

Babel

Version 3.46
2020/07/07

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	7
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	15
1.12	The base option	17
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	32
1.18	Dates	34
1.19	Accessing language info	34
1.20	Hyphenation and line breaking	35
1.21	Selection based on BCP 47 tags	38
1.22	Selecting scripts	39
1.23	Selecting directions	39
1.24	Language attributes	44
1.25	Hooks	44
1.26	Languages supported by babel with ldf files	45
1.27	Unicode character properties in luatex	46
1.28	Tweaking some features	47
1.29	Tips, workarounds, known issues and notes	47
1.30	Current and future work	48
1.31	Tentative and experimental code	49
2	Loading languages with language.dat	49
2.1	Format	49
3	The interface between the core of babel and the language definition files	50
3.1	Guidelines for contributed languages	51
3.2	Basic macros	52
3.3	Skeleton	53
3.4	Support for active characters	54
3.5	Support for saving macro definitions	54
3.6	Support for extending macros	55
3.7	Macros common to a number of languages	55
3.8	Encoding-dependent strings	55
4	Changes	59
4.1	Changes in babel version 3.9	59
II	Source code	60

5	Identification and loading of required files	60
6	locale directory	60
7	Tools	61
7.1	Multiple languages	65
7.2	The Package File (\LaTeX , babel.sty)	65
7.3	base	67
7.4	Conditional loading of shorthands	69
7.5	Cross referencing macros	70
7.6	Marks	73
7.7	Preventing clashes with other packages	74
7.7.1	ifthen	74
7.7.2	varioref	75
7.7.3	hhline	75
7.7.4	hyperref	76
7.7.5	fancyhdr	76
7.8	Encoding and fonts	76
7.9	Basic bidi support	78
7.10	Local Language Configuration	83
8	The kernel of Babel (babel.def, common)	87
8.1	Tools	87
9	Multiple languages	88
9.1	Selecting the language	90
9.2	Errors	99
9.3	Hooks	102
9.4	Setting up language files	104
9.5	Shorthands	106
9.6	Language attributes	115
9.7	Support for saving macro definitions	117
9.8	Short tags	118
9.9	Hyphens	119
9.10	Multiencoding strings	120
9.11	Macros common to a number of languages	126
9.12	Making glyphs available	126
9.12.1	Quotation marks	126
9.12.2	Letters	128
9.12.3	Shorthands for quotation marks	129
9.12.4	Umlauts and tremas	130
9.13	Layout	131
9.14	Load engine specific macros	132
9.15	Creating and modifying languages	132
10	Adjusting the Babel bahavior	149
11	Loading hyphenation patterns	150
12	Font handling with fontspec	155

13	Hooks for XeTeX and LuaTeX	159
13.1	XeTeX	159
13.2	Layout	161
13.3	LuaTeX	163
13.4	Southeast Asian scripts	169
13.5	CJK line breaking	172
13.6	Automatic fonts and ids switching	173
13.7	Layout	183
13.8	Auto bidi with basic and basic-r	186
14	Data for CJK	196
15	The ‘nil’ language	197
16	Support for Plain T_EX (plain.def)	197
16.1	Not renaming hyphen.tex	197
16.2	Emulating some L ^A T _E X features	198
16.3	General tools	199
16.4	Encoding related macros	203
17	Acknowledgements	205

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	8
Unknown language ‘LANG’	8
Argument of \language@active@arg” has an extra }	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	28
Package babel Info: The following fonts are not babel standard families	28

Part I

User guide

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

1 The user interface

1.1 Monolingual documents

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

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

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[russian]{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

PDFTEX

```
\documentclass{article}

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

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

1.3 Mostly monolingual documents

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

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

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

EXAMPLE A trivial document is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

1.4 Modifiers

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

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}.
```

- 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
```

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

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

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

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.

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

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

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

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

`\foreignlanguage` [*<option-list>*]{*<language>*}{*<text>*}

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one.

This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the `bidi` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

New 3.44 As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

```
\foreignlanguage[date]{polish}{\today}
```

In addition, captions can be switched with `captions` (or both, of course, with `date, captions`). Until 3.43 you had to write something like `{\selectlanguage{...} ...}`, which was not always the most convenient way.

1.8 Auxiliary language selectors

`\begin{otherlanguage}` {*<language>*} ... `\end{otherlanguage}`

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

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

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

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

Spaces after the environment are ignored.

`\begin{otherlanguage*}` [*<option-list>*]{*<language>*} ... `\end{otherlanguage*}`

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

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

`\begin{hyphenrules}` {*<language>*} ... `\end{hyphenrules}`

The environment `hyphenrules` can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’,

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

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 other `language*` (the starred version) is preferred, as the former does not take into account possible changes in encodings of characters like, say, ‘ done by some languages (eg, italian, french, ukraineb). To set hyphenation exceptions, use `\babelhyphenation` (see below).

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

They are activated when the language is selected (at the `afterextras` event), and it makes some assumptions which could not be fulfilled in some languages. Note also you should include only macros defined by the language, not global macros (eg, `\TeX` 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, 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 `\kernbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

NOTE Note the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

\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.

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

\useshorthands `*{\<char>}`

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

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

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

\defineshorthand `[\<language>,\<language>,...]{\<shorthand>}{\<code>}`

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

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

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

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

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

⁵With it, encoded strings may not work as expected.

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

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

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

`\languageshorthands` `{⟨language⟩}`

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

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

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

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

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

`\babelshorthand` `{⟨shorthand⟩}`

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

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

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

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

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

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

⁷Thanks to Enrico Gregorio

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

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

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

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

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

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

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

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

1.11 Package options

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

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

KeepShorthandsActive	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
activeacute	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
activegrave	Same for `.
shorthands=	<p>$\langle char \rangle \langle char \rangle \dots$ off</p> <p>The only language shorthands activated are those given, like, eg:</p> <pre style="background-color: #f0f0f0; padding: 10px;">\usepackage[esperanto,french,shorthands=;!]{babel}</pre> <p>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.</p>
safe=	<p>none ref bib</p> <p>Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of New 3.34, in $\epsilon\TeX$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).</p>
math=	<p>active normal</p> <p>Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.</p>
config=	<p>$\langle file \rangle$</p> <p>Load $\langle file \rangle.cfg$ instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).</p>
main=	<p>$\langle language \rangle$</p> <p>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.</p>
headfoot=	<p>$\langle language \rangle$</p> <p>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.</p>

- 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 | *<label>* | **
 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 | first | 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. 1.23.
- layout=** **New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.23.

1.12 The base option

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

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 the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.

last language passed as option (by its name in `language.dat`). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

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

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

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

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

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

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

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

1.13 ini files

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

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

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

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

LUATEX/XETEX

```
\documentclass{book}

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

```

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

NOTE The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow (which could no longer be valid when you read this manual, if the packages involved have been updated). The Harfbuzz renderer has still some issues, so as a rule of thumb prefer the default renderer, and resort to Harfbuzz only if the former does not work for you. Fortunately, fonts can be loaded twice with different renderers; for example:

```

\babelfont[spanish]{rm}{FreeSerif}
\babelfont[hindi]{rm}[Renderer=Harfbuzz]{FreeSerif}

```

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

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

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

```

\newfontscript{Devanagari}{deva}

```

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

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

