@aux{#1}{#2}}
2088 \fi

```

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

```

2089 \def\bbl@ini@captions@aux#1#2{%
2090   \bbl@trim\def\bbl@tempa{#1}%
2091   \bbl@ifblank{#2}%
2092   {\bbl@exp{%
2093     \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
2094   {\bbl@trim\toks@{#2}}}%
2095   \bbl@exp{%
2096     \bbl@add\bbl@savestrings{%
2097       \SetString\<\bbl@tempa name>{\the\toks@}}}}

```

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

TODO. Remove copypaste pattern.

```

2098 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%           for defaults
2099   \bbl@inidate#1...\relax{#2}{}}
2100 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
2101   \bbl@inidate#1...\relax{#2}{islamic}}
2102 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
2103   \bbl@inidate#1...\relax{#2}{hebrew}}
2104 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
2105   \bbl@inidate#1...\relax{#2}{persian}}
2106 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
2107   \bbl@inidate#1...\relax{#2}{indian}}
2108 \ifcase\bbl@engine
2109   \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%   override
2110     \bbl@inidate#1...\relax{#2}{}}
2111   \bbl@csarg\def{secpre@date.gregorian.licr}{%           discard uni

```

```

2112 \ifcase\bb1@engine\let\bb1@savodate\@empty\fi}
2113 \fi
2114 % eg: 1=months, 2=wide, 3=1, 4=dummy
2115 \def\bb1@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2116 \bb1@trim@def\bb1@tempa{#1.#2}%
2117 \bb1@ifsamestring{\bb1@tempa}{months.wide}% to savodate
2118 {\bb1@trim@def\bb1@tempa{#3}%
2119 \bb1@trim\toks@{#5}%
2120 \bb1@exp{%
2121 \\bb1@add\\bb1@savodate{%
2122 \\SetString\<month\romannumeral\bb1@tempa#6name>{\the\toks@}}}%
2123 {\bb1@ifsamestring{\bb1@tempa}{date.long}% defined now
2124 {\bb1@trim@def\bb1@toreplace{#5}%
2125 \bb1@TG@date
2126 \global\bb1@csarg\let{date@\languagename}\bb1@toreplace
2127 \bb1@exp{%
2128 \gdef\<\languagename date>{\protect\<\languagename date >}%
2129 \gdef\<\languagename date >####1####2####3{%
2130 \\bb1@usedategrouptrue
2131 \<bb1@ensure@\languagename>{%
2132 \<bb1@date@\languagename>{####1}{####2}{####3}}}%
2133 \\bb1@add\\bb1@savetoday{%
2134 \\SetString\\today{%
2135 \<\languagename date>{\the\year}{\the\month}{\the\day}}}}}%
2136 {}

```

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.

```

2137 \let\bb1@calendar\@empty
2138 \newcommand\BabelDateSpace{\nobreakspace}
2139 \newcommand\BabelDateDot{.\@}
2140 \newcommand\BabelDated[1]{\number#1}
2141 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
2142 \newcommand\BabelDateM[1]{\number#1}
2143 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2144 \newcommand\BabelDateMMMM[1]{%
2145 \csname month\romannumeral#1\bb1@calendar name\endcsname}}%
2146 \newcommand\BabelDatey[1]{\number#1}%
2147 \newcommand\BabelDateyy[1]{%
2148 \ifnum#1<10 0\number#1 %
2149 \else\ifnum#1<100 \number#1 %
2150 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2151 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2152 \else
2153 \bb1@error
2154 {Currently two-digit years are restricted to the\
2155 range 0-9999.}%
2156 {There is little you can do. Sorry.}%
2157 \fi\fi\fi\fi}}
2158 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2159 \def\bb1@replace@finish@iii#1{%
2160 \bb1@exp{\def\#1####1####2####3{\the\toks@}}
2161 \def\bb1@TG@date{%
2162 \bb1@replace\bb1@toreplace{[ ]}{\BabelDateSpace}}%
2163 \bb1@replace\bb1@toreplace{[. ]}{\BabelDateDot}}%
2164 \bb1@replace\bb1@toreplace{[d]}{\BabelDated{####3}}%
2165 \bb1@replace\bb1@toreplace{[dd]}{\BabelDatedd{####3}}%
2166 \bb1@replace\bb1@toreplace{[M]}{\BabelDateM{####2}}%

```

```

2167 \bbl@replace\bbl@toreplace{[MM]}\BabelDateMM{###2}%
2168 \bbl@replace\bbl@toreplace{[MMMM]}\BabelDateMMMM{###2}%
2169 \bbl@replace\bbl@toreplace{[y]}\BabelDatey{###1}%
2170 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{###1}%
2171 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{###1}%
2172 % Note after \bbl@replace \toks@ contains the resulting string.
2173 % TODO - Using this implicit behavior doesn't seem a good idea.
2174 \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.

```

2175 \def\bbl@provide@lsys#1{%
2176 \bbl@ifunset{bbl@lname@#1}%
2177   {\bbl@ini@ids{#1}}%
2178   }%
2179 \bbl@csarg\let{lsys@#1}\@empty
2180 \bbl@ifunset{bbl@sname@#1}\bbl@csarg\gdef{sname@#1}{Default}}}%
2181 \bbl@ifunset{bbl@sotf@#1}\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
2182 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2183 \bbl@ifunset{bbl@lname@#1}}%
2184   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2185 \bbl@csarg\bbl@tglobal{lsys@#1}}

```

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

```

2186 \def\bbl@ini@ids#1{%
2187 \def\BabelBeforeIni##1##2{%
2188 \begingroup
2189   \bbl@add\bbl@secpost@identification{\closein1 }%
2190   \catcode`\[=12 \catcode`\]=12 \catcode`\=12 %
2191   \bbl@read@ini{##1}%
2192 \endgroup}% boxed, to avoid extra spaces:
2193 {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}}

```

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

10.1 The redefinition of the style commands

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

```

2194 {\def\format{lplain}
2195 \ifx\fmtname\format
2196 \else
2197 \def\format{LaTeX2e}
2198 \ifx\fmtname\format
2199 \else
2200 \aftergroup\endinput
2201 \fi
2202 \fi}

```

10.2 Cross referencing macros

The \LaTeX book states:

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

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

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

The only way to accomplish this in most cases is to use the trick described in the \TeX book [2] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, ‘`\meaning\A`’ with `\A` defined as ‘`\def\A#1{\B}`’ expands to the characters ‘`macro:#1->\B`’ with all category codes set to ‘other’ or ‘space’.

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

```
2203 %\bbl@redefine\newlabel#1#2{%
2204 % \@safe@activestru\vorg@newlabel{#1}{#2}\@safe@activesfalse}
```

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

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

```
2205 <<{*More package options}>> ≡
2206 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2207 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2208 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2209 <</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.

```
2210 \bbl@trace{Cross referencing macros}
2211 \ifx\bbl@opt@safe\@empty\else
2212 \def\@newl@bel#1#2#3{%
2213   {\@safe@activestru
2214     \bbl@ifunset{#1@#2}%
2215     \relax
2216     {\gdef\@multiplelabels{%
2217       \@latex@warning@no@line{There were multiply-defined labels}}%
2218       \@latex@warning@no@line{Label `#2' multiply defined}}%
2219     \global\@namedef{#1@#2}{#3}}}
```

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

```
2220 \CheckCommand*\@testdef[3]{%
2221   \def\reserved@a{#3}%
2222   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2223   \else
2224     \@tempwatru
2225   \fi}
```

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

```
2226 \def\@testdef#1#2#3{%
2227   \@safe@activestru
```

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

```
2228 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```
2229 \def\bbl@tempb{#3}%
2230 \@safe@activesfalse
```

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

```
2231 \ifx\bbl@tempa\relax
2232 \else
2233   \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2234 \fi
```

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

```
2235 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```
2236 \ifx\bbl@tempa\bbl@tempb
2237 \else
2238   \@tempswatrue
2239 \fi}
2240 \fi
```

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

```
2241 \bbl@xin@{R}\bbl@opt@safe
2242 \ifin@
2243 \bbl@redefineroobust\ref#1{%
2244   \@safe@activestru\org@ref{#1}\@safe@activesfalse}
2245 \bbl@redefineroobust\pageref#1{%
2246   \@safe@activestru\org@pageref{#1}\@safe@activesfalse}
2247 \else
2248 \let\org@ref\ref
2249 \let\org@pageref\pageref
2250 \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.

```
2251 \bbl@xin@{B}\bbl@opt@safe
2252 \ifin@
2253 \bbl@redefine\@citex[#1]#2{%
2254   \@safe@activestru\edef\@tempa{#2}\@safe@activesfalse
2255   \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.

```

2256 \AtBeginDocument{%
2257   \@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.)

```

2258   \def\@citex[#1][#2]#3{%
2259     \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
2260     \org@citex[#1][#2]{\@tempa}%
2261   }{}}

```

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

```

2262 \AtBeginDocument{%
2263   \@ifpackageloaded{cite}{%
2264     \def\@citex[#1]#2{%
2265       \@safe@activestrue\org@citex[#1][#2]\@safe@activesfalse}%
2266     }{}}

```

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

```

2267 \bbl@redefine\nocite#1{%
2268   \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

2269 \bbl@redefine\bibcite{%
2270   \bbl@cite@choice
2271   \bibcite}

```

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

```

2272 \def\bbl@bibcite#1#2{%
2273   \org@bibcite{#1}{\@safe@activesfalse#2}}

```

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

```

2274 \def\bbl@cite@choice{%
2275   \global\let\bibcite\bbl@bibcite

```

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

```

2276   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}}%
2277   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}}%

```

Make sure this only happens once.

```

2278   \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.

```
2279 \AtBeginDocument{\bbl@cite@choice}
```

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

```
2280 \bbl@redefine\@bibitem#1{%
2281   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
2282 \else
2283   \let\org@nocite\nocite
2284   \let\org@@citex\@citex
2285   \let\org@bibcite\bibcite
2286   \let\org@@bibitem\@bibitem
2287 \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.

```
2288 \bbl@trace{Marks}
2289 \IfBabelLayout{sectioning}
2290   {\ifx\bbl@opt@headfoot\@nnil
2291     \g@addto@macro\@resetactivechars{%
2292       \set@typeset@protect
2293       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2294       \let\protect\noexpand
2295       \edef\thepage{%
2296         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2297     \fi}
2298   {\bbl@redefine\markright#1{%
2299     \bbl@ifblank{#1}%
2300     {\org@markright{}}%
2301     {\toks@{#1}%
2302       \bbl@exp{%
2303         \\org@markright{\\protect\\foreignlanguage{\languagename}%
2304           {\\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 needed to do that again with the new definition of \markboth.

```
2305 \ifx\@mkboth\markboth
2306   \def\bbl@tempc{\let\@mkboth\markboth}
2307 \else
2308   \def\bbl@tempc{}
2309 \fi
```

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

```
2310 \bbl@redefine\markboth#1#2{%
2311   \protected@edef\bbl@tempb##1{%
2312     \protect\foreignlanguage
2313     {\languagename}{\protect\bbl@restore@actives##1}}%
2314   \bbl@ifblank{#1}%
2315     {\toks@{}}%
2316     {\toks@\expandafter{\bbl@tempb{#1}}}%
2317   \bbl@ifblank{#2}%
2318     {\@temptokena{}}%
2319     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2320   \bbl@exp{\@org@markboth{the\toks@}{the\@temptokena}}}
```

and copy it to `\mkboth` if necessary.

```
2321 \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.

```
2322 \bbl@trace{Preventing clashes with other packages}
2323 \bbl@xin@{R}\bbl@opt@safe
2324 \ifin@
2325 \AtBeginDocument{%
2326   \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```
2327   \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.

```
2328     \let\bbl@temp@pref\pageref
2329     \let\pageref\org@pageref
2330     \let\bbl@temp@ref\ref
2331     \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.

```
2332     \@safe@activestrue
2333     \org@ifthenelse{#1}%
2334       {\let\pageref\bbl@temp@pref
2335        \let\ref\bbl@temp@ref
```



```

2336         \@safe@activesfalse
2337         #2}%
2338     {\let\pageref\bbl@temp@pref
2339      \let\ref\bbl@temp@ref
2340      \@safe@activesfalse
2341      #3}%
2342     }%
2343   }{}%
2344 }

```

10.4.2 varioref

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

```

\Ref 2345 \AtBeginDocument{%
2346     \@ifpackageloaded{varioref}{%
2347         \bbl@redefine\@vppageref#1[#2]#3{%
2348             \@safe@activetrue
2349             \org@@@vppageref{#1}[#2]{#3}%
2350             \@safe@activesfalse}%

```

The same needs to happen for `\vrefpagemum`.

```

2351     \bbl@redefine\vrefpagemum#1#2{%
2352         \@safe@activetrue
2353         \org@vrefpagemum{#1}{#2}%
2354         \@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.

```

2355     \expandafter\def\csname Ref \endcsname#1{%
2356         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2357     }{}%
2358 }
2359 \fi

```

10.4.3 hpline

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hpline` 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 `hpline` is loaded.

```

2360 \AtEndOfPackage{%
2361     \AtBeginDocument{%
2362         \@ifpackageloaded{hpline}%

```

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

```

2363         {\expandafter\ifx\csname normal@char:string\endcsname\relax
2364             \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.

```

2365         \makeatletter
2366         \def\@currname{hpline}\input{hpline.sty}\makeatother

```

```
2367     \fi}%
2368     {}}}
```

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.

```
2369 \AtBeginDocument{%
2370   \ifx\pdfstringdefDisableCommands\undefined\else
2371     \pdfstringdefDisableCommands{\languageshortands{system}}%
2372   \fi}
```

10.4.5 fancyhdr

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

```
2373 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2374   \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.

```
2375 \def\substitutefontfamily#1#2#3{%
2376   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2377   \immediate\write15{%
2378     \string\ProvidesFile{#1#2.fd}%
2379     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2380     \space generated font description file]^J
2381     \string\DeclareFontFamily{#1}{#2}{}^J
2382     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^J
2383     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^J
2384     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^J
2385     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^J
2386     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^J
2387     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^J
2388     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^J
2389     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^J
2390   }%
2391   \closeout15
2392 }
```

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

```
2393 \@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

```
2394 \bbl@trace{Encoding and fonts}
2395 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
2396 \newcommand\BabelNonText{TS1,T3,TS3}
2397 \let\org@TeX\TeX
2398 \let\org@LaTeX\LaTeX
2399 \let\ensureascii\@firstofone
2400 \AtBeginDocument{%
2401   \in@false
2402   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2403     \ifin@%else
2404       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
2405     \fi}%
2406   \ifin@ % if a text non-ascii has been loaded
2407     \def\ensureascii#1{\fontencoding{OT1}\selectfont#1}}%
2408     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2409     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2410     \def\bbl@tempb#1\@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
2411     \def\bbl@tempc#1ENC.DEF#2\@{\%
2412       \ifx\@empty#2%else
2413         \bbl@ifunset{T@#1}%
2414         {}%
2415         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
2416         \ifin@
2417           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2418           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2419         \else
2420           \def\ensureascii##1{\fontencoding{#1}\selectfont##1}}%
2421         \fi}%
2422     \fi}%
2423   \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@ de mas??
2424   \bbl@xin@{\,cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
2425   \ifin@%else
2426     \edef\ensureascii#1{%
2427       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2428   \fi
2429 \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.