```

\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{ໂ ລ ອ ນ ງ ນ າ} % Random

```

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

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

Latin, Greek, Cyrillic Combining chars with the default luatex font renderer might be wrong; on the other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenation points are discarded (this bug seems related to kerning, so it depends on the font). With xetex both combining characters and hyphenation work as expected (not quite, but in most cases it works; the problem here are font clusters).

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

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

af	Afrikaans ^{ul}	cs	Czech ^{ul}
agq	Aghem	cu	Church Slavic
ak	Akan	cu-Cyrs	Church Slavic
am	Amharic ^{ul}	cu-Glag	Church Slavic
ar	Arabic ^{ul}	cy	Welsh ^{ul}
ar-DZ	Arabic ^{ul}	da	Danish ^{ul}
ar-MA	Arabic ^{ul}	dav	Taita
ar-SY	Arabic ^{ul}	de-AT	German ^{ul}
as	Assamese	de-CH	German ^{ul}
asa	Asu	de	German ^{ul}
ast	Asturian ^{ul}	dje	Zarma
az-Cyrl	Azerbaijani	dsb	Lower Sorbian ^{ul}
az-Latn	Azerbaijani	dua	Duala
az	Azerbaijani ^{ul}	dyo	Jola-Fonyi
bas	Basaa	dz	Dzongkha
be	Belarusian ^{ul}	ebu	Embu
bem	Bemba	ee	Ewe
bez	Bena	el	Greek ^{ul}
bg	Bulgarian ^{ul}	el-polyton	Polytonic Greek ^{ul}
bm	Bambara	en-AU	English ^{ul}
bn	Bangla ^{ul}	en-CA	English ^{ul}
bo	Tibetan ^u	en-GB	English ^{ul}
brx	Bodo	en-NZ	English ^{ul}
bs-Cyrl	Bosnian	en-US	English ^{ul}
bs-Latn	Bosnian ^{ul}	en	English ^{ul}
bs	Bosnian ^{ul}	eo	Esperanto ^{ul}
ca	Catalan ^{ul}	es-MX	Spanish ^{ul}
ce	Chechen	es	Spanish ^{ul}
cgg	Chiga	et	Estonian ^{ul}
chr	Cherokee	eu	Basque ^{ul}
ckb	Central Kurdish	ewo	Ewondo
cop	Coptic	fa	Persian ^{ul}

ff	Fulah	ksb	Shambala
fi	Finnish ^{ul}	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French ^{ul}	ky	Kyrgyz
fr-BE	French ^{ul}	lag	Langi
fr-CA	French ^{ul}	lb	Luxembourgish
fr-CH	French ^{ul}	lg	Ganda
fr-LU	French ^{ul}	lkt	Lakota
fur	Friulian ^{ul}	ln	Lingala
fy	Western Frisian	lo	Lao ^{ul}
ga	Irish ^{ul}	lrc	Northern Luri
gd	Scottish Gaelic ^{ul}	lt	Lithuanian ^{ul}
gl	Galician ^{ul}	lu	Luba-Katanga
grc	Ancient Greek ^{ul}	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian ^{ul}
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa ¹	mg	Malagasy
ha	Hausa	mgf	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ¹
hy	Armenian ^u	ms-SG	Malay ¹
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
ka	Georgian ^{ul}	nl	Dutch ^{ul}
kab	Kabyle	nmg	Kwasio
kam	Kamba	nn	Norwegian Nynorsk ^{ul}
kde	Makonde	nnh	Ngiemboon
kea	Kabuverdianu	nus	Nuer
khq	Koyra Chiini	nyn	Nyankole
ki	Kikuyu	om	Oromo
kk	Kazakh	or	Odia
kkj	Kako	os	Ossetic
kl	Kalaallisut	pa-Arab	Punjabi
klj	Kalenjin	pa-Guru	Punjabi
km	Khmer	pa	Punjabi
kn	Kannada ^{ul}	pl	Polish ^{ul}
ko	Korean	pms	Piedmontese ^{ul}
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese ^{ul}

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

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 `\babel provide` with a valueless `import`.

aghem	american
akan	amharic
albanian	ancientgreek

arabic	chinese-simplified-hongkongsarchina
arabic-algeria	chinese-simplified-macausarchina
arabic-DZ	chinese-simplified-singapore
arabic-morocco	chinese-simplified
arabic-MA	chinese-traditional-hongkongsarchina
arabic-syria	chinese-traditional-macausarchina
arabic-SY	chinese-traditional
armenian	chinese
assamese	churchslavic
asturian	churchslavic-cyrs
asu	churchslavic-oldcyrillic ¹³
australian	churchsslavic-glag
austrian	churchsslavic-glagolitic
azerbaijani-cyrillic	cognian
azerbaijani-cyrl	cornish
azerbaijani-latin	croatian
azerbaijani-latn	czech
azerbaijani	danish
bafia	duala
bambara	dutch
basaa	dzongkha
basque	embu
belarusian	english-au
bemba	english-australia
ben	english-ca
bengali	english-canada
bodo	english-gb
bosnian-cyrillic	english-newzealand
bosnian-cyrl	english-nz
bosnian-latin	english-unitedkingdom
bosnian-latn	english-unitedstates
bosnian	english-us
brazilian	english
breton	esperanto
british	estonian
bulgarian	ewe
burmese	ewondo
canadian	faroes
cantonese	filipino
catalan	finnish
centralatlastamazight	french-be
centralkurdish	french-belgium
chechen	french-ca
cherokee	french-canada
chiga	french-ch
chinese-hans-hk	french-lu
chinese-hans-mo	french-luxembourg
chinese-hans-sg	french-switzerland
chinese-hans	french
chinese-hant-hk	friulian
chinese-hant-mo	fulah
chinese-hant	galician

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

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
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt

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
shona	welsh
sichuanyi	westernfrisian
sinhala	yangben

yiddish
yoruba

zarma
zulu afrikaans

Modifying and adding values to ini files

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

1.14 Selecting fonts

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

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

NOTE See the note in the previous section about some issues in specific languages.

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}
```

¹⁴See also the package `combofont` for a complementary approach.

```
Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NOTE These macros (`\captions $\langle lang \rangle$` , `\extras $\langle lang \rangle$`) 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` [$\langle options \rangle$] { $\langle language-name \rangle$ }

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

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

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

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

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

EXAMPLE If you need a language named `arhinish`:

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

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

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

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

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

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

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

import= *<language-tag>*

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

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

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

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

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

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

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

`\<language>date{\the\year}{\the\month}{\the\day}`. **New 3.44** More convenient is usually `\localedate`, with prints the date for the current locale.

captions= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

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

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

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

main This valueless option makes the language the main one (thus overriding that set when `babel` is loaded). Only in newly defined languages.

EXAMPLE Let's assume your document is mainly in Polytonic Greek, but with some sections in Italian. Then, the first attempt should be:

```
\usepackage[italian, greek.polutonic]{babel}
```

But if, say, accents in Greek are not shown correctly, you could try:

```
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
```

script= \langle *script-name* \rangle

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

language= \langle *language-name* \rangle

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

alph= \langle *counter-name* \rangle

Assigns to `\alph` that counter. See the next section.

Alph= \langle *counter-name* \rangle

Same for `\Alph`.

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

onchar= ids | fonts

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

NOTE An alternative approach with `luatex` and `Harfbuzz` is the `font` option `RawFeature={multiscript=auto}`. It does not switch the `babel` language and therefore the line breaking rules, but in many cases it could be enough.

mapfont= direction

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

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

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

intrapenalty= $\langle penalty \rangle$

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

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

1.17 Digits and counters

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

For example:

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

```

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

```

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

The styles are:

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

Amharic `afar`, `agaw`, `ari`, `blin`, `dizi`, `gedeo`, `gumuz`, `hadiyya`, `harari`, `kaffa`, `kebena`, `kembata`, `konso`, `kunama`, `meen`, `oromo`, `saho`, `sidama`, `silti`, `tigre`, `wolaita`, `yemsa`

Arabic `abjad`, `maghrebi.abjad`

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

Bengali `alphabetic`

Coptic `epact`, `lower.letters`

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

Hindi `alphabetic`

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

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

Georgian `letters`

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

Khmer `consonant`

Korean consonant, syllable, hanja.informal, hanja.formal, hangul.formal,
 cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,
 fullwidth.upper.alpha
Marathi alphabetic
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Syriac letters
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,
 fullwidth.upper.alpha

New 3.45 In addition, native digits (in languages defining them) may be printed with the numeral style digits.

1.18 Dates

New 3.45 When the data is taken from an ini file, you may print the date corresponding to the Gregorian calendar and other lunisolar systems with the following command.

\localedate [*<calendar=.., variant=..>*]{*<year>*}{*<month>*}{*<day>*}

By default the calendar is the Gregorian, but a ini files may define strings for other calendars (currently ar, ar-*, he, fa, hi.) In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like 30. *Çileyâ Pêşîn 2019*, but with variant=iza fa it prints 31'ê *Çileyâ Pêşînê 2019*.

1.19 Accessing language info

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

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

\iflanguage {*<language>*}{*<true>*}{*<false>*}

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

\localeinfo {*<field>*}

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

`name.english` as provided by the Unicode CLDR.

`tag.ini` is the tag of the ini file (the way this file is identified in its name).

`tag.bcp47` is the full BCP 47 tag (see the warning below).
`language.tag.bcp47` is the BCP 47 language tag.
`tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).
`script.name`, as provided by the Unicode CLDR.
`script.tag.bcp47` is the BCP 47 tag of the script used by this locale.
`script.tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).

WARNING **New 3.46** As of version 3.46 `tag.bcp47` returns the full BCP 47 tag. Formerly it returned just the language subtag, which was clearly counterintuitive.

`\getlocaleproperty` `{⟨macro⟩}{⟨locale⟩}{⟨property⟩}`

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

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

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

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

`\babelhyphen` `*{⟨type⟩}`
`\babelhyphen` `*{⟨text⟩}`

New 3.9a It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in `TeX` are entered as `-`, and (2) *optional* or *soft hyphens*, which are entered as `\-`. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in `TeX` terms, a

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

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

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

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

There are also some differences with \TeX : (1) the character used is that set for the current font, while in \TeX it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in \TeX , 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 patterns for the language, you can add at least some typical cases.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

1.21 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{ autload.bcp47 = on }

\begin{document}

\today

\selectlanguage{fr-CA}

\today

\end{document}

```

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

The behaviour is adjusted with `\babeladjust` with the following parameters:

`autload.bcp47` with values on and off.

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

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

New 3.46 If an `ldf` file has been loaded, you can enable the corresponding language tags as selector names with:

```
\babeladjust{ bcp47.toname = on }
```

(You can deactivate it with `off`.) So, if `dutch` is one of the package (or class) options, you can write `\selectlanguage{nl}`. Note the language name does not change (in this example is still `dutch`), but you can get it with `\localeinfo` or `\getlanguageproperty`. It must be turned on explicitly for similar reasons to those explained above.

1.22 Selecting scripts

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

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

`\ensureascii` `{⟨text⟩}`

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

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

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

1.23 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%).

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

¹⁷But still defined for backwards compatibility.

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

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

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

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

There are some package options controlling **bidi** writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection{.section}`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for

numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it could depend on the counter format.

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

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 option does exactly is also explained there).

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr `{\lr-text}`

Digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, xetex). This command is provided to set `{\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:

¹⁸Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

```
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` `{<section-name>}`

Mainly for bidi text, but it could be useful in other cases. `\BabelPatchSection` and the corresponding option `layout=sectioning` takes a more logical approach (at least in many cases) because it applies the global language to the section format (including the `\chaptername` in `\chapter`), while the section text is still the current language. The latter is passed to tocs and marks, too, and with `sectioning` in `layout` they both reset the “global” language to the main one, while the text uses the “local” language. With `layout=sectioning` all the standard sectioning commands are redefined (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` `{<cmd>}{<local-language>}{<before>}{<after>}`

New 3.17 Something like:

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

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

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

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

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

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

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

```
\BabelFootnote{\enfootnote}{english}\{.}
```

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

1.24 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).

1.25 Hooks

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

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

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

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.26 Languages supported by babel with ldf files

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

Afrikaans afrikaans

Azerbaijani azerbaijani

Basque basque

Breton breton

Bulgarian bulgarian

Catalan catalan

Croatian croatian

Czech czech

Danish danish

Dutch dutch

English english, USenglish, american, UKenglish, british, canadian, australian, newzealand

Esperanto esperanto

Estonian estonian

Finnish finnish

French french, francais, canadien, acadian

Galician galician

German austrian, german, germanb, ngerman, naustrian

Greek greek, polutonikogreek

Hebrew hebrew

Icelandic icelandic

Indonesian indonesian (bahasa, indon, bahasai)

Interlingua interlingua

Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu (bahasam)
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, brazilian (portuges, brazil)¹⁹
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppsorbian
Welsh welsh

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

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

```

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

```

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

1.27 Unicode character properties in luatex

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

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

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

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

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

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

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

1.28 Tweaking some features

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

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

1.29 Tips, workarounds, known issues and notes

- If you use the document class *book* and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), \TeX 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 `lATeX` 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 (`lATeX`); (2) make it unactive (your settings); (3) make `babel` shorthands active (`babel`); (4) reload `hline` (`babel`, now with the correct catcodes for `|` and `:`).

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

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been

finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.30 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²¹ But that is the easy part, because they don't require modifying the \LaTeX internals.

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

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ből", in Spanish an item labelled "3.^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

²⁰This explains why \LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingsphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

²¹See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to \TeX because their aim is just to display information and not fine typesetting.

`\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).

1.31 Tentative and experimental code

See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). For old an deprecated functions, see the wiki.

`\babelprehyphenation`

New 3.44 Note it is tentative, but the current behavior for glyphs should be correct. It is similar to `\babelposthyphenation`, but (as its name implies) applied before hyphenation. There are other differences: (1) the first argument is the locale instead the name of hyphenation patterns; (2) in the search patterns = has no special meaning (| is still reserved, but currently unused); (3) in the replacement, discretionaries are not accepted, only remove, , and string = ...
Currently it handles glyphs, not discretionaries or spaces (in particular, it will not catch the hyphen and you can't insert or remove spaces). Also, you are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.
Performance is still somewhat poor.

2 Loading languages with `language.dat`

\TeX and most engines based on it (pdf \TeX , xetex, ϵ - \TeX , the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, \LaTeX , Xe \LaTeX , pdf \LaTeX). babel provides a tool which has become standard in many distributions and based on a “configuration file” named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

New 3.9q With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).²² Until 3.9n, this task was delegated to the package `luatex-hyphen`, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).²³

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 stored²⁴. 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
```

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

²³The 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`.

²⁴This is because different operating systems sometimes use very different file-naming conventions.

```
=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.²⁵ For example:

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

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

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

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain $\text{T}_{\text{E}}\text{X}$ users, so the files have to be coded so that they can be read by both $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ and plain $\text{T}_{\text{E}}\text{X}$. The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the `babel` system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date<lang>` but not `\captions<lang>` does not raise an error but can lead to unexpected results.

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

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

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

3.1 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.

²⁶But not removed, for backward compatibility.

- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.
- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

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

3.2 Basic macros

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

\addlanguage The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. Here “language” is used in the T_EX 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 T_EX sense of set of hyphenation patterns.

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

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

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

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

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

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

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

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

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

\main@language To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.

\ProvidesLanguage The macro `\ProvidesLanguage` should be used to identify the language definition files. Its

	syntax is similar to the syntax of the \LaTeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions{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 \LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

```

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

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

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

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

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

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

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

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

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

```

```
% More strings

\EndBabelCommands

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

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

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

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

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

`\declare@shorthand`

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

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

The \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.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \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 this²⁷.

`\babel@save`

To save the current meaning of any control sequence, the macro `\babel@save` is provided.

²⁷This mechanism was introduced by Bernd Raichle.

It takes one argument, $\langle csname \rangle$, the control sequence for which the meaning has to be saved.

`\babel@savevariable` A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the $\langle variable \rangle$.