```
2430 \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.

```
2431 \AtBeginDocument{%
2432   \@ifpackageloaded{fontspec}%
2433   {\xdef\latinencoding{%
2434     \ifx\UTFencname\@undefined
2435       EU\ifcase\bbl@engine\or2\or1\fi
2436     \else
2437       \UTFencname
```

```

2438     \fi}}%
2439   {\gdef\latinencoding{OT1}%
2440     \ifx\cf@encoding\bbl@t@one
2441     \xdef\latinencoding{\bbl@t@one}%
2442     \else
2443     \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
2444     \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.

```

2445 \DeclareRobustCommand{\latintext}{%
2446   \fontencoding{\latinencoding}\selectfont
2447   \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.

```

2448 \ifx\@undefined\DeclareTextFontCommand
2449   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2450 \else
2451   \DeclareTextFontCommand{\textlatin}{\latintext}
2452 \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>).

```

2453 \bbl@trace{Basic (internal) bidi support}
2454 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2455 \def\bbl@rscripts{%
2456   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2457   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
2458   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
2459   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%

```

```

2460 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2461 Old South Arabian,}%
2462 \def\bbl@provide@dirs#1{%
2463   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2464   \ifin@
2465     \global\bbl@csarg\chardef{wdir@#1}\@ne
2466     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2467     \ifin@
2468       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2469       \fi
2470     \else
2471       \global\bbl@csarg\chardef{wdir@#1}\z@
2472     \fi
2473   \ifodd\bbl@engine
2474     \bbl@csarg\ifcase{wdir@#1}%
2475       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
2476     \or
2477       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
2478     \or
2479       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
2480     \fi
2481   \fi}
2482 \def\bbl@switchdir{%
2483   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%
2484   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}}%
2485   \bbl@exp{\bbl@setdirs\bbl@cs{wdir@\languagename}}}}
2486 \def\bbl@setdirs#1{% TODO - math
2487   \ifcase\bbl@select@type % TODO - strictly, not the right test
2488     \bbl@bodydir{#1}%
2489     \bbl@pardir{#1}%
2490   \fi
2491   \bbl@textdir{#1}}
2492 \ifodd\bbl@engine % luatex=1
2493   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2494   \DisableBabelHook{babel-bidi}
2495   \chardef\bbl@thetextdir\z@
2496   \chardef\bbl@thepardir\z@
2497   \def\bbl@getluadir#1{%
2498     \directlua{
2499       if tex.#1dir == 'TLT' then
2500         tex.sprint('0')
2501       elseif tex.#1dir == 'TRT' then
2502         tex.sprint('1')
2503       end}}
2504   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 r1
2505     \ifcase#3\relax
2506       \ifcase\bbl@getluadir{#1}\relax\else
2507         #2 TLT\relax
2508       \fi
2509     \else
2510       \ifcase\bbl@getluadir{#1}\relax
2511         #2 TRT\relax
2512       \fi
2513     \fi}
2514   \def\bbl@textdir#1{%
2515     \bbl@setluadir{text}\textdir{#1}%
2516     \chardef\bbl@thetextdir#1\relax
2517     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2518   \def\bbl@pardir#1{%

```

```

2519 \bbl@setluadir{par}\pardir{#1}%
2520 \chardef\bbl@thepardir#1\relax}
2521 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2522 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2523 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
2524 % Sadly, we have to deal with boxes in math with basic.
2525 % Activated every math with the package option bidi=:
2526 \def\bbl@mathboxdir{%
2527 \ifcase\bbl@thetextdir\relax
2528 \everyhbox{\textdir TLT\relax}%
2529 \else
2530 \everyhbox{\textdir TRT\relax}%
2531 \fi}
2532 \else % pdftex=0, xetex=2
2533 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2534 \DisableBabelHook{babel-bidi}
2535 \newcount\bbl@dirlevel
2536 \chardef\bbl@thetextdir\z@
2537 \chardef\bbl@thepardir\z@
2538 \def\bbl@textdir#1{%
2539 \ifcase#1\relax
2540 \chardef\bbl@thetextdir\z@
2541 \bbl@textdir@i\beginL\endL
2542 \else
2543 \chardef\bbl@thetextdir@ne
2544 \bbl@textdir@i\beginR\endR
2545 \fi}
2546 \def\bbl@textdir@i#1#2{%
2547 \ifhmode
2548 \ifnum\currentgrouplevel>\z@
2549 \ifnum\currentgrouplevel=\bbl@dirlevel
2550 \bbl@error{Multiple bidi settings inside a group}%
2551 {I'll insert a new group, but expect wrong results.}%
2552 \bgroup\aftergroup#2\aftergroup\egroup
2553 \else
2554 \ifcase\currentgrouptype\or % 0 bottom
2555 \aftergroup#2% 1 simple {}
2556 \or
2557 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2558 \or
2559 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2560 \or\or\or % vbox vtop align
2561 \or
2562 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2563 \or\or\or\or\or\or % output math disc insert vcent mathchoice
2564 \or
2565 \aftergroup#2% 14 \begingroup
2566 \else
2567 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2568 \fi
2569 \fi
2570 \bbl@dirlevel\currentgrouplevel
2571 \fi
2572 #1%
2573 \fi}
2574 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2575 \let\bbl@bodydir@gobble
2576 \let\bbl@pagedir@gobble
2577 \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).

```

2578 \def\bbl@xebidipar{%
2579   \let\bbl@xebidipar\relax
2580   \TeXeTstate\@ne
2581   \def\bbl@xeverypar{%
2582     \ifcase\bbl@thepardir
2583       \ifcase\bbl@thetextdir\else\beginR\fi
2584     \else
2585       {\setbox\z@\lastbox\beginR\box\z@}%
2586     \fi}%
2587   \let\bbl@severypar\everypar
2588   \newtoks\everypar
2589   \everypar=\bbl@severypar
2590   \bbl@severypar{\bbl@xeverypar\the\everypar}}
2591 \@ifpackagewith{babel}{bidi=bidi}%
2592 {\let\bbl@textdir@i@gobbletwo
2593   \let\bbl@xebidipar\@empty
2594   \AddBabelHook{bidi}{foreign}{%
2595     \def\bbl@tempa{\def\BabelText###1}%
2596     \ifcase\bbl@thetextdir
2597       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
2598     \else
2599       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
2600     \fi}
2601   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}}
2602 {}%
2603 \fi

```

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

```

2604 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
2605 \AtBeginDocument{%
2606   \ifx\pdfstringdefDisableCommands\undefined\else
2607     \ifx\pdfstringdefDisableCommands\relax\else
2608       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2609     \fi
2610   \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`.

```

2611 \bbl@trace{Local Language Configuration}
2612 \ifx\loadlocalcfg\undefined
2613   \@ifpackagewith{babel}{noconfigs}%
2614   {\let\loadlocalcfg@gobble}%
2615   {\def\loadlocalcfg#1{%
2616     \InputIfFileExists{#1.cfg}%
2617     {\typeout{*****^J%
2618               * Local config file #1.cfg used^^J%
2619               *}}%
2620     \@empty}}

```

```
2621 \fi
```

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

```
2622 \ifx\@unexpandable@protect\@undefined
2623   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2624   \long\def\protected@write#1#2#3{%
2625     \begingroup
2626       \let\thepage\relax
2627       #2%
2628       \let\protect\@unexpandable@protect
2629       \edef\reserved@a{\write#1{#3}}%
2630       \reserved@a
2631     \endgroup
2632   \if@nobreak\ifvmode\nobreak\fi\fi}
2633 \fi
2634 </core>
2635 <*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.

```
2636 <<Make sure ProvidesFile is defined>>
2637 \ProvidesFile{switch.def}[\<date>] <<version>> Babel switching mechanism]
2638 <<Load macros for plain if not LaTeX>>
2639 <<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.

```
2640 \def\bbl@version{\<version>}
2641 \def\bbl@date{\<date>}
2642 \def\adddialect#1#2{%
2643   \global\chardef#1#2\relax
2644   \bbl@usehooks{adddialect}{#1}{#2}}%
2645   \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.

```
2646 \def\bbl@fixname#1{%
2647   \begingroup
2648     \def\bbl@tempe{l@}%
2649     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2650     \bbl@tempd
2651     {\lowercase\expandafter{\bbl@tempd}%
2652      {\uppercase\expandafter{\bbl@tempd}%
2653       \@empty
2654       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2655        \uppercase\expandafter{\bbl@tempd}}}}%
2656     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2657      \lowercase\expandafter{\bbl@tempd}}}%
```



```

2658     \@empty
2659     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2660     \bbl@tempd}
2661 \def\bbl@iflanguage#1{%
2662   \@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.

```

2663 \def\iflanguage#1{%
2664   \bbl@iflanguage{#1}{%
2665     \ifnum\csname l@#1\endcsname=\language
2666       \expandafter\@firstoftwo
2667     \else
2668       \expandafter\@secondoftwo
2669     \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.

```

2670 \let\bbl@select@type\z@
2671 \edef\selectlanguage{%
2672   \noexpand\protect
2673   \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`.

```

2674 \ifx\@undefined\protect\let\protect\relax\fi

```

As L^AT_EX 2.09 writes to files *expanded* whereas L^AT_EX 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.

```

2675 \ifx\documentclass\@undefined
2676   \def\xstring{\string\string\string}
2677 \else
2678   \let\xstring\string
2679 \fi

```

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

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

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

```
2680 \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`

```
2681 \def\bbl@push@language{%
2682   \xdef\bbl@language@stack{\languagename+\bbl@language@stack}}
```

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

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

```
2683 \def\bbl@pop@lang#1+#2-#3{%
2684   \edef\languagename{#1}\xdef#3{#2}}
```

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

```
2685 \let\bbl@ifrestoring\@secondoftwo
2686 \def\bbl@pop@language{%
2687   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2688   \let\bbl@ifrestoring\@firstoftwo
2689   \expandafter\bbl@set@language\expandafter{\languagename}%
2690   \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.

```
2691 \chardef\localeid\z@
2692 \def\bbl@id@last{0} % No real need for a new counter
2693 \def\bbl@id@assign{%
2694   \bbl@ifunset{bbl@id@\languagename}%
```

```

2695   {\count@bbl@id@last\relax
2696   \advance\count@ne
2697   \bbl@csarg\chardef{id@\languagename}\count@
2698   \edef\bbl@id@last{\the\count@}%
2699   \ifcase\bbl@engine\or
2700   \directlua{
2701     Babel = Babel or {}
2702     Babel.locale_props = Babel.locale_props or {}
2703     Babel.locale_props[\bbl@id@last] = {}
2704   }%
2705   \fi}%
2706   {}}

```

The unprotected part of `\selectlanguage`.

```

2707 \expandafter\def\csname selectlanguage \endcsname#1{%
2708   \ifnum\bbl@hymapsel=@cclv\let\bbl@hymapsel\tw\fi
2709   \bbl@push@language
2710   \aftergroup\bbl@pop@language
2711   \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 `\languagename` 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.

```

2712 \def\BabelContentsFiles{toc,lof,lot}
2713 \def\bbl@set@language#1{% from selectlanguage, pop@
2714   \edef\languagename{%
2715     \ifnum\escapechar=\expandafter`string#1\@empty
2716     \else\string#1\@empty\fi}%
2717   \select@language{\languagename}%
2718   % write to aux
2719   \expandafter\ifx\csname date\languagename\endcsname\relax\else
2720     \if@filesw
2721       \protected@write\auxout{{}\string\babel@aux{\languagename}}}%
2722       \bbl@usehooks{write}}}%
2723   \fi
2724   \fi}
2725 \def\select@language#1{% from set@, babel@aux
2726   % set hmap
2727   \ifnum\bbl@hymapsel=@cclv\chardef\bbl@hymapsel4\relax\fi
2728   % set name
2729   \edef\languagename{#1}%
2730   \bbl@fixname\languagename
2731   \bbl@iflanguage\languagename{%
2732     \expandafter\ifx\csname date\languagename\endcsname\relax
2733     \bbl@error
2734     {Unknown language `#1'. Either you have\\%
2735     misspelled its name, it has not been installed,\\%
2736     or you requested it in a previous run. Fix its name,\\%
2737     install it or just rerun the file, respectively. In\\%
2738     some cases, you may need to remove the aux file}%
2739     {You may proceed, but expect wrong results}%
2740   \else
2741     % set type

```

```

2742     \let\bbl@select@type\z@
2743     \expandafter\bbl@switch\expandafter{\language}%
2744     \fi}}
2745 \def\babel@aux#1#2{%
2746   \expandafter\ifx\csname date#1\endcsname\relax
2747     \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
2748       \@namedef{bbl@auxwarn@#1}{}%
2749       \bbl@warning
2750       {Unknown language `#1'. Very likely you\%
2751        requested it in a previous run. Expect some\%
2752        wrong results in this run, which should vanish\%
2753        in the next one. Reported}%
2754     \fi
2755   \else
2756     \select@language{#1}%
2757     \bbl@foreach\BabelContentsFiles{%
2758       \@writefile{##1}{\babel@toc{#1}{#2}}}% %% TODO - ok in plain?
2759   \fi}
2760 \def\babel@toc#1#2{%
2761   \select@language{#1}}

```

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

```

2762 \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.

```

2763 \newif\ifbbl@usedategroup
2764 \def\bbl@switch#1{% from select@, foreign@
2765   % restore
2766   \originalTeX
2767   \expandafter\def\expandafter\originalTeX\expandafter{%
2768     \csname noextras#1\endcsname
2769     \let\originalTeX\@empty
2770     \babel@beginsave}%
2771   \bbl@usehooks{afterreset}{}%
2772   \languageshortands{none}%
2773   % set the locale id
2774   \bbl@id@assign
2775   \chardef\localeid\@nameuse{bbl@id@\@language}%
2776   % switch captions, date
2777   \ifcase\bbl@select@type
2778     \ifhmode
2779       \hskip\z@skip % trick to ignore spaces
2780     \csname captions#1\endcsname\relax

```

```

2781     \csname date#1\endcsname\relax
2782     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2783     \else
2784     \csname captions#1\endcsname\relax
2785     \csname date#1\endcsname\relax
2786     \fi
2787 \else
2788     \ifbbl@usedategroup % if \foreign... within \<lang>date
2789     \bbl@usedategroupfalse
2790     \ifhmode
2791     \hskip\z@skip % trick to ignore spaces
2792     \csname date#1\endcsname\relax
2793     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2794     \else
2795     \csname date#1\endcsname\relax
2796     \fi
2797     \fi
2798 \fi
2799 % switch extras
2800 \bbl@usehooks{beforeextras}{}%
2801 \csname extras#1\endcsname\relax
2802 \bbl@usehooks{afterextras}{}%
2803 % > babel-ensure
2804 % > babel-sh-<short>
2805 % > babel-bidi
2806 % > babel-fontspec
2807 % hyphenation - case mapping
2808 \ifcase\bbl@opt@hyphenmap\or
2809 \def\BabelLower##1##2{\lcode##1=##2\relax}%
2810 \ifnum\bbl@hymapsel>4\else
2811 \csname\languagenam @bbl@hyphenmap\endcsname
2812 \fi
2813 \chardef\bbl@opt@hyphenmap\z@
2814 \else
2815 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2816 \csname\languagenam @bbl@hyphenmap\endcsname
2817 \fi
2818 \fi
2819 \global\let\bbl@hymapsel\@cclv
2820 % hyphenation - patterns
2821 \bbl@patterns{#1}%
2822 % hyphenation - mins
2823 \babel@savevariable\lefthyphenmin
2824 \babel@savevariable\righthyphenmin
2825 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2826 \set@hyphenmins\tw@\thr@\relax
2827 \else
2828 \expandafter\expandafter\expandafter\set@hyphenmins
2829 \csname #1hyphenmins\endcsname\relax
2830 \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.

```
2831 \long\def\otherlanguage#1{%
```

```

2832 \ifnum\bb1@hymapsel=\@cclv\let\bb1@hymapsel\thr@@\fi
2833 \csname selectlanguage \endcsname{#1}%
2834 \ignorespaces}

```

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

```

2835 \long\def\endotherlanguage{%
2836 \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`.

```

2837 \expandafter\def\csname otherlanguage*\endcsname#1{%
2838 \ifnum\bb1@hymapsel=\@cclv\chardef\bb1@hymapsel4\relax\fi
2839 \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”.

```

2840 \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`.

`\bb1@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`.

```

2841 \providecommand\bb1@beforeforeign{}
2842 \edef\foreignlanguage{%
2843 \noexpand\protect
2844 \expandafter\noexpand\csname foreignlanguage \endcsname}
2845 \expandafter\def\csname foreignlanguage \endcsname{%
2846 \@ifstar\bb1@foreign@s\bb1@foreign@x}
2847 \def\bb1@foreign@x#1#2{%
2848 \begingroup
2849 \let\BabelText\@firstofone
2850 \bb1@beforeforeign
2851 \foreign@language{#1}%
2852 \bb1@usehooks{foreign}{}%
2853 \BabelText{#2}% Now in horizontal mode!

```

```

2854 \endgroup}
2855 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
2856 \begingroup
2857   {\par}%
2858   \let\BabelText\@firstofone
2859   \foreign@language{#1}%
2860   \bbl@usehooks{foreign*}{}%
2861   \bbl@dirparastext
2862   \BabelText{#2}% Still in vertical mode!
2863   {\par}%
2864 \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`.

```

2865 \def\foreign@language#1{%
2866   % set name
2867   \edef\languagename{#1}%
2868   \bbl@fixname\languagename
2869   \bbl@iflanguage\languagename{%
2870     \expandafter\ifx\csname date\languagename\endcsname\relax
2871     \bbl@warning % TODO - why a warning, not an error?
2872       {Unknown language `#1'. Either you have\\%
2873         misspelled its name, it has not been installed,\\%
2874         or you requested it in a previous run. Fix its name,\\%
2875         install it or just rerun the file, respectively. In\\%
2876         some cases, you may need to remove the aux file.\\%
2877         I'll proceed, but expect wrong results.\\%
2878         Reported}%
2879     \fi
2880     % set type
2881     \let\bbl@select@type\@ne
2882     \expandafter\bbl@switch\expandafter{\languagename}}

```

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

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

```

2883 \let\bbl@hyphlist\@empty
2884 \let\bbl@hyphenation@\relax
2885 \let\bbl@pttnlist\@empty
2886 \let\bbl@patterns@\relax
2887 \let\bbl@hymapsel=\@ccclv
2888 \def\bbl@patterns#1{%
2889   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2890     \csname l@#1\endcsname
2891     \edef\bbl@tempa{#1}%
2892   \else
2893     \csname l@#1:\f@encoding\endcsname
2894     \edef\bbl@tempa{#1:\f@encoding}%
2895   \fi
2896   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%

```

```

2897 % > luatex
2898 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
2899   \begingroup
2900     \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
2901     \ifin@else
2902       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
2903       \hyphenation{%
2904         \bbl@hyphenation@
2905         \@ifundefined{bbl@hyphenation@#1}%
2906           \empty
2907           {\space\csname bbl@hyphenation@#1\endcsname}}%
2908       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2909       \fi
2910   \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*.

```

2911 \def\hyphenrules#1{%
2912   \edef\bbl@tempf{#1}%
2913   \bbl@fixname\bbl@tempf
2914   \bbl@iflanguage\bbl@tempf{%
2915     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2916     \languageshortands{none}%
2917     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2918       \set@hyphenmins\tw@\thr@\relax
2919     \else
2920       \expandafter\expandafter\expandafter\set@hyphenmins
2921       \csname\bbl@tempf hyphenmins\endcsname\relax
2922     \fi}}
2923 \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.

```

2924 \def\providehyphenmins#1#2{%
2925   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2926     \@namedef{#1hyphenmins}{#2}%
2927   \fi}

```

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

```

2928 \def\set@hyphenmins#1#2{%
2929   \lefthyphenmin#1\relax
2930   \righthyphenmin#2\relax}

```

\ProvidesLanguage The identification code for each file is something that was introduced in $\LaTeX 2_{\epsilon}$. 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.

```

2931 \ifx\ProvidesFile\@undefined
2932   \def\ProvidesLanguage#1[#2 #3 #4]{%
2933     \wlog{Language: #1 #4 #3 <#2>}%
2934   }
2935 \else
2936   \def\ProvidesLanguage#1{%

```



```

2937 \begingroup
2938 \catcode`\ 10 %
2939 \@makeother\%
2940 \@ifnextchar[%]
2941   {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
2942 \def\@provideslanguage#1[#2]{%
2943   \wlog{Language: #1 #2}%
2944   \expandafter\edef\csname ver@#1.ldf\endcsname{#2}%
2945   \endgroup}
2946 \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`

```

2947 \def\LdfInit{%
2948   \chardef\atcatcode=\catcode`\@
2949   \catcode`\@=11\relax
2950   \input babel.def\relax
2951   \catcode`\@=\atcatcode \let\atcatcode\relax
2952   \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`.

```
2953 \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.

```
2954 \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’:

```

2955 \providecommand\setlocale{%
2956   \bbl@error
2957   {Not yet available}%
2958   {Find an armchair, sit down and wait}}
2959 \let\uselocale\setlocale
2960 \let\locale\setlocale
2961 \let\selectlocale\setlocale
2962 \let\textlocale\setlocale
2963 \let\textlanguage\setlocale
2964 \let\languagetext\setlocale

```

11.2 Errors

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

`\@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'.

```

2965 \edef\bbl@nulllanguage{\string\language=0}
2966 \ifx\PackageError\@undefined
2967   \def\bbl@error#1#2{%
2968     \begingroup
2969       \newlinechar=`^^J
2970       \def\{^^J(babel) }%
2971       \errhelp{#2}\errmessage{\#1}%
2972     \endgroup}
2973 \def\bbl@warning#1{%
2974   \begingroup
2975     \newlinechar=`^^J
2976     \def\{^^J(babel) }%
2977     \message{\#1}%
2978   \endgroup}
2979 \def\bbl@info#1{%
2980   \begingroup
2981     \newlinechar=`^^J
2982     \def\{^^J}%
2983     \wlog{#1}%
2984   \endgroup}
2985 \else
2986   \def\bbl@error#1#2{%
2987     \begingroup
2988       \def\{\MessageBreak}%
2989       \PackageError{babel}{#1}{#2}%
2990     \endgroup}
2991   \def\bbl@warning#1{%
2992     \begingroup
2993       \def\{\MessageBreak}%
2994       \PackageWarning{babel}{#1}%
2995     \endgroup}
2996   \def\bbl@info#1{%
2997     \begingroup
2998       \def\{\MessageBreak}%
2999       \PackageInfo{babel}{#1}%
3000     \endgroup}
3001 \fi
3002 \@ifpackagewith{babel}{silent}
3003   {\let\bbl@info@gobble
3004    \let\bbl@warning@gobble}
3005   {}
3006 \def\bbl@nocaption{\protect\bbl@nocaption@i}
3007 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
3008   \global\@namedef{#2}{\textbf{#1?}}%
3009   \@nameuse{#2}%
3010   \bbl@warning{%
3011     \@backslashchar#2 not set. Please, define\%
3012     it in the preamble with something like:\%
3013     \string\renewcommand\@backslashchar#2{..\}%
3014     Reported}}
3015 \def\bbl@tentative{\protect\bbl@tentative@i}
3016 \def\bbl@tentative@i#1{%
3017   \bbl@warning{%
3018     Some functions for '#1' are tentative.\%
3019     They might not work as expected and their behavior\%
3020     could change in the future.\%

```

```

3021   Reported}}
3022 \def\nolanerr#1{%
3023   \bbl@error
3024   {You haven't defined the language #1\space yet}%
3025   {Your command will be ignored, type <return> to proceed}}
3026 \def\nopatterns#1{%
3027   \bbl@warning
3028   {No hyphenation patterns were preloaded for\%
3029     the language `#1' into the format.\%
3030     Please, configure your TeX system to add them and\%
3031     rebuild the format. Now I will use the patterns\%
3032     preloaded for \bbl@nulllanguage\space instead}}
3033 \let\bbl@usehooks\@gobbletwo
3034 </kernel>
3035 <*patterns>

```

12 Loading hyphenation patterns

The following code is meant to be read by `iniTeX` because it should instruct `TeX` to read hyphenation patterns. To this end the `docstrip` option `patterns` can be used to include this code in the file `hyphen.cfg`. Code is written with lower level macros. We want to add a message to the message `LTEX 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 `LTEX` fills the register. There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with `SLTEX` the above scheme won't work. The reason is that `SLTEX` overwrites the contents of the `\everyjob` register with its own message.
- Plain `TeX` does not use the `\everyjob` register so the message would not be displayed.

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, `\dump`. Therefore the original `\dump` is saved in `\org@dump` and a new definition is supplied.

To make sure that `LTEX 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.

```

3036 <<Make sure ProvidesFile is defined>>
3037 \ProvidesFile{hyphen.cfg}[<<date>>] <<version>> Babel hyphens]
3038 \xdef\bbl@format{\jobname}

```

```

3039 \ifx\AtBeginDocument\@undefined
3040 \def\@empty{}
3041 \let\orig@dump\dump
3042 \def\dump{%
3043   \ifx\@ztryfc\@undefined
3044   \else
3045     \toks0=\expandafter{\@preamblecmds}%
3046     \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3047     \def\@begindocumenthook{}}%
3048   \fi
3049   \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3050 \fi
3051 <<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.

```

3052 \def\process@line#1#2 #3 #4 {%
3053   \ifx=#1%
3054     \process@synonym{#2}%
3055   \else
3056     \process@language{#1#2}{#3}{#4}%
3057   \fi
3058   \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.

```

3059 \toks@{}
3060 \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.

```

3061 \def\process@synonym#1{%
3062   \ifnum\last@language=\m@ne
3063     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3064   \else
3065     \expandafter\chardef\csname l@#1\endcsname\last@language
3066     \wlog{\string\l@#1=\string\language\the\last@language}%
3067     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3068       \csname\languagename hyphenmins\endcsname
3069     \let\bbl@elt\relax
3070     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}}%
3071   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read. For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the

name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\langle lang \rangle hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form `\bbl@elt{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. 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.

```

3072 \def\process@language#1#2#3{%
3073   \expandafter\addlanguage\csname l@#1\endcsname
3074   \expandafter\language\csname l@#1\endcsname
3075   \edef\languagename{#1}%
3076   \bbl@hook@everylanguage{#1}%
3077   % > luatex
3078   \bbl@get@enc#1::\@@@
3079   \begingroup
3080     \lefthyphenmin\m@ne
3081     \bbl@hook@loadpatterns{#2}%
3082     % > luatex
3083     \ifnum\lefthyphenmin=\m@ne
3084     \else
3085       \expandafter\xdef\csname #1hyphenmins\endcsname{%
3086         \the\lefthyphenmin\the\righthyphenmin}%
3087     \fi
3088   \endgroup
3089   \def\bbl@tempa{#3}%
3090   \ifx\bbl@tempa\@empty\else
3091     \bbl@hook@loadexceptions{#3}%
3092     % > luatex
3093   \fi
3094   \let\bbl@elt\relax
3095   \edef\bbl@languages{%
3096     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3097   \ifnum\the\language=\z@
3098     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3099     \set@hyphenmins\tw@\thr@@\relax
3100   \else
3101     \expandafter\expandafter\expandafter\set@hyphenmins
3102     \csname #1hyphenmins\endcsname
3103   \fi
3104   \the\toks@
3105   \toks@{}%

```

```
3106 \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.

```
3107 \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.

```
3108 \def\bbl@hook@everylanguage#1{}
3109 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3110 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3111 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
3112 \begingroup
3113 \def\AddBabelHook#1#2{%
3114 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3115 \def\next{\toks1}%
3116 \else
3117 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
3118 \fi
3119 \next}
3120 \ifx\directlua\undefined
3121 \ifx\XeTeXinputencoding\undefined\else
3122 \input xebabel.def
3123 \fi
3124 \else
3125 \input luababel.def
3126 \fi
3127 \openin1 = babel-\bbl@format.cfg
3128 \ifeof1
3129 \else
3130 \input babel-\bbl@format.cfg\relax
3131 \fi
3132 \closein1
3133 \endgroup
3134 \bbl@hook@loadkernel{switch.def}
```

`\readconfigfile` The configuration file can now be opened for reading.

```
3135 \openin1 = language.dat
```

See if the file exists, if not, use the default hyphenation file `hyphen.tex`. The user will be informed about this.

```
3136 \def\languagename{english}%
3137 \ifeof1
3138 \message{I couldn't find the file language.dat,\space
3139 I will try the file hyphen.tex}
3140 \input hyphen.tex\relax
3141 \chardef\l@english\z@
3142 \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`.

```
3143 \last@language\m@ne
```

We now read lines from the file until the end is found

```
3144 \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.

```
3145 \endlinechar\m@ne
3146 \read1 to \bbl@line
3147 \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.

```
3148 \if T\ifeof1F\fi T\relax
3149 \ifx\bbl@line\@empty\else
3150 \edef\bbl@line{\bbl@line\space\space\space}%
3151 \expandafter\process@line\bbl@line\relax
3152 \fi
3153 \repeat
```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns.

```
3154 \begingroup
3155 \def\bbl@elt#1#2#3#4{%
3156 \global\language=#2\relax
3157 \gdef\language#1}%
3158 \def\bbl@elt##1##2##3##4{}}%
3159 \bbl@languages
3160 \endgroup
3161 \fi
```

and close the configuration file.

```
3162 \closein1
```

We add a message about the fact that babel is loaded in the format and with which language patterns to the `\everyjob` register.

```
3163 \if/\the\toks@/\else
3164 \errhelp{language.dat loads no language, only synonyms}
3165 \errmessage{Orphan language synonym}
3166 \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.

```
3167 \let\bbl@line\@undefined
3168 \let\process@line\@undefined
3169 \let\process@synonym\@undefined
3170 \let\process@language\@undefined
3171 \let\bbl@get@enc\@undefined
3172 \let\bbl@hyph@enc\@undefined
3173 \let\bbl@tempa\@undefined
3174 \let\bbl@hook@loadkernel\@undefined
3175 \let\bbl@hook@everylanguage\@undefined
3176 \let\bbl@hook@loadpatterns\@undefined
3177 \let\bbl@hook@loadexceptions\@undefined
3178 </patterns>
```

Here the code for `iniTeX` ends.