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

`\bbl@allowhyphens` In several languages compound words are used. This means that when \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 `\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` $\langle language-list \rangle \{ \langle category \rangle \} [\langle selector \rangle]$

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

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

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

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with `strings=generic` (no block is taken into account except those). With `strings=encoded`, strings in those blocks are set as default (internally, ?). With `strings=encoded` strings are protected, but they are correctly expanded in `\MakeUppercase` and the like. If there is no key strings, string definitions are ignored, but `\SetCases` are still 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.²⁸ 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}
```

²⁸In future releases further categories may be added.

```

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

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

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

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

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

\EndBabelCommands

```

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.²⁹

\EndBabelCommands Marks the end of the series of blocks.

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

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

\SetString $\{ \langle macro-name \rangle \} \{ \langle string \rangle \}$

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

²⁹This replaces in 3.9g a short-lived $\backslash\text{UseStrings}$ which has been removed because it did not work.

with the hook `stringprocess`).

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

\SetStringLoop $\{ \langle macro-name \rangle \} \{ \langle string-list \rangle \}$

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

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

`#1` is replaced by the roman numeral.

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

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

```
\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]
\SetCase
{\uccode"10=`I\relax}
{\lccode`I="10\relax}

\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
{\uccode`i=`İ\relax
 \uccode`ı=`I\relax}
{\lccode`İ=`i\relax
 \lccode`I=`ı\relax}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap $\{ \langle to-lower-macros \rangle \}$

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with `^` (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that led to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with `babel` were not recognized when called as global options.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

babel.sty is the \LaTeX package, which set options and load language styles.

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

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

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 encondings.

[captions] section of captions in the file charset

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

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

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won't conflict with new "global" keys (which start always with a lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.

7 Tools

```
1 <<version=3.46>>
2 <<date=2020/07/07>>
```

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@cl#1{\csname bbl@#1\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19   \fi}
20 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\empty\else#3\fi}}
```

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

```
21 \def\bbl@add@list#1#2{%
22   \edef#1{%
23     \bbl@ifunset{\bbl@stripslash#1}%
24     {}%
25     {\ifx#1\empty\else#1,\fi}%
26     #2}}
```

`\bbl@afterelse` `\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`-statement³⁰. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

³⁰This code is based on code presented in TUGboat vol. 12, no2, June 1991 in "An expansion Power Lemma" by Sonja Maus.

```

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

```

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

```

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

```

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

```

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

```

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

```

48 \begingroup
49 \gdef\bbl@ifunset#1{%
50   \expandafter\ifx\csname#1\endcsname\relax
51     \expandafter\@firstoftwo
52   \else
53     \expandafter\@secondoftwo
54   \fi}
55 \bbl@ifunset{ifcsname}%
56 {}%
57 {\gdef\bbl@ifunset#1{%
58   \ifcsname#1\endcsname
59     \expandafter\ifx\csname#1\endcsname\relax
60       \bbl@afterelse\expandafter\@firstoftwo
61     \else
62       \bbl@afterfi\expandafter\@secondoftwo
63     \fi
64   \else
65     \expandafter\@firstoftwo
66   \fi}}
67 \endgroup

```

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

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

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

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

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

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

`\bbl@replace`

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

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

```
103 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
104   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax%
105   \def\bbl@tempa{#1}%
106   \def\bbl@tempb{#2}%
107   \def\bbl@tempe{#3}}
108 \def\bbl@sreplace#1#2#3{%
```



```

109 \begingroup
110 \expandafter\bb1@parsedef\meaning#1\relax
111 \def\bb1@tempc{#2}%
112 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
113 \def\bb1@tempd{#3}%
114 \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
115 \bb1@xin@\bb1@tempc{\bb1@tempe}% If not in macro, do nothing
116 \ifin@
117 \bb1@exp{\bb1@replace\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
118 \def\bb1@tempc{% Expanded an executed below as 'uplevel'
119 \\\makeatletter % "internal" macros with @ are assumed
120 \\\scantokens{%
121 \bb1@tempa\\\@namedef{\bb1@stripslash#1}\bb1@tempb{\bb1@tempe}}%
122 \catcode64=\the\catcode64\relax}% Restore @
123 \else
124 \let\bb1@tempc\@empty % Not \relax
125 \fi
126 \bb1@exp{% For the 'uplevel' assignments
127 \endgroup
128 \bb1@tempc}} % empty or expand to set #1 with changes
129 \fi

```

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

```

130 \def\bb1@ifsamestring#1#2{%
131 \begingroup
132 \protected@edef\bb1@tempb{#1}%
133 \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
134 \protected@edef\bb1@tempc{#2}%
135 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
136 \ifx\bb1@tempb\bb1@tempc
137 \aftergroup\@firstoftwo
138 \else
139 \aftergroup\@secondoftwo
140 \fi
141 \endgroup}
142 \chardef\bb1@engine=%
143 \ifx\directlua\@undefined
144 \ifx\XeTeXinputencoding\@undefined
145 \z@
146 \else
147 \tw@
148 \fi
149 \else
150 \@ne
151 \fi
152 <</Basic macros>>

```

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

```

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

```

7.1 Multiple languages

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

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

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

`\addlanguage` This macro was introduced for \TeX < 2. Preserved for compatibility.

```
165 <<*Define core switching macros>> ≡
166 <<*Define core switching macros>> ≡
167 \countdef\last@language=19 % TODO. why? remove?
168 \def\addlanguage{\csname newlanguage\endcsname}
169 <</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 \LaTeX 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

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

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

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

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

The first two options are for debugging.

```
170 <*package>
171 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
172 \ProvidesPackage{babel}[\<<date>> \<<version>>] The Babel package]
173 \@ifpackagewith{babel}{debug}
174   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
175   \let\bbl@debug\@firstofone}
176   {\providecommand\bbl@trace[1]{}%
177   \let\bbl@debug\@gobble}
178 <<Basic macros>>
179 % Temporarily repeat here the code for errors
180 \def\bbl@error#1#2{%
181   \begingroup
182     \def\{\MessageBreak}%
183     \PackageError{babel}{#1}{#2}%
```

```

184 \endgroup}
185 \def\bbl@warning#1{%
186 \begingroup
187 \def\{\MessageBreak}%
188 \PackageWarning{babel}{#1}%
189 \endgroup}
190 \def\bbl@infowarn#1{%
191 \begingroup
192 \def\{\MessageBreak}%
193 \GenericWarning
194 {(babel) \@spaces\@spaces\@spaces}%
195 {Package babel Info: #1}%
196 \endgroup}
197 \def\bbl@info#1{%
198 \begingroup
199 \def\{\MessageBreak}%
200 \PackageInfo{babel}{#1}%
201 \endgroup}
202 \def\bbl@nocaption{\protect\bbl@nocaption@i}
203 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
204 \global\@namedef{#2}{\textbf{?#1?}}%
205 \@nameuse{#2}%
206 \bbl@warning{%
207 \@backslashchar#2 not set. Please, define\\%
208 it in the preamble with something like:\\%
209 \string\renewcommand\@backslashchar#2{..}\\%
210 Reported}}
211 \def\bbl@tentative{\protect\bbl@tentative@i}
212 \def\bbl@tentative@i#1{%
213 \bbl@warning{%
214 Some functions for '#1' are tentative.\\%
215 They might not work as expected and their behavior\\%
216 could change in the future.\\%
217 Reported}}
218 \def\@nolanerr#1{%
219 \bbl@error
220 {You haven't defined the language #1\space yet.\\%
221 Perhaps you misspelled it or your installation\\%
222 is not complete}%
223 {Your command will be ignored, type <return> to proceed}}
224 \def\@nopatterns#1{%
225 \bbl@warning
226 {No hyphenation patterns were preloaded for\\%
227 the language '#1' into the format.\\%
228 Please, configure your TeX system to add them and\\%
229 rebuild the format. Now I will use the patterns\\%
230 preloaded for \bbl@nulllanguage\space instead}}
231 % End of errors
232 \@ifpackagewith{babel}{silent}
233 {\let\bbl@info\@gobble
234 \let\bbl@infowarn\@gobble
235 \let\bbl@warning\@gobble}
236 {}
237 %
238 \def\AfterBabelLanguage#1{%
239 \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