13 Font handling with fontspec

Add the bidi handler just before luaoffload, 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].

```
3179 <<{*More package options}>> ≡
3180 \ifodd\bbl@engine
3181 \DeclareOption{bidi=basic-r}%
3182   {\ExecuteOptions{bidi=basic}}
3183 \DeclareOption{bidi=basic}%
3184   {\let\bbl@beforeforeign\leavevmode
3185    % TODO - to locale_props, not as separate attribute
3186    \newattribute\bbl@attr@dir
3187    % I don't like it, hackish:
3188    \frozen@everymath\expandafter{%
3189     \expandafter\bbl@mathboxdir\the\frozen@everymath}%
3190    \frozen@everydisplay\expandafter{%
3191     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}%
3192    \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
3193 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
3194 \else
3195 \DeclareOption{bidi=basic-r}%
3196   {\ExecuteOptions{bidi=basic}}
3197 \DeclareOption{bidi=basic}%
3198   {\bbl@error
3199    {The bidi method `basic' is available only in\\%
3200     luatex. I'll continue with `bidi=default', so\\%
3201     expect wrong results}%
3202    {See the manual for further details.}%
3203    \let\bbl@beforeforeign\leavevmode
3204    \AtEndOfPackage{%
3205     \EnableBabelHook{babel-bidi}%
3206     \bbl@xebidipar}}
3207 \def\bbl@loadxebidi#1{%
3208   \ifx\RTLfootnotetext\@undefined
3209     \AtEndOfPackage{%
3210       \EnableBabelHook{babel-bidi}%
3211       \ifx\fontspec\@undefined
3212         \usepackage{fontspec}% bidi needs fontspec
3213       \fi
3214       \usepackage#1{bidi}}%
3215   \fi}
3216 \DeclareOption{bidi=bidi}%
3217   {\bbl@tentative{bidi=bidi}%
3218   \bbl@loadxebidi{}}
3219 \DeclareOption{bidi=bidi-r}%
3220   {\bbl@tentative{bidi=bidi-r}%
3221   \bbl@loadxebidi{[rldocument]}}
3222 \DeclareOption{bidi=bidi-l}%
3223   {\bbl@tentative{bidi=bidi-l}%
3224   \bbl@loadxebidi{}}
3225 \fi
3226 \DeclareOption{bidi=default}%
3227   {\let\bbl@beforeforeign\leavevmode
3228   \ifodd\bbl@engine
3229     \newattribute\bbl@attr@dir
3230     \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
3231 \fi
```



```

3232 \AtEndOfPackage{%
3233   \EnableBabelHook{babel-bidi}%
3234   \ifodd\bbbl@engine\else
3235     \bbbl@xebidipar
3236     \fi}}
3237 <</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.

```

3238 <<{*Font selection}>> ≡
3239 \bbbl@trace{Font handling with fontspec}
3240 \@onlypreamble\babelfont
3241 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
3242   \edef\bbbl@tempa{#1}%
3243   \def\bbbl@tempb{#2}%
3244   \ifx\fontspec\undefined
3245     \usepackage{fontspec}%
3246     \fi
3247   \EnableBabelHook{babel-fontspec}% Just calls \bbbl@switchfont
3248   \bbbl@bbfont}
3249 \newcommand\bbbl@bbfont[2][]{% 1=features 2=fontname
3250   \bbbl@ifunset{\bbbl@tempb family}{\bbbl@providefam{\bbbl@tempb}}{}}%
3251 % For the default font, just in case:
3252 \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\languagename}}{}}%
3253 \expandafter\bbbl@ifblank\expandafter{\bbbl@tempa}%
3254   {\bbbl@csarg\edef{\bbbl@tempb dflt@}{<{#1}{#2}}% save \bbbl@rmdflt@
3255   \bbbl@exp{%
3256     \let\<\bbbl@bbfont@tempb dflt@\languagename>\<\bbbl@bbfont@tempb dflt@>%
3257     \bbbl@font@set\<\bbbl@bbfont@tempb dflt@\languagename>%
3258     \<\bbbl@tempb default>\<\bbbl@tempb family>}}%
3259   {\bbbl@foreach\bbbl@tempa{% ie \bbbl@rmdflt@lang / *schr
3260     \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:

```

3261 \def\bbbl@providefam#1{%
3262   \bbbl@exp{%
3263     \newcommand\<#1default>{}% Just define it
3264     \bbbl@add@list\bbbl@font@fams{#1}%
3265     \DeclareRobustCommand\<#1family>%
3266     \not@math@alphabet\<#1family>\relax
3267     \fontfamily\<#1default>\selectfont}%
3268     \DeclareTextFontCommand{\<text#1>}{\<#1family>}}

```

The following macro is activated when the hook babel-fontspec is enabled.

```

3269 \def\bbbl@switchfont{%
3270   \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\languagename}}{}}%
3271   \bbbl@exp{% eg Arabic -> arabic
3272     \lowercase{\edef\bbbl@tempa{\bbbl@cs{sname@\languagename}}}}%
3273   \bbbl@foreach\bbbl@font@fams{%
3274     \bbbl@ifunset{\bbbl@##1dflt@\languagename}% (1) language?
3275     {\bbbl@ifunset{\bbbl@##1dflt*\bbbl@tempa}% (2) from script?
3276     {\bbbl@ifunset{\bbbl@##1dflt@}% 2=F - (3) from generic?
3277     {}}% 123=F - nothing!
3278     {\bbbl@exp{% 3=T - from generic
3279       \global\let\<\bbbl@##1dflt@\languagename>%
3280       \<\bbbl@##1dflt@>}}}%
3281     {\bbbl@exp{% 2=T - from script
3282       \global\let\<\bbbl@##1dflt@\languagename>%
3283       \<\bbbl@##1dflt*\bbbl@tempa>}}}}}%

```

```

3284     {}}%                               1=T - language, already defined
3285 \def\bb1@tempa{%
3286   \bb1@warning{The current font is not a standard family:\%
3287     \fontname\font\%
3288     Script and Language are not applied. Consider\%
3289     defining a new family with \string\babelfont.\%
3290     Reported}}%
3291 \bb1@foreach\bb1@font@fams{%   don't gather with prev for
3292   \bb1@ifunset{bb1@##1dflt@\languagename}%
3293     {\bb1@cs{famrst@##1}%
3294       \global\bb1@csarg\let{famrst@##1}\relax}%
3295     {\bb1@exp{% order is relevant
3296       \\bb1@add\\originalTeX{%
3297         \\bb1@font@rst{\bb1@cs{##1dflt@\languagename}}%
3298           \<##1default>\<##1family>{##1}}%
3299         \\bb1@font@set\<bb1@##1dflt@\languagename>% the main part!
3300           \<##1default>\<##1family>}}}%
3301   \bb1@ifrestoring{}{\bb1@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 \bb1@mapselect because \selectfont is called internally when a font is defined.

```

3302 \def\bb1@font@set#1#2#3{% eg \bb1@rmdflt@lang \rmdefault \rmfamily
3303   \bb1@xin@{<>}{#1}%
3304   \ifin@
3305     \bb1@exp{\bb1@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
3306     \fi
3307   \bb1@exp{%
3308     \def\#2{#1}%           eg, \rmdefault{\bb1@rmdflt@lang}
3309     \\bb1@ifsamestring{#2}{\f@family}{\#3\let\\bb1@tempa\relax}{}}%
3310 % TODO - next should be global?, but even local does its job. I'm
3311 % still not sure -- must investigate:
3312 \def\bb1@fontspec@set#1#2#3#4{% eg \bb1@rmdflt@lang fnt-opt fnt-nme \xxfamily
3313   \let\bb1@tempa\bb1@mapselect
3314   \let\bb1@mapselect\relax
3315   \let\bb1@temp@fam#4%     eg, '\rmfamily', to be restored below
3316   \let#4\relax           % So that can be used with \newfontfamily
3317   \bb1@exp{%
3318     \let\\bb1@temp@pfam\<\bb1@stripslash#4\space>% eg, '\rmfamily '
3319     \<keys_if_exist:nnF>{fontspec-opentype}%
3320       {Script/\bb1@cs{sname@\languagename}}%
3321       {\newfontscript{\bb1@cs{sname@\languagename}}%
3322         {\bb1@cs{sotf@\languagename}}}%
3323     \<keys_if_exist:nnF>{fontspec-opentype}%
3324       {Language/\bb1@cs{lname@\languagename}}%
3325       {\newfontlanguage{\bb1@cs{lname@\languagename}}%
3326         {\bb1@cs{lotf@\languagename}}}%
3327     \\newfontfamily\#4%
3328     [\bb1@cs{lsys@\languagename},#2]}{#3}% ie \bb1@exp{.}{#3}
3329   \begingroup
3330     #4%
3331     \xdef#1{\f@family}%     eg, \bb1@rmdflt@lang{FreeSerif(0)}
3332   \endgroup
3333   \let#4\bb1@temp@fam
3334   \bb1@exp{\let\<\bb1@stripslash#4\space>\bb1@temp@pfam}
3335   \let\bb1@mapselect\bb1@tempa}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```
3336 \def\bb1@font@rst#1#2#3#4{%
3337 \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.

```
3338 \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 :-).

```
3339 \newcommand\babelFSstore[2][]{%
3340 \bb1@ifblank{#1}%
3341 {\bb1@csarg\def{sname@#2}{Latin}}%
3342 {\bb1@csarg\def{sname@#2}{#1}}%
3343 \bb1@provide@dirs{#2}%
3344 \bb1@csarg\ifnum{wdir@#2}>\z@
3345 \let\bb1@beforeforeign\leavevmode
3346 \EnableBabelHook{babel-bidi}%
3347 \fi
3348 \bb1@foreach{#2}{%
3349 \bb1@FSstore{##1}{rm}\rmdefault\bb1@save@rmdefault
3350 \bb1@FSstore{##1}{sf}\sfdefault\bb1@save@sfdefault
3351 \bb1@FSstore{##1}{tt}\ttdefault\bb1@save@ttdefault}}
3352 \def\bb1@FSstore#1#2#3#4{%
3353 \bb1@csarg\edef{#2default#1}{#3}%
3354 \expandafter\addto\csname extras#1\endcsname{%
3355 \let#4#3%
3356 \ifx#3\f@family
3357 \edef#3{\csname bbl@#2default#1\endcsname}%
3358 \fontfamily{#3}\selectfont
3359 \else
3360 \edef#3{\csname bbl@#2default#1\endcsname}%
3361 \fi}%
3362 \expandafter\addto\csname noextras#1\endcsname{%
3363 \ifx#3\f@family
3364 \fontfamily{#4}\selectfont
3365 \fi
3366 \let#3#4}}
3367 \let\bb1@langfeatures\@empty
3368 \def\babelFSfeatures{% make sure \fontspec is redefined once
3369 \let\bb1@ori@fontspec\fontspec
3370 \renewcommand\fontspec[1][]{%
3371 \bb1@ori@fontspec[\bb1@langfeatures##1]}
3372 \let\babelFSfeatures\bb1@FSfeatures
3373 \babelFSfeatures}
3374 \def\bb1@FSfeatures#1#2{%
3375 \expandafter\addto\csname extras#1\endcsname{%
3376 \babel@save\bb1@langfeatures
3377 \edef\bb1@langfeatures{#2,}}
3378 <</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 `xetex` they must be reset to the proper value. Most of the work is done in `xe(la)tex.ini`, so here we just “undo” some of the changes done by `ℒTeX`. Anyway, for consistency `luaTeX` also resets the catcodes.

```
3379 <<{*Restore Unicode catcodes before loading patterns}>> ≡
3380 \begingroup
3381 % Reset chars "80-"C0 to category "other", no case mapping:
3382 \catcode\@=11 \count@=128
3383 \loop\ifnum\count@<192
3384 \global\uccode\count@=0 \global\lccode\count@=0
3385 \global\catcode\count@=12 \global\sffcode\count@=1000
3386 \advance\count@ by 1 \repeat
3387 % Other:
3388 \def\O ##1 {%
3389 \global\uccode"##1=0 \global\lccode"##1=0
3390 \global\catcode"##1=12 \global\sffcode"##1=1000 }%
3391 % Letter:
3392 \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3393 \global\uccode"##1="##2
3394 \global\lccode"##1="##3
3395 % Uppercase letters have sffcode=999:
3396 \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
3397 % Letter without case mappings:
3398 \def\l ##1 {\L ##1 ##1 ##1 }%
3399 \l 00AA
3400 \L 00B5 039C 00B5
3401 \l 00BA
3402 \O 00D7
3403 \l 00DF
3404 \O 00F7
3405 \L 00FF 0178 00FF
3406 \endgroup
3407 \input #1\relax
3408 <</Restore Unicode catcodes before loading patterns>>
```

Some more common code.

```
3409 <<{*Footnote changes}>> ≡
3410 \bbl@trace{Bidi footnotes}
3411 \ifx\bbl@beforeforeign\leavevmode
3412 \def\bbl@footnote#1#2#3{%
3413 \@ifnextchar[%
3414 {\bbl@footnote@o{#1}{#2}{#3}}%
3415 {\bbl@footnote@x{#1}{#2}{#3}}}
3416 \def\bbl@footnote@x#1#2#3#4{%
3417 \bgroup
3418 \select@language@x{\bbl@main@language}%
3419 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3420 \egroup}
3421 \def\bbl@footnote@o#1#2#3[#4]#5{%
3422 \bgroup
3423 \select@language@x{\bbl@main@language}%
3424 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%

```

```

3425 \egroup}
3426 \def\bbl@footnotetext#1#2#3{%
3427 \@ifnextchar[%
3428 {\bbl@footnotetext@o{#1}{#2}{#3}}%
3429 {\bbl@footnotetext@x{#1}{#2}{#3}}}
3430 \def\bbl@footnotetext@x#1#2#3#4{%
3431 \bgroup
3432 \select@language@x{\bbl@main@language}%
3433 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3434 \egroup}
3435 \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3436 \bgroup
3437 \select@language@x{\bbl@main@language}%
3438 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3439 \egroup}
3440 \def\BabelFootnote#1#2#3#4{%
3441 \ifx\bbl@fn@footnote\@undefined
3442 \let\bbl@fn@footnote\footnote
3443 \fi
3444 \ifx\bbl@fn@footnotetext\@undefined
3445 \let\bbl@fn@footnotetext\footnotetext
3446 \fi
3447 \bbl@ifblank{#2}%
3448 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3449 \@namedef{\bbl@stripslash#1text}%
3450 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3451 {\def#1{\bbl@exp{\@firstofone}{\@firstofone}{#2}}{#3}{#4}}%
3452 \@namedef{\bbl@stripslash#1text}%
3453 {\bbl@exp{\@firstofone}{\@firstofone}{#2}}{#3}{#4}}}}
3454 \fi
3455 <</Footnote changes>>

```

Now, the code.

```

3456 (*xetex)
3457 \def\BabelStringsDefault{unicode}
3458 \let\xebbl@stop\relax
3459 \AddBabelHook{xetex}{encodedcommands}{%
3460 \def\bbl@tempa{#1}%
3461 \ifx\bbl@tempa\@empty
3462 \XeTeXinputencoding"bytes"%
3463 \else
3464 \XeTeXinputencoding"#1"%
3465 \fi
3466 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3467 \AddBabelHook{xetex}{stopcommands}{%
3468 \xebbl@stop
3469 \let\xebbl@stop\relax}
3470 \def\bbl@intraspace#1 #2 #3\@@{%
3471 \bbl@csarg\gdef{\xeisp@\bbl@cs{sbcpr@languagename}}%
3472 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3473 \def\bbl@intrapenalty#1\@@{%
3474 \bbl@csarg\gdef{\xeipn@\bbl@cs{sbcpr@languagename}}%
3475 {\XeTeXlinebreakpenalty #1\relax}}
3476 \AddBabelHook{xetex}{loadkernel}{%
3477 <<Restore Unicode catcodes before loading patterns>>}
3478 \ifx\DisableBabelHook\@undefined\endinput\fi
3479 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3480 \DisableBabelHook{babel-fontspec}
3481 <<Font selection>>

```

```

3482 \input txtbabel.def
3483 \xetex

```

14.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titlesp, 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 pdf_{te}x and xetex.

```

3484 (*texxet)
3485 \bbl@trace{Redefinitions for bidi layout}
3486 \def\bbl@sspre@caption{%
3487   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
3488 \ifx\bbl@opt@layout\@nil\endinput\fi % No layout
3489 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3490 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3491 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3492   \def\@hangfrom#1{%
3493     \setbox\@tempboxa\hbox{#1}%
3494     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3495     \noindent\box\@tempboxa}
3496 \def\raggedright{%
3497   \let\@centercr
3498   \bbl@startskip\z@skip
3499   \@rightskip\@flushglue
3500   \bbl@endskip\@rightskip
3501   \parindent\z@
3502   \parfillskip\bbl@startskip}
3503 \def\raggedleft{%
3504   \let\@centercr
3505   \bbl@startskip\@flushglue
3506   \bbl@endskip\z@skip
3507   \parindent\z@
3508   \parfillskip\bbl@endskip}
3509 \fi
3510 \IfBabelLayout{lists}
3511   {\def\list#1#2{%
3512     \ifnum \@listdepth >5\relax
3513       \@toodeep
3514     \else
3515       \global\advance\@listdepth\@ne
3516     \fi
3517     \rightmargin\z@
3518     \listparindent\z@
3519     \itemindent\z@
3520     \csname @list\romannumeral\the\@listdepth\endcsname
3521     \def\@itemlabel{#1}%
3522     \let\makelabel\@mklab
3523     \@nmbrolistfalse
3524     #2\relax
3525     \@trivlist
3526     \parskip\parsep
3527     \parindent\listparindent

```

```

3528 \advance\linewidth-\rightmargin
3529 \advance\linewidth-\leftmargin
3530 \advance\@totalleftmargin
3531 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi
3532 \parshape\@ne\@totalleftmargin\linewidth
3533 \ignorespaces}%
3534 \ifcase\bbl@engine
3535 \def\labelenumii{}\theenumii}%
3536 \def\p@enumiii{\p@enumii}\theenumii}%
3537 \fi
3538 \def\@verbatim{%
3539 \trivlist \item\relax
3540 \if@minipage\else\vskip\parskip\fi
3541 \bbl@startskip\textwidth
3542 \advance\bbl@startskip-\linewidth
3543 \bbl@endskip\z@skip
3544 \parindent\z@
3545 \parfillskip\@flushglue
3546 \parskip\z@skip
3547 \@par
3548 \language\@nohyphenation
3549 \@tempwafalse
3550 \def\par{%
3551 \if@tempwa
3552 \leavevmode\null
3553 \@par\penalty\interlinepenalty
3554 \else
3555 \@tempwatrue
3556 \ifhmode\@par\penalty\interlinepenalty\fi
3557 \fi}%
3558 \let\do\@makeother \dospecials
3559 \obeylines \verbatim@font \@noligs
3560 \everypar\expandafter{\the\everypar\unpenalty}}
3561 {}
3562 \IfBabelLayout{contents}
3563 {\def\@dottedtocline#1#2#3#4#5{%
3564 \ifnum#1>\c@tocdepth\else
3565 \vskip \z@ \@plus.2\p@
3566 {\bbl@startskip#2\relax
3567 \bbl@endskip\@tocrmarg
3568 \parfillskip-\bbl@endskip
3569 \parindent#2\relax
3570 \@afterindenttrue
3571 \interlinepenalty\M
3572 \leavevmode
3573 \@tempdima#3\relax
3574 \advance\bbl@startskip\@tempdima
3575 \null\nobreak\hskip-\bbl@startskip
3576 {#4}\nobreak
3577 \leaders\hbox{%
3578 $m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
3579 \hfill\nobreak
3580 \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
3581 \par}%
3582 \fi}}
3583 {}
3584 \IfBabelLayout{columns}
3585 {\def\@outputdblcol{%
3586 \if@firstcolumn

```

```

3587 \global\@firstcolumnfalse
3588 \global\setbox\@leftcolumn\copy\@outputbox
3589 \splitmaxdepth\maxdimen
3590 \vbadness\maxdimen
3591 \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
3592 \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3593 \toks@\expandafter{\topmark}%
3594 \xdef\@firstcoltopmark{\the\toks@}%
3595 \toks@\expandafter{\splitfirstmark}%
3596 \xdef\@firstcolfirstmark{\the\toks@}%
3597 \ifx\@firstcolfirstmark\@empty
3598 \global\let\@setmarks\relax
3599 \else
3600 \gdef\@setmarks{%
3601 \let\firstmark\@firstcolfirstmark
3602 \let\topmark\@firstcoltopmark}%
3603 \fi
3604 \else
3605 \global\@firstcolumntrue
3606 \setbox\@outputbox\vbox{%
3607 \hb@xt@\textwidth{%
3608 \hskip\columnwidth
3609 \hfil
3610 {\normalcolor\vrule \@width\columnseprule}%
3611 \hfil
3612 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3613 \hskip-\textwidth
3614 \hb@xt@\columnwidth{\box\@outputbox \hss}%
3615 \hskip\columnsep
3616 \hskip\columnwidth}}%
3617 \@combinedblfloats
3618 \@setmarks
3619 \@outputpage
3620 \begingroup
3621 \@dblfloatplacement
3622 \@startdblcolumn
3623 \@whilesw\if@fcolmade \fi{\@outputpage
3624 \@startdblcolumn}%
3625 \endgroup
3626 \fi}}%
3627 {}
3628 <<Footnote changes>>
3629 \IfBabelLayout{footnotes}%
3630 {\BabelFootnote\footnote\languagename{}}{}%
3631 \BabelFootnote\localfootnote\languagename{}}{}%
3632 \BabelFootnote\mainfootnote{}}{}%
3633 {}

```

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.

```

3634 \IfBabelLayout{counters}%
3635 {\let\bb1@latinarabic=\@arabic
3636 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
3637 \let\bb1@asciroman=\@roman
3638 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
3639 \let\bb1@asciiRoman=\@Roman
3640 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}}{}
3641 </texet>

```


14.3 LuaTeX

The new loader for luatex is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they has been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, `lua(e)tex` is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need `catcode` tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). For the moment, a dangerous approach is used – just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

```
3642 (*luatex)
3643 \ifx\AddBabelHook\undefined
3644 \bbl@trace{Read language.dat}
3645 \begingroup
3646 \toks@{}
3647 \count@% 0=start, 1=0th, 2=normal
3648 \def\bbl@process@line#1#2 #3 #4 {%
3649   \ifx=#1%
3650     \bbl@process@synonym{#2}%
3651   \else
3652     \bbl@process@language{#1#2}{#3}{#4}%
3653   \fi
3654 \ignorespaces}
3655 \def\bbl@manylang{%
3656   \ifnum\bbl@last>\@ne
3657     \bbl@info{Non-standard hyphenation setup}%
3658   \fi
3659 \let\bbl@manylang\relax}
3660 \def\bbl@process@language#1#2#3{%
3661   \ifcase\count@
3662     \@ifundefined{zth#1}{\count@tw@}{\count@\@ne}%
3663   \or
3664     \count@tw@
3665   \fi
```

```

3666 \ifnum\count@=\tw@
3667 \expandafter\addlanguage\csname l@#1\endcsname
3668 \language\allocationnumber
3669 \chardef\bbl@last\allocationnumber
3670 \bbl@manylang
3671 \let\bbl@elt\relax
3672 \xdef\bbl@languages{%
3673 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3674 \fi
3675 \the\toks@
3676 \toks@{}}
3677 \def\bbl@process@synonym@aux#1#2{%
3678 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3679 \let\bbl@elt\relax
3680 \xdef\bbl@languages{%
3681 \bbl@languages\bbl@elt{#1}{#2}{}}}%
3682 \def\bbl@process@synonym#1{%
3683 \ifcase\count@
3684 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3685 \or
3686 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
3687 \else
3688 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3689 \fi}
3690 \ifx\bbl@languages@undefined % Just a (sensible?) guess
3691 \chardef\l@english\z@
3692 \chardef\l@USenglish\z@
3693 \chardef\bbl@last\z@
3694 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
3695 \gdef\bbl@languages{%
3696 \bbl@elt{english}{0}{\hyphen.tex}}%
3697 \bbl@elt{USenglish}{0}{}}
3698 \else
3699 \global\let\bbl@languages@format\bbl@languages
3700 \def\bbl@elt#1#2#3#4{% Remove all except language 0
3701 \ifnum#2>\z@\else
3702 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3703 \fi}%
3704 \xdef\bbl@languages{\bbl@languages}%
3705 \fi
3706 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3707 \bbl@languages
3708 \openin1=language.dat
3709 \ifeof1
3710 \bbl@warning{I couldn't find language.dat. No additional\%
3711 patterns loaded. Reported}%
3712 \else
3713 \loop
3714 \endlinechar\m@ne
3715 \read1 to \bbl@line
3716 \endlinechar\^^M
3717 \if T\ifeof1F\fi T\relax
3718 \ifx\bbl@line\empty\else
3719 \edef\bbl@line{\bbl@line\space\space\space}%
3720 \expandafter\bbl@process@line\bbl@line\relax
3721 \fi
3722 \repeat
3723 \fi
3724 \endgroup

```

```

3725 \bbl@trace{Macros for reading patterns files}
3726 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
3727 \ifx\babelcatcodetablenum\undefined
3728   \def\babelcatcodetablenum{5211}
3729 \fi
3730 \def\bbl@luapatterns#1#2{%
3731   \bbl@get@enc#1::\@@@
3732   \setbox\z@\hbox\bgroup
3733     \begingroup
3734       \ifx\catcodetable\undefined
3735         \let\savecatcodetable\luatexsavecatcodetable
3736         \let\initcatcodetable\luatexinitcatcodetable
3737         \let\catcodetable\luatexcatcodetable
3738       \fi
3739       \savecatcodetable\babelcatcodetablenum\relax
3740       \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3741       \catcodetable\numexpr\babelcatcodetablenum+1\relax
3742       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3743       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
3744       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3745       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3746       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
3747       \catcode`\`=12 \catcode`\'=12 \catcode`\ "=12
3748       \input #1\relax
3749       \catcodetable\babelcatcodetablenum\relax
3750     \endgroup
3751   \def\bbl@tempa{#2}%
3752   \ifx\bbl@tempa\@empty\else
3753     \input #2\relax
3754   \fi
3755 \egroup}%
3756 \def\bbl@patterns@lua#1{%
3757   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3758     \csname l@#1\endcsname
3759     \edef\bbl@tempa{#1}%
3760   \else
3761     \csname l@#1:\f@encoding\endcsname
3762     \edef\bbl@tempa{#1:\f@encoding}%
3763   \fi\relax
3764   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3765   \@ifundefined{bbl@hyphendata@the\language}{%
3766     {\def\bbl@elt##1##2##3##4{%
3767       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3768       \def\bbl@tempb{##3}%
3769       \ifx\bbl@tempb\@empty\else % if not a synonymous
3770         \def\bbl@tempc{##3}{##4}%
3771       \fi
3772       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3773     \fi}%
3774   \bbl@languages
3775   \@ifundefined{bbl@hyphendata@the\language}{%
3776     {\bbl@info{No hyphenation patterns were set for\%
3777       language '\bbl@tempa'. Reported}}%
3778     {\expandafter\expandafter\expandafter\bbl@luapatterns
3779       \csname bbl@hyphendata@the\language\endcsname}}}}
3780 \endinput\fi
3781 \begingroup
3782 \catcode`\%=12
3783 \catcode`\ '=12

```

```

3784 \catcode`\:=12
3785 \catcode`\:=12
3786 \directlua{
3787   Babel = Babel or {}
3788   function Babel.bytes(line)
3789     return line:gsub(".",
3790       function (chr) return unicode.utf8.char(string.byte(chr)) end)
3791   end
3792   function Babel.begin_process_input()
3793     if luatexbase and luatexbase.add_to_callback then
3794       luatexbase.add_to_callback('process_input_buffer',
3795         Babel.bytes, 'Babel.bytes')
3796     else
3797       Babel.callback = callback.find('process_input_buffer')
3798       callback.register('process_input_buffer', Babel.bytes)
3799     end
3800   end
3801   function Babel.end_process_input ()
3802     if luatexbase and luatexbase.remove_from_callback then
3803       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3804     else
3805       callback.register('process_input_buffer', Babel.callback)
3806     end
3807   end
3808   function Babel.addpatterns(pp, lg)
3809     local lg = lang.new(lg)
3810     local pats = lang.patterns(lg) or ''
3811     lang.clear_patterns(lg)
3812     for p in pp:gmatch('[^%s]+') do
3813       ss = ''
3814       for i in string.utfcharacters(p:gsub('%d', '')) do
3815         ss = ss .. '%d?' .. i
3816       end
3817       ss = ss:gsub('^%%d%?.', '%%.') .. '%d?'
3818       ss = ss:gsub('%.%%d%?$', '%%.')
3819       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3820       if n == 0 then
3821         tex.sprint(
3822           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
3823           .. p .. [[{}]])
3824         pats = pats .. ' ' .. p
3825       else
3826         tex.sprint(
3827           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
3828           .. p .. [[{}]])
3829       end
3830     end
3831     lang.patterns(lg, pats)
3832   end
3833 }
3834 \endgroup
3835 \ifx\newattribute\@undefined\else
3836   \newattribute\bbl@attr@locale
3837   \AddBabelHook{luatex}{beforeextras}{%
3838     \setattribute\bbl@attr@locale\localeid}
3839 \fi
3840 \def\BabelStringsDefault{unicode}
3841 \let\luabbl@stop\relax
3842 \AddBabelHook{luatex}{encodedcommands}{%