```

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

```

240 \ifx\bbl@languages\undefined\else
241   \begingroup
242     \catcode`\^^I=12
243     \@ifpackagewith{babel}{showlanguages}{%
244       \begingroup
245         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
246         \wlog{<*languages>}%
247         \bbl@languages
248         \wlog{</languages>}%
249       \endgroup}{%
250     \endgroup
251     \def\bbl@elt#1#2#3#4{%
252       \ifnum#2=\z@
253         \gdef\bbl@nulllanguage{#1}%
254         \def\bbl@elt##1##2##3##4{%
255           \fi}%
256       \bbl@languages
257     \fi%

```

7.3 base

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

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

```

258 \bbl@trace{Defining option 'base'}
259 \@ifpackagewith{babel}{base}{%
260   \let\bbl@onlyswitch\@empty
261   \let\bbl@provide@locale\relax
262   \input babel.def
263   \let\bbl@onlyswitch\@undefined
264   \ifx\directlua\undefined
265     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
266   \else
267     \input luababel.def
268     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
269   \fi
270   \DeclareOption{base}{}%
271   \DeclareOption{showlanguages}{}%
272   \ProcessOptions
273   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
274   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
275   \global\let\@ifl@ter@\@ifl@ter
276   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
277   \endinput}{}%
278 % \end{macrocode}
279 %
280 % \subsection{\texttt{key=value} options and other general option}
281 %
282 %   The following macros extract language modifiers, and only real
283 %   package options are kept in the option list. Modifiers are saved
284 %   and assigned to \BabelModifiers at \bbl@load@language; when
285 %   no modifiers have been given, the former is \relax. How
286 %   modifiers are handled are left to language styles; they can use
287 %   \in@, loop them with \@for or load |keyval|, for example.
288 %

```

```

289% \begin{macrocode}
290 \bbl@trace{key=value and another general options}
291 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
292 \def\bbl@tempb#1.#2{%
293   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
294 \def\bbl@tempd#1.#2\@nnil{%
295   \ifx\@empty#2%
296     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
297   \else
298     \in@{=}{#1}\ifin@
299     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
300   \else
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
303   \fi
304 \fi}
305 \let\bbl@tempc\@empty
306 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
307 \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.

```

308 \DeclareOption{KeepShorthandsActive}{}
309 \DeclareOption{activeacute}{}
310 \DeclareOption{activegrave}{}
311 \DeclareOption{debug}{}
312 \DeclareOption{noconfigs}{}
313 \DeclareOption{showlanguages}{}
314 \DeclareOption{silent}{}
315 \DeclareOption{mono}{}
316 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
317 % Don't use. Experimental. TODO.
318 \newif\ifbbl@single
319 \DeclareOption{selectors=off}{\bbl@singletrue}
320 <<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.

```

321 \let\bbl@opt@shorthands\@nnil
322 \let\bbl@opt@config\@nnil
323 \let\bbl@opt@main\@nnil
324 \let\bbl@opt@headfoot\@nnil
325 \let\bbl@opt@layout\@nnil

```

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

```

326 \def\bbl@tempa#1=#2\bbl@tempa{%
327   \bbl@csarg\ifx{opt@#1}\@nnil
328     \bbl@csarg\edef{opt@#1}{#2}%
329   \else
330     \bbl@error
331     {Bad option `#1=#2'. Either you have misspelled the\\%
332     key or there is a previous setting of `#1'. Valid\\%
333     keys are, among others, `shorthands', `main', `bidi',\\%

```

```

334     'strings', 'config', 'headfoot', 'safe', 'math'.}%
335     {See the manual for further details.}
336   \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.

```

337 \let\bbl@language@opts\@empty
338 \DeclareOption*{%
339   \bbl@xin@{\string=}{\CurrentOption}%
340   \ifin@
341     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
342   \else
343     \bbl@add@list\bbl@language@opts{\CurrentOption}%
344   \fi}

```

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

```

345 \ProcessOptions*

```

7.4 Conditional loading of shorthands

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

```

346 \bbl@trace{Conditional loading of shorthands}
347 \def\bbl@sh@string#1{%
348   \ifx#1\@empty\else
349     \ifx#1t\string~%
350     \else\ifx#1c\string,%
351     \else\string#1%
352   \fi\fi
353   \expandafter\bbl@sh@string
354 \fi}
355 \ifx\bbl@opt@shorthands\@nnil
356   \def\bbl@ifshorthand#1#2#3{#2}%
357 \else\ifx\bbl@opt@shorthands\@empty
358   \def\bbl@ifshorthand#1#2#3{#3}%
359 \else

```

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

```

360 \def\bbl@ifshorthand#1{%
361   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
362   \ifin@
363     \expandafter\@firstoftwo
364   \else
365     \expandafter\@secondoftwo
366   \fi}

```

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

```

367 \edef\bbl@opt@shorthands{%
368   \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.

```
369 \bbl@ifshorthand{'}%
370   {\PassOptionsToPackage{activeacute}{babel}}{}
371 \bbl@ifshorthand{`}%
372   {\PassOptionsToPackage{activegrave}{babel}}{}
373 \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.

```
374 \ifx\bbl@opt@headfoot\@nnil\else
375   \g@addto@macro\@resetactivechars{%
376     \set@typeset@protect
377     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
378     \let\protect\noexpand}
379 \fi
```

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

```
380 \ifx\bbl@opt@safe\@undefined
381   \def\bbl@opt@safe{BR}
382 \fi
383 \ifx\bbl@opt@main\@nnil\else
384   \edef\bbl@language@opts{%
385     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
386     \bbl@opt@main}
387 \fi
```

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

```
388 \bbl@trace{Defining IfBabelLayout}
389 \ifx\bbl@opt@layout\@nnil
390   \newcommand\IfBabelLayout[3]{#3}%
391 \else
392   \newcommand\IfBabelLayout[1]{%
393     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
394     \ifin@
395       \expandafter\@firstoftwo
396     \else
397       \expandafter\@secondoftwo
398     \fi}
399 \fi
```

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

```
400 \input babel.def
```

7.5 Cross referencing macros

The \LaTeX book states:

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

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

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

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

```
401 <<*More package options>> ≡
402 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
403 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
404 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
405 <</More package options>>
```

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

```
406 \bbl@trace{Cross referencing macros}
407 \ifx\bbl@opt@safe\@empty\else
408   \def\@newl@bel#1#2#3{%
409     {\@safe@activestrue
410       \bbl@ifunset{#1@#2}%
411         \relax
412         {\gdef\@multiplelabels{%
413           \@latex@warning@no@line{There were multiply-defined labels}}%
414           \@latex@warning@no@line{Label `#2' multiply defined}}%
415       \global\@namedef{#1@#2}{#3}}}
```

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

```
416 \CheckCommand*\@testdef[3]{%
417   \def\reserved@a{#3}%
418   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
419   \else
420     \@tempwattrue
421   \fi}
```

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

```
422 \def\@testdef#1#2#3{% TODO. With @samestring?
423   \@safe@activestrue
424   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
425   \def\bbl@tempb{#3}%
426   \@safe@activesfalse
427   \ifx\bbl@tempa\relax
428   \else
429     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
430   \fi
431   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
432   \ifx\bbl@tempa\bbl@tempb
433   \else
434     \@tempwattrue
435   \fi}
436 \fi
```

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


```

437 \bbl@xin@{R}\bbl@opt@safe
438 \ifin@
439 \bbl@redefineroobust\ref#1{%
440 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
441 \bbl@redefineroobust\pageref#1{%
442 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
443 \else
444 \let\org@ref\ref
445 \let\org@pageref\pageref
446 \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.

```

447 \bbl@xin@{B}\bbl@opt@safe
448 \ifin@
449 \bbl@redefine\@citex[#1]#2{%
450 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
451 \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.

```

452 \AtBeginDocument{%
453 \@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.)

```

454 \def\@citex[#1][#2]#3{%
455 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
456 \org@@citex[#1][#2]{\@tempa}}%
457 }{}

```

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

```

458 \AtBeginDocument{%
459 \@ifpackageloaded{cite}{%
460 \def\@citex[#1]#2{%
461 \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
462 }{}

```

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

```

463 \bbl@redefine\nocite#1{%
464 \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

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

```
465 \bbl@redefine\bibcite{%
466   \bbl@cite@choice
467   \bibcite}
```

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

```
468 \def\bbl@bibcite#1#2{%
469   \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.

```
470 \def\bbl@cite@choice{%
471   \global\let\bibcite\bbl@bibcite
472   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
473   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
474   \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.

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

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

```
476 \bbl@redefine\@bibitem#1{%
477   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
478 \else
479   \let\org@nocite\nocite
480   \let\org@@citex\@citex
481   \let\org@bibcite\bibcite
482   \let\org@@bibitem\@bibitem
483 \fi
```

7.6 Marks

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

```
484 \bbl@trace{Marks}
485 \IfBabelLayout{sectioning}
486   {\ifx\bbl@opt@headfoot\@nnil
487     \g@addto@macro\@resetactivechars{%
488       \set@typeset@protect
489       \expandafter\select@language@x\expandafter{\bbl@main@language}%
490       \let\protect\noexpand
491       \edef\thepage{% TODO. Only with bidi. See also above
492         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
493     \fi}
494   {\ifbbl@single\else
495     \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust
496     \markright#1{%
497       \bbl@ifblank{#1}%
```

```

498      {\org@markright{}}}%
499      {\toks@{#1}}%
500      \bbl@exp{%
501        \\org@markright{\\protect\\foreignlanguage{\\language\\name}}%
502        {\\protect\\bbl@restore@actives\\the\\toks@}}}%

```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers. The documentclasses report and book define and set the headings for the page. While doing so they also store a copy of `\markboth` in `\@mkboth`. Therefore we need to check whether `\@mkboth` has already been set. If so we need to do that again with the new definition of `\markboth`. (As of Oct 2019, L^AT_EX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

503      \ifx\@mkboth\markboth
504        \def\bbl@tempc{\let\@mkboth\markboth}
505      \else
506        \def\bbl@tempc{}
507      \fi
508      \bbl@ifunset{markboth}{\bbl@redefine\bbl@redefineroobust
509      \markboth#1#2}%
510      \protected@edef\bbl@tempb##1{%
511        \protect\foreignlanguage
512        {\language\\name}{\protect\bbl@restore@actives##1}}%
513      \bbl@ifblank{#1}%
514        {\toks@{}}%
515        {\toks@\expandafter{\bbl@tempb{#1}}}%
516      \bbl@ifblank{#2}%
517        {\@temptokena{}}%
518        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
519      \bbl@exp{\\org@markboth{\the\\toks@}{\the\\@temptokena}}
520      \bbl@tempc
521      \fi} % end ifbbl@single, end \IfBabelLayout

```

7.7 Preventing clashes with other packages

7.7.1 `ifthen`

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

```

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

```

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

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

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

```

522 \bbl@trace{Preventing clashes with other packages}
523 \bbl@xin@{R}\bbl@opt@safe
524 \ifin@
525   \AtBeginDocument{%
526     \@ifpackageloaded{ifthen}{%

```

```

527 \bbl@redefine@long\ifthenelse#1#2#3{%
528 \let\bbl@temp@pref\pageref
529 \let\pageref\org@pageref
530 \let\bbl@temp@ref\ref
531 \let\ref\org@ref
532 \@safe@activestrue
533 \org@ifthenelse{#1}%
534 {\let\pageref\bbl@temp@pref
535 \let\ref\bbl@temp@ref
536 \@safe@activesfalse
537 #2}%
538 {\let\pageref\bbl@temp@pref
539 \let\ref\bbl@temp@ref
540 \@safe@activesfalse
541 #3}%
542 }%
543 }{}%
544 }

```

7.7.2 varioref

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

```

545 \AtBeginDocument{%
546 \ifpackageloaded{varioref}{%
547 \bbl@redefine\@vpageref#1[#2]#3{%
548 \@safe@activestrue
549 \org@@@vpageref{#1}[#2]{#3}%
550 \@safe@activesfalse}%
551 \bbl@redefine\vrefpagemum#1#2{%
552 \@safe@activestrue
553 \org@vrefpagemum{#1}{#2}%
554 \@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.

```

555 \expandafter\def\csname Ref \endcsname#1{%
556 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
557 }{}%
558 }
559 \fi

```

7.7.3 hline

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

```

560 \AtEndOfPackage{%
561 \AtBeginDocument{%
562 \ifpackageloaded{hline}%
563 {\expandafter\ifx\csname normal@char\string\endcsname\relax

```

```

564     \else
565     \makeatletter
566     \def\@currname{hhline}\input{hhline.sty}\makeatother
567     \fi}%
568     {}}}

```

7.7.4 hyperref

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

```

569 \AtBeginDocument{%
570   \ifx\pdfstringdefDisableCommands\undefined\else
571     \pdfstringdefDisableCommands{\languageshorthands{system}}%
572   \fi}

```

7.7.5 fancyhdr

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

```

573 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
574   \lowercase{\foreignlanguage{#1}}}

```

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

```

575 \def\substitutefontfamily#1#2#3{%
576   \lowercase{\immediate\openout15=#1#2.fd\relax}%
577   \immediate\write15{%
578     \string\ProvidesFile{#1#2.fd}%
579     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
580     \space generated font description file]^{}
581     \string\DeclareFontFamily{#1}{#2}{}^{}
582     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^{}
583     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^{}
584     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^{}
585     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^{}
586     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^{}
587     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^{}
588     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^{}
589     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^{}
590   }%
591   \closeout15
592 }
593 \@onlypreamble\substitutefontfamily

```

7.8 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`

```

594 \bbl@trace{Encoding and fonts}
595 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
596 \newcommand\BabelNonText{TS1,T3,TS3}
597 \let\org@TeX\TeX
598 \let\org@LaTeX\LaTeX
599 \let\ensureascii\@firstofone
600 \AtBeginDocument{%
601   \in@false
602   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
603     \ifin@false
604       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
605     \fi}%
606   \ifin@ % if a text non-ascii has been loaded
607     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
608     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
609     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
610     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@@}%
611     \def\bbl@tempc#1ENC.DEF#2\@@{%
612       \ifx\empty#2\else
613         \bbl@ifunset{T#1}%
614         {}%
615         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
616       \ifin@
617         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
618         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
619       \else
620         \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
621       \fi}%
622     \fi}%
623   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
624   \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
625   \ifin@false
626     \edef\ensureascii#1{{%
627       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
628   \fi
629 \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.

```

630 \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.

```

631 \AtBeginDocument{%
632   \ifpackageloaded{fontspec}%
633     {\xdef\latinencoding{%
634       \ifx\UTFencname\@undefined

```

```

635      EU\ifcase\bbl@engine\or2\or1\fi
636      \else
637      \UTFencname
638      \fi}}%
639  {\gdef\latinencoding{OT1}%
640   \ifx\cf@encoding\bbl@t@one
641   \xdef\latinencoding{\bbl@t@one}%
642   \else
643   \ifx\@fontenc@load@list\@undefined
644   \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
645   \else
646   \def\@elt#1{,#1,}%
647   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
648   \let\@elt\relax
649   \bbl@xin@{,T1,}\bbl@tempa
650   \ifin@
651   \xdef\latinencoding{\bbl@t@one}%
652   \fi
653   \fi
654   \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.

```

655 \DeclareRobustCommand{\latintext}{%
656   \fontencoding{\latinencoding}\selectfont
657   \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.