```

```

3843 \def\bb1@tempa{utf8}\def\bb1@tempb{#1}%
3844 \ifx\bb1@tempa\bb1@tempb\else
3845 \directlua{Babel.begin_process_input()}%
3846 \def\luabbl@stop{%
3847 \directlua{Babel.end_process_input()}}%
3848 \fi}%
3849 \AddBabelHook{luatex}{stopcommands}{%
3850 \luabbl@stop
3851 \let\luabbl@stop\relax}
3852 \AddBabelHook{luatex}{patterns}{%
3853 \@ifundefined{bbl@hyphendata@the\language}%
3854 {\def\bb1@elt##1##2##3##4{%
3855 \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3856 \def\bb1@tempb{##3}%
3857 \ifx\bb1@tempb@empty\else % if not a synonymous
3858 \def\bb1@tempc{##3}{##4}%
3859 \fi
3860 \bbl@csarg\xdef{hyphendata@##2}{\bb1@tempc}%
3861 \fi}%
3862 \bbl@languages
3863 \@ifundefined{bbl@hyphendata@the\language}%
3864 {\bbl@info{No hyphenation patterns were set for\%
3865 language '#2'. Reported}}%
3866 {\expandafter\expandafter\expandafter\bbl@luapatterns
3867 \csname bbl@hyphendata@the\language\endcsname}}}%
3868 \@ifundefined{bbl@patterns@}{}%
3869 \begingroup
3870 \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
3871 \ifin@else
3872 \ifx\bbl@patterns@empty\else
3873 \directlua{ Babel.addpatterns(
3874 [[\bbl@patterns@]], \number\language) }%
3875 \fi
3876 \@ifundefined{bbl@patterns@#1}%
3877 \@empty
3878 {\directlua{ Babel.addpatterns(
3879 [[\space\csname bbl@patterns@#1\endcsname]],
3880 \number\language) }}%
3881 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3882 \fi
3883 \endgroup}}
3884 \AddBabelHook{luatex}{everylanguage}{%
3885 \def\process@language##1##2##3{%
3886 \def\process@line####1####2 ####3 ####4 {}}
3887 \AddBabelHook{luatex}{loadpatterns}{%
3888 \input #1\relax
3889 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3890 {#1}}
3891 \AddBabelHook{luatex}{loadexceptions}{%
3892 \input #1\relax
3893 \def\bb1@tempb##1##2{##1}{##1}%
3894 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3895 {\expandafter\expandafter\expandafter\bbl@tempb
3896 \csname bbl@hyphendata@the\language\endcsname}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

3897 \@onlypreamble\babelpatterns
3898 \AtEndOfPackage{%
3899   \newcommand\babelpatterns[2][\@empty]{%
3900     \ifx\bbl@patterns@relax
3901       \let\bbl@patterns@\@empty
3902     \fi
3903     \ifx\bbl@pttnlist\@empty\else
3904       \bbl@warning{%
3905         You must not intermingle \string\selectlanguage\space and\%
3906         \string\babelpatterns\space or some patterns will not\%
3907         be taken into account. Reported}%
3908     \fi
3909     \ifx\@empty#1%
3910       \protected@edef\bbl@patterns@\bbl@patterns@\space#2}%
3911     \else
3912       \edef\bbl@tempb{\zap@space#1 \@empty}%
3913       \bbl@for\bbl@tempa\bbl@tempb{%
3914         \bbl@fixname\bbl@tempa
3915         \bbl@iflanguage\bbl@tempa{%
3916           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3917             \@ifundefined{bbl@patterns@\bbl@tempa}%
3918             \@empty
3919             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3920             #2}}}%
3921     \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).

```

3922 \def\bbl@intraspace#1 #2 #3\@@{%
3923   \directlua{
3924     Babel = Babel or {}
3925     Babel.intraspaces = Babel.intraspaces or {}
3926     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
3927       {b = #1, p = #2, m = #3}
3928     Babel.locale_props[\the\localeid].intraspace = %
3929       {b = #1, p = #2, m = #3}
3930   }}
3931 \def\bbl@intrapenalty#1\@@{%
3932   \directlua{
3933     Babel = Babel or {}
3934     Babel.intrapenalties = Babel.intrapenalties or {}
3935     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
3936     Babel.locale_props[\the\localeid].intrapenalty = #1
3937   }}
3938 \begingroup
3939 \catcode`\%=12
3940 \catcode`\^=14
3941 \catcode`\'=12
3942 \catcode`\~=12
3943 \gdef\bbl@seaintraspace{^
3944   \let\bbl@seaintraspace\relax
3945   \directlua{
3946     Babel = Babel or {}
3947     Babel.sea_enabled = true

```

```

3948 Babel.sea_ranges = Babel.sea_ranges or {}
3949 function Babel.set_chranges (script, chrng)
3950     local c = 0
3951     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
3952         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
3953         c = c + 1
3954     end
3955 end
3956 function Babel.sea_disc_to_space (head)
3957     local sea_ranges = Babel.sea_ranges
3958     local last_char = nil
3959     local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
3960     for item in node.traverse(head) do
3961         local i = item.id
3962         if i == node.id'glyph' then
3963             last_char = item
3964         elseif i == 7 and item.subtype == 3 and last_char
3965             and last_char.char > 0x0C99 then
3966             quad = font.getfont(last_char.font).size
3967             for lg, rg in pairs(sea_ranges) do
3968                 if last_char.char > rg[1] and last_char.char < rg[2] then
3969                     lg = lg:sub(1, 4)
3970                     local intraspace = Babel.intraspaces[lg]
3971                     local intrapenalty = Babel.intrapenalties[lg]
3972                     local n
3973                     if intrapenalty ~= 0 then
3974                         n = node.new(14, 0)      ^^ penalty
3975                         n.penalty = intrapenalty
3976                         node.insert_before(head, item, n)
3977                     end
3978                     n = node.new(12, 13)      ^^ (glue, spaceskip)
3979                     node.setglue(n, intraspace.b * quad,
3980                                 intraspace.p * quad,
3981                                 intraspace.m * quad)
3982                     node.insert_before(head, item, n)
3983                     node.remove(head, item)
3984                 end
3985             end
3986         end
3987     end
3988 end
3989 }^^
3990 \bbl@luahyphenate}
3991 \catcode`\%=14
3992 \gdef\bbl@cjkintraspaces{%
3993 \let\bbl@cjkintraspaces\relax
3994 \directlua{
3995     Babel = Babel or {}
3996     require'babel-data-cjk.lua'
3997     Babel.cjk_enabled = true
3998     function Babel.cjk_linebreak(head)
3999         local GLYPH = node.id'glyph'
4000         local last_char = nil
4001         local quad = 655360      % 10 pt = 655360 = 10 * 65536
4002         local last_class = nil
4003
4004         for item in node.traverse(head) do
4005             if item.id == GLYPH then
4006

```

```

4007     local LOCALE = node.get_attribute(item,
4008         luatexbase.registernumber'bbl@attr@locale')
4009
4010     local props = Babel.locale_props[LOCALE]
4011
4012     class = Babel.cjk_class[item.char].c
4013
4014     if class == 'cp' then class = 'cl' end % )] as CL
4015     if class == 'id' then class = 'I' end
4016
4017     if class and last_class and Babel.cjk_breaks[last_class][class] then
4018         br = Babel.cjk_breaks[last_class][class]
4019     else
4020         br = 0
4021     end
4022
4023     if br == 1 and props.linebreak == 'c' then
4024         local intrapenalty = props.intrapenalty
4025         if intrapenalty ~= 0 then
4026             local n = node.new(14, 0)    % penalty
4027             n.penalty = intrapenalty
4028             node.insert_before(head, item, n)
4029         end
4030         local intraspace = props.intraspace
4031         local n = node.new(12, 13)    % (glue, spaceskip)
4032         node.setglue(n, intraspace.b * quad,
4033             intraspace.p * quad,
4034             intraspace.m * quad)
4035         node.insert_before(head, item, n)
4036     end
4037
4038     quad = font.getfont(item.font).size
4039     last_class = class
4040     else % if penalty, glue or anything else
4041         last_class = nil
4042     end
4043     end
4044     lang.hyphenate(head)
4045 end
4046 }%
4047 \bbl@luahyphenate}
4048 \gdef\bbl@luahyphenate{%
4049 \let\bbl@luahyphenate\relax
4050 \directlua{
4051     luatexbase.add_to_callback('hyphenate',
4052     function (head, tail)
4053         if Babel.cjk_enabled then
4054             Babel.cjk_linebreak(head)
4055         end
4056         lang.hyphenate(head)
4057         if Babel.sea_enabled then
4058             Babel.sea_disc_to_space(head)
4059         end
4060     end,
4061     'Babel.hyphenate')
4062 }
4063 }
4064 \endgroup

```


14.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended by 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 (but does not strictly follows) on 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.

```
4065 \AddBabelHook{luatex}{loadkernel}{%
4066 <<Restore Unicode catcodes before loading patterns>>}
4067 \ifx\DisableBabelHook\@undefined\endinput\fi
4068 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4069 \DisableBabelHook{babel-fontspec}
4070 <<Font selection>>
```

Temporary fix for luatex <1.10, which sometimes inserted a spurious closing dir node with a `\textdir` within `\hboxes`. This will be eventually removed.

```
4071 \def\bbl@luafixboxdir{%
4072   \setbox\z@\hbox{\textdir TLT}%
4073   \directlua{
4074     function Babel.first_dir(head)
4075       for item in node.traverse_id(node.id'dir', head) do
4076         return item
4077       end
4078       return nil
4079     end
4080     if Babel.first_dir(tex.box[0].head) then
4081       function Babel.fixboxdirs(head)
4082         local fd = Babel.first_dir(head)
4083         if fd and fd.dir:sub(1,1) == '-' then
4084           head = node.remove(head, fd)
4085         end
4086         return head
4087       end
4088     end
4089   }}
4090 \AtBeginDocument{\bbl@luafixboxdir}
```

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.) and with `bidi=basic-r`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

```
4091 \bbl@trace{Redefinitions for bidi layout}
4092 \ifx\@eqnnum\@undefined\else
4093   \ifx\bbl@attr@dir\@undefined\else
4094     \edef\@eqnnum{%
```

```

4095     \unexpanded{\ifcase\bbbl@attr@dir\else\bbbl@textdir\@ne\fi}%
4096     \unexpanded\expandafter{\@eqnnum}}
4097 \fi
4098 \fi
4099 \ifx\bbbl@opt@layout\@nnil\endinput\fi % if no layout
4100 \ifx\bbbl@beforeforeign\leavevmode % A poor test for bidi=
4101 \def\bbbl@nextfake#1{% non-local changes, use always inside a group!
4102     \bbbl@exp{%
4103         \mathdir\the\bodydir
4104         #1%           Once entered in math, set boxes to restore values
4105         \<ifmmode>%
4106         \everyvbox{%
4107             \the\everyvbox
4108             \bodydir\the\bodydir
4109             \mathdir\the\mathdir
4110             \everyhbox{\the\everyhbox}%
4111             \everyvbox{\the\everyvbox}}%
4112         \everyhbox{%
4113             \the\everyhbox
4114             \bodydir\the\bodydir
4115             \mathdir\the\mathdir
4116             \everyhbox{\the\everyhbox}%
4117             \everyvbox{\the\everyvbox}}%
4118         \<fi>}}%
4119 \def\@hangfrom#1{%
4120     \setbox\@tempboxa\hbox{{#1}}%
4121     \hangindent\wd\@tempboxa
4122     \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
4123         \shapemode\@ne
4124     \fi
4125     \noindent\box\@tempboxa}
4126 \fi
4127 \IfBabelLayout{tabular}
4128 {\bbbl@replace\@tabular{$}\bbbl@nextfake$}%
4129 \let\bbbl@tabular\@tabular
4130 \AtBeginDocument{%
4131     \ifx\bbbl@tabular\@tabular\else
4132         \bbbl@replace\@tabular{$}\bbbl@nextfake$}%
4133     \fi}}
4134 {}
4135 \IfBabelLayout{lists}
4136 {\expandafter\def\expandafter\bbbl@toreplace\expandafter{\list{##1}{##2}}%
4137 \bbbl@replace\bbbl@toreplace{\parshape}{\bbbl@listparshape}%
4138 \bbbl@exp{\def\\list##1##2{\the\toks@}}% \toks@ <- implicit result
4139 \def\bbbl@listparshape#1#2#3{%
4140     \parshape #1 #2 #3 %
4141     \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
4142         \shapemode\tw@
4143     \fi}}
4144 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic-r, but there are some additional readjustments for bidi=default.

```

4145 \IfBabelLayout{counters}%
4146 {\def\@textsuperscript#1{% lua has separate settings for math
4147     \m@th
4148     \mathdir\pagedir % required with basic-r; ok with default, too
4149     \ensuremath{\mbox {\fontsize \sf@size \z@ #1}}}%

```

```

4150 \let\bbllatinarabic=@arabic
4151 \def\@arabic#1{\babelsublr{\bbllatinarabic#1}}%
4152 \@ifpackagewith{babel}{bidi=default}%
4153 {\let\bbllasciroman=@roman
4154 \def\@roman#1{\babelsublr{\ensureascii{\bbllasciroman#1}}}%
4155 \let\bbllasciiRoman=@Roman
4156 \def\@Roman#1{\babelsublr{\ensureascii{\bbllasciiRoman#1}}}%
4157 \def\labelenumii{\theenumii}%
4158 \def\p@enumiii{\p@enumii}\theenumii{}}{}{}
4159 <<Footnote changes>>
4160 \IfBabelLayout{footnotes}%
4161 {\BabelFootnote\footnote\languagename{}}{}%
4162 \BabelFootnote\localfootnote\languagename{}}{}%
4163 \BabelFootnote\mainfootnote{}}{}{}
4164 {}

```

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.

```

4165 \IfBabelLayout{extras}%
4166 {\def\underline#1{%
4167 \relax
4168 \ifmmode@@underline{#1}%
4169 \else\bbllnextfake$\@@underline{\hbox{#1}}\m@th$\relax\fi}%
4170 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
4171 \if b\expandafter\@car\@series\@nil\boldmath\fi
4172 \babelsublr{%
4173 \LaTeX\kern.15em2\bbllnextfake$_{\textstyle\varepsilon}$}}}
4174 {}
4175 </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):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (`<l>`, `<r>` or `<al>`).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```

4176 (*basic-r)
4177 Babel = Babel or {}
4178
4179 Babel.bidi_enabled = true
4180
4181 require('babel-data-bidi.lua')
4182
4183 local characters = Babel.characters
4184 local ranges = Babel.ranges
4185
4186 local DIR = node.id("dir")
4187
4188 local function dir_mark(head, from, to, outer)
4189   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
4190   local d = node.new(DIR)
4191   d.dir = '+' .. dir
4192   node.insert_before(head, from, d)
4193   d = node.new(DIR)
4194   d.dir = '-' .. dir
4195   node.insert_after(head, to, d)
4196 end
4197
4198 function Babel.bidi(head, ispar)
4199   local first_n, last_n      -- first and last char with nums
4200   local last_es             -- an auxiliary 'last' used with nums
4201   local first_d, last_d     -- first and last char in L/R block
4202   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):