```

658 \ifx\@undefined\DeclareTextFontCommand
659   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
660 \else
661   \DeclareTextFontCommand{\textlatin}{\latintext}
662 \fi

```

7.9 Basic bidi support

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

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

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

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.

- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but `bidi` text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `LuaTeX-j` shows, vertical typesetting is possible, too.

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

```

663 \ifodd\bbl@engine
664   \def\bbl@activate@preotf{%
665     \let\bbl@activate@preotf\relax % only once
666     \directlua{
667       Babel = Babel or {}
668       %
669       function Babel.pre_otfload_v(head)
670         if Babel.numbers and Babel.digits_mapped then
671           head = Babel.numbers(head)
672         end
673         if Babel.bidi_enabled then
674           head = Babel.bidi(head, false, dir)
675         end
676         return head
677       end
678       %
679       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
680         if Babel.numbers and Babel.digits_mapped then
681           head = Babel.numbers(head)
682         end
683         if Babel.bidi_enabled then
684           head = Babel.bidi(head, false, dir)
685         end
686         return head
687       end
688       %
689       luatexbase.add_to_callback('pre_linebreak_filter',
690         Babel.pre_otfload_v,
691         'Babel.pre_otfload_v',
692         luatexbase.priority_in_callback('pre_linebreak_filter',
693           'luaotfload.node_processor') or nil)
694       %
695       luatexbase.add_to_callback('hpack_filter',
696         Babel.pre_otfload_h,
697         'Babel.pre_otfload_h',
698         luatexbase.priority_in_callback('hpack_filter',
699           'luaotfload.node_processor') or nil)
700     }}
701 \fi

```

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

```

702 \bbl@trace{Loading basic (internal) bidi support}
703 \ifodd\bbl@engine
704   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
705     \let\bbl@beforeforeign\leavevmode
706     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
707     \RequirePackage{luatexbase}
708     \bbl@activate@preotf
709     \directlua{

```



```

710     require('babel-data-bidi.lua')
711     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
712         require('babel-bidi-basic.lua')
713     \or
714         require('babel-bidi-basic-r.lua')
715     \fi}
716 % TODO - to locale_props, not as separate attribute
717 \newattribute\bbl@attr@dir
718 % TODO. I don't like it, hackish:
719 \bbl@exp{\output{\bodydir\pagedir\the\output}}
720 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
721 \fi\fi
722 \else
723 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
724     \bbl@error
725     {The bidi method 'basic' is available only in\%
726     luatex. I'll continue with 'bidi=default', so\%
727     expect wrong results}%
728     {See the manual for further details.}%
729     \let\bbl@beforeforeign\leavevmode
730     \AtEndOfPackage{%
731         \EnableBabelHook{babel-bidi}%
732         \bbl@xebidipar}
733 \fi\fi
734 \def\bbl@loadxebidi#1{%
735     \ifx\RTLfootnotetext\undefined
736         \AtEndOfPackage{%
737             \EnableBabelHook{babel-bidi}%
738             \ifx\fontspec\undefined
739                 \usepackage{fontspec}% bidi needs fontspec
740             \fi
741             \usepackage#1{bidi}}%
742     \fi}
743 \ifnum\bbl@bidimode>200
744     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
745         \bbl@tentative{bidi=bidi}
746         \bbl@loadxebidi{}
747     \or
748         \bbl@tentative{bidi=bidi-r}
749         \bbl@loadxebidi{[rldocument]}
750     \or
751         \bbl@tentative{bidi=bidi-l}
752         \bbl@loadxebidi{}
753     \fi
754 \fi
755 \fi
756 \ifnum\bbl@bidimode=\@ne
757     \let\bbl@beforeforeign\leavevmode
758     \ifodd\bbl@engine
759         \newattribute\bbl@attr@dir
760         \bbl@exp{\output{\bodydir\pagedir\the\output}}%
761     \fi
762     \AtEndOfPackage{%
763         \EnableBabelHook{babel-bidi}%
764         \ifodd\bbl@engine\else
765             \bbl@xebidipar
766         \fi}
767 \fi

```

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

```

768 \bbl@trace{Macros to switch the text direction}
769 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
770 \def\bbl@rscripts{% TODO. Base on codes ??
771   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
772   Old Hungarian,Old Hungarian,Lybian,Mandaean,Manichaean,%
773   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
774   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
775   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
776   Old South Arabian,}%
777 \def\bbl@provide@dirs#1{%
778   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
779   \ifin@
780     \global\bbl@csarg\chardef{wdir@#1}\@ne
781     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
782     \ifin@
783       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
784       \fi
785     \else
786       \global\bbl@csarg\chardef{wdir@#1}\z@
787       \fi
788     \ifodd\bbl@engine
789       \bbl@csarg\ifcase{wdir@#1}%
790         \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
791       \or
792         \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
793       \or
794         \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
795       \fi
796     \fi}
797 \def\bbl@switchdir{%
798   \bbl@ifunset{bbl@sys@\languagename}{\bbl@provide@sys{\languagename}}{}%
799   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
800   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
801 \def\bbl@setdirs#1{% TODO - math
802   \ifcase\bbl@select@type % TODO - strictly, not the right test
803     \bbl@bodydir{#1}%
804     \bbl@pardir{#1}%
805   \fi
806   \bbl@texdir{#1}}
807 % TODO. Only if \bbl@bidimode > 0?:
808 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
809 \DisableBabelHook{babel-bidi}

```

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

```

810 \ifodd\bbl@engine % luatex=1
811   \chardef\bbl@thetextdir\z@
812   \chardef\bbl@thepardir\z@
813   \def\bbl@getluadir#1{%
814     \directlua{
815       if tex.#1dir == 'TLT' then
816         tex.sprint('0')
817       elseif tex.#1dir == 'TRT' then
818         tex.sprint('1')
819       end}}
820 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texdir.. 3=0 lr/1 rl
821   \ifcase#3relax

```

```

822     \ifcase\bbbl@getluadir{#1}\relax\else
823     #2 TLT\relax
824     \fi
825   \else
826     \ifcase\bbbl@getluadir{#1}\relax
827     #2 TRT\relax
828     \fi
829   \fi}
830 \def\bbbl@textdir#1{%
831   \bbbl@setluadir{text}\textdir{#1}%
832   \chardef\bbbl@thetextdir#1\relax
833   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*3+#1}}
834 \def\bbbl@pardir#1{%
835   \bbbl@setluadir{par}\pardir{#1}%
836   \chardef\bbbl@thepardir#1\relax}
837 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}
838 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}
839 \def\bbbl@dirparastext{\pardir\the\textdir\relax}%   %%%
840 % Sadly, we have to deal with boxes in math with basic.
841 % Activated every math with the package option bidi=:
842 \def\bbbl@mathboxdir{%
843   \ifcase\bbbl@thetextdir\relax
844     \everyhbox{\textdir TLT\relax}%
845   \else
846     \everyhbox{\textdir TRT\relax}%
847   \fi}
848 \frozen@everymath\expandafter{%
849   \expandafter\bbbl@mathboxdir\the\frozen@everymath}
850 \frozen@everydisplay\expandafter{%
851   \expandafter\bbbl@mathboxdir\the\frozen@everydisplay}
852 \else % pdftex=0, xetex=2
853   \newcount\bbbl@dirlevel
854   \chardef\bbbl@thetextdir\z@
855   \chardef\bbbl@thepardir\z@
856   \def\bbbl@textdir#1{%
857     \ifcase#1\relax
858       \chardef\bbbl@thetextdir\z@
859       \bbbl@textdir@i\beginL\endL
860     \else
861       \chardef\bbbl@thetextdir\@ne
862       \bbbl@textdir@i\beginR\endR
863     \fi}
864   \def\bbbl@textdir@i#1#2{%
865     \ifhmode
866       \ifnum\currentgrouplevel>\z@
867         \ifnum\currentgrouplevel=\bbbl@dirlevel
868           \bbbl@error{Multiple bidi settings inside a group}%
869           {I'll insert a new group, but expect wrong results.}%
870           \bgroup\aftergroup#2\aftergroup\egroup
871         \else
872           \ifcase\currentgrouptype\or % 0 bottom
873             \aftergroup#2% 1 simple {}
874           \or
875             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
876           \or
877             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
878           \or\or\or % vbox vtop align
879           \or
880             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign

```