```

4203   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
4204   local strong_lr = (strong == 'l') and 'l' or 'r'
4205   local outer = strong
4206
4207   local new_dir = false
4208   local first_dir = false
4209   local inmath = false
4210
4211   local last_lr
4212
4213   local type_n = ''
4214
4215   for item in node.traverse(head) do
4216
4217     -- three cases: glyph, dir, otherwise
4218     if item.id == node.id'glyph'
4219       or (item.id == 7 and item.subtype == 2) then
4220
4221       local itemchar
4222       if item.id == 7 and item.subtype == 2 then
4223         itemchar = item.replace.char
4224       else

```

```

4225     itemchar = item.char
4226     end
4227     local chardata = characters[itemchar]
4228     dir = chardata and chardata.d or nil
4229     if not dir then
4230         for nn, et in ipairs(ranges) do
4231             if itemchar < et[1] then
4232                 break
4233             elseif itemchar <= et[2] then
4234                 dir = et[3]
4235                 break
4236             end
4237         end
4238     end
4239     dir = dir or 'l'
4240     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).

```

4241     if new_dir then
4242         attr_dir = 0
4243         for at in node.traverse(item.attr) do
4244             if at.number == luatexbase.registernumber'bbl@attr@dir' then
4245                 attr_dir = at.value % 3
4246             end
4247         end
4248         if attr_dir == 1 then
4249             strong = 'r'
4250         elseif attr_dir == 2 then
4251             strong = 'al'
4252         else
4253             strong = 'l'
4254         end
4255         strong_lr = (strong == 'l') and 'l' or 'r'
4256         outer = strong_lr
4257         new_dir = false
4258     end
4259
4260     if dir == 'nsm' then dir = strong end           -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

4261     dir_real = dir           -- We need dir_real to set strong below
4262     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:

```

4263     if strong == 'al' then
4264         if dir == 'en' then dir = 'an' end           -- W2
4265         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
4266         strong_lr = 'r'                               -- W3
4267     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

4268     elseif item.id == node.id'dir' and not inmath then
4269         new_dir = true

```

```

4270     dir = nil
4271   elseif item.id == node.id'math' then
4272     inmath = (item.subtype == 0)
4273   else
4274     dir = nil           -- Not a char
4275   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>.

```

4276   if dir == 'en' or dir == 'an' or dir == 'et' then
4277     if dir ~= 'et' then
4278       type_n = dir
4279     end
4280     first_n = first_n or item
4281     last_n = last_es or item
4282     last_es = nil
4283   elseif dir == 'es' and last_n then -- W3+W6
4284     last_es = item
4285   elseif dir == 'cs' then           -- it's right - do nothing
4286   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
4287     if strong_lr == 'r' and type_n ~= '' then
4288       dir_mark(head, first_n, last_n, 'r')
4289     elseif strong_lr == 'l' and first_d and type_n == 'an' then
4290       dir_mark(head, first_n, last_n, 'r')
4291       dir_mark(head, first_d, last_d, outer)
4292       first_d, last_d = nil, nil
4293     elseif strong_lr == 'l' and type_n ~= '' then
4294       last_d = last_n
4295     end
4296     type_n = ''
4297     first_n, last_n = nil, nil
4298   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:

```

4299   if dir == 'l' or dir == 'r' then
4300     if dir ~= outer then
4301       first_d = first_d or item
4302       last_d = item
4303     elseif first_d and dir ~= strong_lr then
4304       dir_mark(head, first_d, last_d, outer)
4305       first_d, last_d = nil, nil
4306     end
4307   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.

```

4308   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
4309     item.char = characters[item.char] and

```

```

4310             characters[item.char].m or item.char
4311 elseif (dir or new_dir) and last_lr ~= item then
4312     local mir = outer .. strong_lr .. (dir or outer)
4313     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
4314         for ch in node.traverse(node.next(last_lr)) do
4315             if ch == item then break end
4316             if ch.id == node.id'glyph' then
4317                 ch.char = characters[ch.char].m or ch.char
4318             end
4319         end
4320     end
4321 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).

```

4322 if dir == 'l' or dir == 'r' then
4323     last_lr = item
4324     strong = dir_real           -- Don't search back - best save now
4325     strong_lr = (strong == 'l') and 'l' or 'r'
4326 elseif new_dir then
4327     last_lr = nil
4328 end
4329 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

4330 if last_lr and outer == 'r' then
4331     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
4332         ch.char = characters[ch.char].m or ch.char
4333     end
4334 end
4335 if first_n then
4336     dir_mark(head, first_n, last_n, outer)
4337 end
4338 if first_d then
4339     dir_mark(head, first_d, last_d, outer)
4340 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

4341 return node.prev(head) or head
4342 end
4343 </basic-r>

```

And here the Lua code for bidi=basic:

```

4344 (*basic)
4345 Babel = Babel or {}
4346
4347 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
4348
4349 Babel.fontmap = Babel.fontmap or {}
4350 Babel.fontmap[0] = {}           -- l
4351 Babel.fontmap[1] = {}           -- r
4352 Babel.fontmap[2] = {}           -- al/an
4353
4354 Babel.bidi_enabled = true
4355
4356 require('babel-data-bidi.lua')
4357
4358 local characters = Babel.characters

```

```

4359 local ranges = Babel.ranges
4360
4361 local DIR = node.id('dir')
4362 local GLYPH = node.id('glyph')
4363
4364 local function insert_implicit(head, state, outer)
4365   local new_state = state
4366   if state.sim and state.eim and state.sim ~= state.eim then
4367     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
4368     local d = node.new(DIR)
4369     d.dir = '+' .. dir
4370     node.insert_before(head, state.sim, d)
4371     local d = node.new(DIR)
4372     d.dir = '-' .. dir
4373     node.insert_after(head, state.eim, d)
4374   end
4375   new_state.sim, new_state.eim = nil, nil
4376   return head, new_state
4377 end
4378
4379 local function insert_numeric(head, state)
4380   local new
4381   local new_state = state
4382   if state.san and state.ean and state.san ~= state.ean then
4383     local d = node.new(DIR)
4384     d.dir = '+TLT'
4385     _, new = node.insert_before(head, state.san, d)
4386     if state.san == state.sim then state.sim = new end
4387     local d = node.new(DIR)
4388     d.dir = '-TLT'
4389     _, new = node.insert_after(head, state.ean, d)
4390     if state.ean == state.eim then state.eim = new end
4391   end
4392   new_state.san, new_state.ean = nil, nil
4393   return head, new_state
4394 end
4395
4396 -- TODO - \hbox with an explicit dir can lead to wrong results
4397 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
4398 -- was s made to improve the situation, but the problem is the 3-dir
4399 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
4400 -- well.
4401
4402 function Babel.bidi(head, ispar, hdir)
4403   local d -- d is used mainly for computations in a loop
4404   local prev_d = ''
4405   local new_d = false
4406
4407   local nodes = {}
4408   local outer_first = nil
4409   local inmath = false
4410
4411   local glue_d = nil
4412   local glue_i = nil
4413
4414   local has_en = false
4415   local first_et = nil
4416
4417   local ATDIR = luatexbase.registernumber'bbl@attr@dir'

```



```

4418
4419 local save_outer
4420 local temp = node.get_attribute(head, ATDIR)
4421 if temp then
4422     temp = temp % 3
4423     save_outer = (temp == 0 and 'l') or
4424                 (temp == 1 and 'r') or
4425                 (temp == 2 and 'al')
4426 elseif ispar then -- Or error? Shouldn't happen
4427     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
4428 else -- Or error? Shouldn't happen
4429     save_outer = ('TRT' == hdir) and 'r' or 'l'
4430 end
4431 -- when the callback is called, we are just _after_ the box,
4432 -- and the textdir is that of the surrounding text
4433 -- if not ispar and hdir ~= tex.textdir then
4434 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
4435 -- end
4436 local outer = save_outer
4437 local last = outer
4438 -- 'al' is only taken into account in the first, current loop
4439 if save_outer == 'al' then save_outer = 'r' end
4440
4441 local fontmap = Babel.fontmap
4442
4443 for item in node.traverse(head) do
4444
4445     -- In what follows, #node is the last (previous) node, because the
4446     -- current one is not added until we start processing the neutrals.
4447
4448     -- three cases: glyph, dir, otherwise
4449     if item.id == GLYPH
4450         or (item.id == 7 and item.subtype == 2) then
4451
4452         local d_font = nil
4453         local item_r
4454         if item.id == 7 and item.subtype == 2 then
4455             item_r = item.replace -- automatic discs have just 1 glyph
4456         else
4457             item_r = item
4458         end
4459         local chardata = characters[item_r.char]
4460         d = chardata and chardata.d or nil
4461         if not d or d == 'nsm' then
4462             for nn, et in ipairs(ranges) do
4463                 if item_r.char < et[1] then
4464                     break
4465                 elseif item_r.char <= et[2] then
4466                     if not d then d = et[3]
4467                     elseif d == 'nsm' then d_font = et[3]
4468                     end
4469                     break
4470                 end
4471             end
4472         end
4473         d = d or 'l'
4474
4475         -- A short 'pause' in bidi for mapfont
4476         d_font = d_font or d

```

```

4477     d_font = (d_font == 'l' and 0) or
4478             (d_font == 'nsm' and 0) or
4479             (d_font == 'r' and 1) or
4480             (d_font == 'al' and 2) or
4481             (d_font == 'an' and 2) or nil
4482     if d_font and fontmap and fontmap[d_font][item_r.font] then
4483         item_r.font = fontmap[d_font][item_r.font]
4484     end
4485
4486     if new_d then
4487         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4488         if inmath then
4489             attr_d = 0
4490         else
4491             attr_d = node.get_attribute(item, ATDIR)
4492             attr_d = attr_d % 3
4493         end
4494         if attr_d == 1 then
4495             outer_first = 'r'
4496             last = 'r'
4497         elseif attr_d == 2 then
4498             outer_first = 'r'
4499             last = 'al'
4500         else
4501             outer_first = 'l'
4502             last = 'l'
4503         end
4504         outer = last
4505         has_en = false
4506         first_et = nil
4507         new_d = false
4508     end
4509
4510     if glue_d then
4511         if (d == 'l' and 'l' or 'r') ~= glue_d then
4512             table.insert(nodes, {glue_i, 'on', nil})
4513         end
4514         glue_d = nil
4515         glue_i = nil
4516     end
4517
4518     elseif item.id == DIR then
4519         d = nil
4520         new_d = true
4521
4522     elseif item.id == node.id'glue' and item.subtype == 13 then
4523         glue_d = d
4524         glue_i = item
4525         d = nil
4526
4527     elseif item.id == node.id'math' then
4528         inmath = (item.subtype == 0)
4529
4530     else
4531         d = nil
4532     end
4533
4534     -- AL <= EN/ET/ES      -- W2 + W3 + W6
4535     if last == 'al' and d == 'en' then

```

```

4536     d = 'an'           -- W3
4537 elseif last == 'al' and (d == 'et' or d == 'es') then
4538     d = 'on'           -- W6
4539 end
4540
4541 -- EN + CS/ES + EN     -- W4
4542 if d == 'en' and #nodes >= 2 then
4543     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4544         and nodes[#nodes-1][2] == 'en' then
4545         nodes[#nodes][2] = 'en'
4546     end
4547 end
4548
4549 -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
4550 if d == 'an' and #nodes >= 2 then
4551     if (nodes[#nodes][2] == 'cs')
4552         and nodes[#nodes-1][2] == 'an' then
4553         nodes[#nodes][2] = 'an'
4554     end
4555 end
4556
4557 -- ET/EN               -- W5 + W7->1 / W6->on
4558 if d == 'et' then
4559     first_et = first_et or (#nodes + 1)
4560 elseif d == 'en' then
4561     has_en = true
4562     first_et = first_et or (#nodes + 1)
4563 elseif first_et then   -- d may be nil here !
4564     if has_en then
4565         if last == 'l' then
4566             temp = 'l'   -- W7
4567         else
4568             temp = 'en' -- W5
4569         end
4570     else
4571         temp = 'on'     -- W6
4572     end
4573     for e = first_et, #nodes do
4574         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4575     end
4576     first_et = nil
4577     has_en = false
4578 end
4579
4580 if d then
4581     if d == 'al' then
4582         d = 'r'
4583         last = 'al'
4584     elseif d == 'l' or d == 'r' then
4585         last = d
4586     end
4587     prev_d = d
4588     table.insert(nodes, {item, d, outer_first})
4589 end
4590
4591 outer_first = nil
4592
4593 end
4594

```

```

4595 -- TODO -- repeated here in case EN/ET is the last node. Find a
4596 -- better way of doing things:
4597 if first_et then      -- dir may be nil here !
4598   if has_en then
4599     if last == 'l' then
4600       temp = 'l'      -- W7
4601     else
4602       temp = 'en'    -- W5
4603     end
4604   else
4605     temp = 'on'      -- W6
4606   end
4607   for e = first_et, #nodes do
4608     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4609   end
4610 end
4611
4612 -- dummy node, to close things
4613 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4614
4615 ----- NEUTRAL -----
4616
4617 outer = save_outer
4618 last = outer
4619
4620 local first_on = nil
4621
4622 for q = 1, #nodes do
4623   local item
4624
4625   local outer_first = nodes[q][3]
4626   outer = outer_first or outer
4627   last = outer_first or last
4628
4629   local d = nodes[q][2]
4630   if d == 'an' or d == 'en' then d = 'r' end
4631   if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4632
4633   if d == 'on' then
4634     first_on = first_on or q
4635   elseif first_on then
4636     if last == d then
4637       temp = d
4638     else
4639       temp = outer
4640     end
4641     for r = first_on, q - 1 do
4642       nodes[r][2] = temp
4643       item = nodes[r][1] -- MIRRORING
4644       if item.id == GLYPH and temp == 'r' then
4645         item.char = characters[item.char].m or item.char
4646       end
4647     end
4648     first_on = nil
4649   end
4650
4651   if d == 'r' or d == 'l' then last = d end
4652 end
4653

```

```

4654 ----- IMPLICIT, REORDER -----
4655
4656 outer = save_outer
4657 last = outer
4658
4659 local state = {}
4660 state.has_r = false
4661
4662 for q = 1, #nodes do
4663     local item = nodes[q][1]
4664
4665     outer = nodes[q][3] or outer
4666
4667     local d = nodes[q][2]
4668
4669     if d == 'nsm' then d = last end           -- W1
4670     if d == 'en' then d = 'an' end
4671     local isdir = (d == 'r' or d == 'l')
4672
4673     if outer == 'l' and d == 'an' then
4674         state.san = state.san or item
4675         state.ean = item
4676     elseif state.san then
4677         head, state = insert_numeric(head, state)
4678     end
4679
4680     if outer == 'l' then
4681         if d == 'an' or d == 'r' then      -- im -> implicit
4682             if d == 'r' then state.has_r = true end
4683             state.sim = state.sim or item
4684             state.eim = item
4685         elseif d == 'l' and state.sim and state.has_r then
4686             head, state = insert_implicit(head, state, outer)
4687         elseif d == 'l' then
4688             state.sim, state.eim, state.has_r = nil, nil, false
4689         end
4690     else
4691         if d == 'an' or d == 'l' then
4692             if nodes[q][3] then -- nil except after an explicit dir
4693                 state.sim = item -- so we move sim 'inside' the group
4694             else
4695                 state.sim = state.sim or item
4696             end
4697             state.eim = item
4698         elseif d == 'r' and state.sim then
4699             head, state = insert_implicit(head, state, outer)
4700         elseif d == 'r' then
4701             state.sim, state.eim = nil, nil
4702         end
4703     end
4704 end
4705
4706 if isdir then
4707     last = d           -- Don't search back - best save now
4708 elseif d == 'on' and state.san then
4709     state.san = state.san or item
4710     state.ean = item
4711 end
4712

```

```

4713 end
4714
4715 return node.prev(head) or head
4716 end
4717 </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.

```

4718 (*nil)
4719 \ProvidesLanguage{nil}[<<date>> <<version>> Nil language]
4720 \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.

```

4721 \ifx\l@nohyphenation\@undefined
4722   \nopatterns{nil}
4723   \adddialect\l@nil0
4724 \else
4725   \let\l@nil\l@nohyphenation
4726 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

4727 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```

\captionnil
\datenil
4728 \let\captionnil\@empty
4729 \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.

```

4730 \ldf@finish{nil}
4731 </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:

17.2 Emulating some L^AT_EX features

The following code duplicates or emulates parts of L^AT_EX 2_ε that are needed for babel.

```
4751 (*plain)
4752 \def\@empty{}
4753 \def\loadlocalcfg#1{%
4754   \openin0#1.cfg
4755   \ifeof0
4756     \closein0
4757   \else
4758     \closein0
4759     {\immediate\write16{*****}%
4760      \immediate\write16{* Local config file #1.cfg used}%
4761      \immediate\write16{*}%
4762     }
4763   \input #1.cfg\relax
4764   \fi
4765   \@endofldf}
```

17.3 General tools

A number of L^AT_EX macro's that are needed later on.

```
4766 \long\def\@firstofone#1{#1}
4767 \long\def\@firstoftwo#1#2{#1}
4768 \long\def\@secondoftwo#1#2{#2}
4769 \def\@nnil{\@nil}
4770 \def\@gobbletwo#1#2{}
4771 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4772 \def\@star@or@long#1{%
4773   \@ifstar
4774   {\let\l@ngrel@x\relax#1}%
4775   {\let\l@ngrel@x\long#1}}
4776 \let\l@ngrel@x\relax
4777 \def\@car#1#2\@nil{#1}
4778 \def\@cdr#1#2\@nil{#2}
4779 \let\@typeset@protect\relax
4780 \let\protected@edef\edef
4781 \long\def\@gobble#1{}
4782 \edef\@backslashchar{\expandafter\@gobble\string\}
4783 \def\strip@prefix#1>{}
4784 \def\g@addto@macro#1#2{%
4785   \toks@\expandafter{#1#2}%
4786   \xdef#1{\the\toks@}}
4787 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4788 \def\@nameuse#1{\csname #1\endcsname}
4789 \def\@ifundefined#1{%
4790   \expandafter\ifx\csname#1\endcsname\relax
4791     \expandafter\@firstoftwo
4792   \else
4793     \expandafter\@secondoftwo
4794   \fi}
4795 \def\@expandtwoargs#1#2#3{%
4796   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4797 \def\zap@space#1 #2{%
4798   #1%
4799   \ifx#2\@empty\else\expandafter\zap@space\fi
4800   #2}
```


L^AT_EX 2_ε has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
4801 \ifx\@preamblecmds\undefined
4802 \def\@preamblecmds{}
4803 \fi
4804 \def\@onlypreamble#1{%
4805 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4806 \@preamblecmds\do#1}}
4807 \@onlypreamble\@onlypreamble
```

Mimick L^AT_EX's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```
4808 \def\begin{document}{%
4809 \@begin{document}hook
4810 \global\let\@begin{document}hook\undefined
4811 \def\do##1{\global\let##1\undefined}%
4812 \@preamblecmds
4813 \global\let\do\noexpand}

4814 \ifx\@begin{document}hook\undefined
4815 \def\@begin{document}hook{}
4816 \fi
4817 \@onlypreamble\@begin{document}hook
4818 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}
```

We also have to mimick L^AT_EX's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```
4819 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
4820 \@onlypreamble\AtEndOfPackage
4821 \def\@endoflfd{}
4822 \@onlypreamble\@endoflfd
4823 \let\bbl@afterlang\empty
4824 \chardef\bbl@opt@hyphenmap\z@
```

L^AT_EX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```
4825 \ifx\if@filesw\undefined
4826 \expandafter\let\csname if@filesw\expandafter\endcsname
4827 \csname iffalse\endcsname
4828 \fi
```

Mimick L^AT_EX's commands to define control sequences.

```
4829 \def\newcommand{\@star@or@long\new@command}
4830 \def\new@command#1{%
4831 \@testopt{\@newcommand#1}0}
4832 \def\@newcommand#1[#2]{%
4833 \@ifnextchar [{\@xargdef#1[#2]}%
4834 {\@argdef#1[#2]}}
4835 \long\def\@argdef#1[#2]#3{%
4836 \@yargdef#1\@ne{#2}{#3}}
4837 \long\def\@xargdef#1[#2][#3]#4{%
4838 \expandafter\def\expandafter#1\expandafter{%
4839 \expandafter\@protected@testopt\expandafter #1%
4840 \csname\string#1\expandafter\endcsname{#3}}%
4841 \expandafter\@yargdef \csname\string#1\endcsname
4842 \tw@{#2}{#4}}
4843 \long\def\@yargdef#1#2#3{%
4844 \@tempcnta#3\relax
4845 \advance \@tempcnta \@ne
```

```

4846 \let\@hash@\relax
4847 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4848 \@tempcntb #2%
4849 \@whilenum\@tempcntb <\@tempcnta
4850 \do{%
4851   \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
4852   \advance\@tempcntb \@ne}%
4853 \let\@hash@###
4854 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
4855 \def\providecommand{\@star@or@long\provide@command}
4856 \def\provide@command#1{%
4857   \begingroup
4858   \escapechar\m@ne\edef\@gtempa{\string#1}}%
4859 \endgroup
4860 \expandafter\@ifundefined\@gtempa
4861   {\def\reserved@a{\new@command#1}}%
4862   {\let\reserved@a\relax
4863   \def\reserved@a{\new@command\reserved@a}}%
4864 \reserved@a}%

4865 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4866 \def\declare@robustcommand#1{%
4867   \edef\reserved@a{\string#1}%
4868   \def\reserved@b{#1}%
4869   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4870   \edef#1{%
4871     \ifx\reserved@a\reserved@b
4872       \noexpand\x@protect
4873       \noexpand#1%
4874     \fi
4875     \noexpand\protect
4876     \expandafter\noexpand\csname
4877       \expandafter\@gobble\string#1 \endcsname
4878   }%
4879   \expandafter\new@command\csname
4880     \expandafter\@gobble\string#1 \endcsname
4881 }
4882 \def\x@protect#1{%
4883   \ifx\protect\@typeset@protect\else
4884     \@x@protect#1%
4885   \fi
4886 }
4887 \def\@x@protect#1\fi#2#3{%
4888   \fi\protect#1%
4889 }

```

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`.

```

4890 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4891 \ifx\in@\@undefined
4892   \def\in@#1#2{%
4893     \def\in@@##1#1##2##3\in@@{%
4894       \ifx\in@@##2\in@false\else\in@true\fi}%
4895     \in@@#2#1\in@\in@@}
4896 \else
4897   \let\bbl@tempa\@empty
4898 \fi

```

4899 \bbl@tempa

L^AT_EX 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 T_EX 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).

4900 \def@ifpackagewith#1#2#3#4{#3}

The L^AT_EX macro \@ifl@aded checks whether a file was loaded. This functionality is not needed for plain T_EX but we need the macro to be defined as a no-op.

4901 \def\@ifl@aded#1#2#3#4{}

For the following code we need to make sure that the commands \newcommand and \providecommand exist with some sensible definition. They are not fully equivalent to their L^AT_EX 2_ε versions; just enough to make things work in plain T_EX environments.

```
4902 \ifx\@tempcnta\@undefined
4903   \csname newcount\endcsname\@tempcnta\relax
4904 \fi
4905 \ifx\@tempcntb\@undefined
4906   \csname newcount\endcsname\@tempcntb\relax
4907 \fi
```

To prevent wasting two counters in L^AT_EX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

```
4908 \ifx\bye\@undefined
4909   \advance\count10 by -2\relax
4910 \fi
4911 \ifx\@ifnextchar\@undefined
4912   \def\@ifnextchar#1#2#3{%
4913     \let\reserved@d=#1%
4914     \def\reserved@a{#2}\def\reserved@b{#3}%
4915     \futurelet\@let@token\@ifnch}
4916   \def\@ifnch{%
4917     \ifx\@let@token\@sptoken
4918       \let\reserved@c\@xifnch
4919     \else
4920       \ifx\@let@token\reserved@d
4921         \let\reserved@c\reserved@a
4922       \else
4923         \let\reserved@c\reserved@b
4924       \fi
4925     \fi
4926     \reserved@c}
4927   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
4928   \def\{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
4929 \fi
4930 \def\@testopt#1#2{%
4931   \@ifnextchar[#{1}{#1[#2]}}
4932 \def\@protected@testopt#1{%
4933   \ifx\protect\@typeset@protect
4934     \expandafter\@testopt
4935   \else
4936     \@x@protect#1%
4937   \fi}
4938 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4939   #2\relax}\fi}
```

```

4940 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4941         \else\expandafter\@gobble\fi{#1}}

```

17.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

4942 \def\DeclareTextCommand{%
4943     \@dec@text@cmd\providecommand
4944 }
4945 \def\ProvideTextCommand{%
4946     \@dec@text@cmd\providecommand
4947 }
4948 \def\DeclareTextSymbol#1#2#3{%
4949     \@dec@text@cmd\chardef#1{#2}#3\relax
4950 }
4951 \def\@dec@text@cmd#1#2#3{%
4952     \expandafter\def\expandafter#2%
4953         \expandafter{%
4954             \csname#3-cmd\expandafter\endcsname
4955             \expandafter#2%
4956             \csname#3\string#2\endcsname
4957         }%
4958 % \let\@ifdefinable\@rc@ifdefinable
4959 \expandafter#1\csname#3\string#2\endcsname
4960 }
4961 \def\@current@cmd#1{%
4962     \ifx\protect\@typeset@protect\else
4963         \noexpand#1\expandafter\@gobble
4964     \fi
4965 }
4966 \def\@changed@cmd#1#2{%
4967     \ifx\protect\@typeset@protect
4968         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4969             \expandafter\ifx\csname ?\string#1\endcsname\relax
4970                 \expandafter\def\csname ?\string#1\endcsname{%
4971                     \@changed@x@err{#1}%
4972                 }%
4973             \fi
4974             \global\expandafter\let
4975                 \csname\cf@encoding \string#1\expandafter\endcsname
4976                 \csname ?\string#1\endcsname
4977             \fi
4978             \csname\cf@encoding\string#1%
4979                 \expandafter\endcsname
4980         \else
4981             \noexpand#1%
4982         \fi
4983 }
4984 \def\@changed@x@err#1{%
4985     \errhelp{Your command will be ignored, type <return> to proceed}%
4986     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4987 \def\DeclareTextCommandDefault#1{%
4988     \DeclareTextCommand#1?%
4989 }
4990 \def\ProvideTextCommandDefault#1{%
4991     \ProvideTextCommand#1?%
4992 }
4993 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd

```

```

4994 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4995 \def\DeclareTextAccent#1#2#3{%
4996   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
4997 }
4998 \def\DeclareTextCompositeCommand#1#2#3#4{%
4999   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
5000   \def\reserved@b{\string##1}%
5001   \def\reserved@c{%
5002     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
5003   \ifx\reserved@b\reserved@c
5004     \expandafter\expandafter\expandafter\ifx
5005       \expandafter\@car\reserved@a\relax\relax\@nil
5006       \@text@composite
5007   \else
5008     \edef\reserved@b##1{%
5009       \def\expandafter\noexpand
5010         \csname#2\string#1\endcsname###1{%
5011           \noexpand\@text@composite
5012             \expandafter\noexpand\csname#2\string#1\endcsname
5013               ###1\noexpand\@empty\noexpand\@text@composite
5014                 {##1}%
5015             }%
5016         }%
5017     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
5018   \fi
5019   \expandafter\def\csname\expandafter\string\csname
5020     #2\endcsname\string#1-\string#3\endcsname{#4}
5021 \else
5022   \errhelp{Your command will be ignored, type <return> to proceed}%
5023   \errmessage{\string\DeclareTextCompositeCommand\space used on
5024     inappropriate command \protect#1}
5025 \fi
5026 }
5027 \def\@text@composite#1#2#3\@text@composite{%
5028   \expandafter\@text@composite@x
5029     \csname\string#1-\string#2\endcsname
5030 }
5031 \def\@text@composite@x#1#2{%
5032   \ifx#1\relax
5033     #2%
5034   \else
5035     #1%
5036   \fi
5037 }
5038 %
5039 \def\@strip@args#1:#2-#3\@strip@args{#2}
5040 \def\DeclareTextComposite#1#2#3#4{%
5041   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
5042   \bgroup
5043     \lccode`\@=#4%
5044     \lowercase{%
5045   \egroup
5046     \reserved@a @%
5047   }%
5048 }
5049 %
5050 \def\UseTextSymbol#1#2{%
5051   \let\@curr@enc\cf@encoding
5052   \@use@text@encoding{#1}%

```

```

5053 #2%
5054 % \@use@text@encoding\@curr@enc
5055 }
5056 \def\UseTextAccent#1#2#3{%
5057 % \let\@curr@enc\cf@encoding
5058 % \@use@text@encoding{#1}%
5059 % #2{\@use@text@encoding\@curr@enc\selectfont#3}%
5060 % \@use@text@encoding\@curr@enc
5061 }
5062 \def\@use@text@encoding#1{%
5063 % \edef\f@encoding{#1}%
5064 % \xdef\font@name{%
5065 % \csname\curr@fontshape/\f@size\endcsname
5066 % }%
5067 % \pickup@font
5068 % \font@name
5069 % \@@enc@update
5070 }
5071 \def\DeclareTextSymbolDefault#1#2{%
5072 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
5073 }
5074 \def\DeclareTextAccentDefault#1#2{%
5075 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
5076 }
5077 \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.

```

5078 \DeclareTextAccent{"}{OT1}{127}
5079 \DeclareTextAccent{'}{OT1}{19}
5080 \DeclareTextAccent{^}{OT1}{94}
5081 \DeclareTextAccent{\`}{OT1}{18}
5082 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```

5083 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
5084 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`}
5085 \DeclareTextSymbol{\textquoteleft}{OT1}{``\`'}
5086 \DeclareTextSymbol{\textquoteright}{OT1}{``\`'}
5087 \DeclareTextSymbol{\i}{OT1}{16}
5088 \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`.

```

5089 \ifx\scriptsize\@undefined
5090 \let\scriptsize\sevenrm
5091 \fi
5092 </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 T_EXbook*, Addison-Wesley, 1986.
- [3] Leslie Lamport, *L^AT_EX, 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.