```

881      \or\or\or\or\or\or % output math disc insert vcent mathchoice
882      \or
883      \aftergroup#2% 14 \begingroup
884      \else
885      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
886      \fi
887      \fi
888      \bbl@dirlevel\currentgrouplevel
889      \fi
890      #1%
891      \fi}
892      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
893      \let\bbl@bodydir\@gobble
894      \let\bbl@pagedir\@gobble
895      \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).

```

896      \def\bbl@xebidipar{%
897      \let\bbl@xebidipar\relax
898      \TeXeTstate\@ne
899      \def\bbl@xeverypar{%
900      \ifcase\bbl@thepardir
901      \ifcase\bbl@thetextdir\else\beginR\fi
902      \else
903      {\setbox\z@\lastbox\beginR\box\z@}%
904      \fi}%
905      \let\bbl@severypar\everypar
906      \newtoks\everypar
907      \everypar=\bbl@severypar
908      \bbl@severypar{\bbl@xeverypar\the\everypar}}
909      \ifnum\bbl@bidimode>200
910      \let\bbl@textdir\i\@gobbletwo
911      \let\bbl@xebidipar\@empty
912      \AddBabelHook{bidi}{foreign}{%
913      \def\bbl@tempa{\def\BabelText####1}%
914      \ifcase\bbl@thetextdir
915      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
916      \else
917      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
918      \fi}
919      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
920      \fi
921      \fi

```

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

```

922      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
923      \AtBeginDocument{%
924      \ifx\pdfstringdefDisableCommands\@undefined\else
925      \ifx\pdfstringdefDisableCommands\relax\else
926      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
927      \fi
928      \fi}

```

7.10 Local Language Configuration

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

with the extension .cfg. For instance the file norsk.cfg will be loaded when the language definition file norsk.ldf is loaded.

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

```

929 \bbl@trace{Local Language Configuration}
930 \ifx\loadlocalcfg\undefined
931   \@ifpackagewith{babel}{noconfigs}%
932   {\let\loadlocalcfg@gobble}%
933   {\def\loadlocalcfg#1{%
934     \InputIfFileExists{#1.cfg}%
935     {\typeout{*****^J%
936               * Local config file #1.cfg used^^J%
937               *}}%
938     \@empty}}
939 \fi

```

Just to be compatible with L^AT_EX 2.09 we add a few more lines of code. TODO. Necessary? Correct place? Used by some ldf file?

```

940 \ifx\@unexpandable@protect\undefined
941   \def\@unexpandable@protect{\noexpand\protect\noexpand}
942   \long\def\protected@write#1#2#3{%
943     \begingroup
944       \let\thepage\relax
945       #2%
946       \let\protect\@unexpandable@protect
947       \edef\reserved@a{\write#1{#3}}%
948       \reserved@a
949     \endgroup
950     \if@nobreak\ifvmode\nobreak\fi\fi}
951 \fi
952 %
953 % \subsection{Language options}
954 %
955 % Languages are loaded when processing the corresponding option
956 % \textit{except} if a |main| language has been set. In such a
957 % case, it is not loaded until all options has been processed.
958 % The following macro inputs the ldf file and does some additional
959 % checks (\input works, too, but possible errors are not caught).
960 %
961 %   \begin{macrocode}
962 \bbl@trace{Language options}
963 \let\bbl@afterlang\relax
964 \let\BabelModifiers\relax
965 \let\bbl@loaded\@empty
966 \def\bbl@load@language#1{%
967   \InputIfFileExists{#1.ldf}%
968   {\edef\bbl@loaded{\CurrentOption
969     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
970     \expandafter\let\expandafter\bbl@afterlang
971       \csname\CurrentOption.ldf-h@@k\endcsname
972     \expandafter\let\expandafter\BabelModifiers
973       \csname bbl@mod@\CurrentOption\endcsname}%
974   {\bbl@error{%
975     Unknown option '\CurrentOption'. Either you misspelled it\\%
976     or the language definition file \CurrentOption.ldf was not found}}%
977     Valid options are: shorthands=, KeepShorthandsActive,\\%
978     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
979     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

980 \def\bbl@try@load@lang#1#2#3{%
981   \IfFileExists{\CurrentOption.lda}%
982     {\bbl@load@language{\CurrentOption}}%
983     {#1\bbl@load@language{#2}#3}}
984 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
985 \DeclareOption{hebrew}{%
986   \input{rlbabel.def}%
987   \bbl@load@language{hebrew}}
988 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
989 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
990 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
991 \DeclareOption{polutonikogreek}{%
992   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
993 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
994 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
995 \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 .lda 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.

```

996 \ifx\bbl@opt@config\@nnil
997   \@ifpackagewith{babel}{noconfigs}{}%
998   {\InputIfFileExists{bblopts.cfg}%
999     {\typeout{*****^J%
1000              * Local config file bblopts.cfg used^^J%
1001              *}}%
1002     {}}%
1003 \else
1004   \InputIfFileExists{\bbl@opt@config.cfg}%
1005   {\typeout{*****^J%
1006            * Local config file \bbl@opt@config.cfg used^^J%
1007            *}}%
1008   {\bbl@error{%
1009     Local config file '\bbl@opt@config.cfg' not found}{%
1010     Perhaps you misspelled it.}}%
1011 \fi
```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages (note this list also contains the language given with `main`). If not declared above, the names of the option and the file are the same.

```

1012 \bbl@for\bbl@tempa\bbl@language@opts{%
1013   \bbl@ifunset{ds@\bbl@tempa}%
1014   {\edef\bbl@tempb{%
1015     \noexpand\DeclareOption
1016     {\bbl@tempa}%
1017     {\noexpand\bbl@load@language{\bbl@tempa}}}%
1018     \bbl@tempb}%
1019   \@empty}
```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an lda 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.

```

1020 \bbl@foreach\@classoptionslist{%
1021   \bbl@ifunset{ds@#1}%
1022     {\IfFileExists{#1.ldf}%
1023       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1024       {}}%
1025   {}}

```

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

```

1026 \ifx\bbl@opt@main\@nnil\else
1027   \expandafter
1028   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1029   \DeclareOption{\bbl@opt@main}{}
1030 \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):

```

1031 \def\AfterBabelLanguage#1{%
1032   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1033 \DeclareOption*{}
1034 \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.

```

1035 \bbl@trace{Option 'main'}
1036 \ifx\bbl@opt@main\@nnil
1037   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1038   \let\bbl@tempc\@empty
1039   \bbl@for\bbl@tempb\bbl@tempa{%
1040     \bbl@xin@{\,\bbl@tempb,}{\,\bbl@loaded,}%
1041     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1042   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1043   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1044   \ifx\bbl@tempb\bbl@tempc\else
1045     \bbl@warning{%
1046       Last declared language option is '\bbl@tempc',\%
1047       but the last processed one was '\bbl@tempb'.\%
1048       The main language cannot be set as both a global\%
1049       and a package option. Use 'main=\bbl@tempc' as\%
1050       option. Reported}%
1051   \fi
1052 \else
1053   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1054   \ExecuteOptions{\bbl@opt@main}
1055   \DeclareOption*{}
1056   \ProcessOptions*
1057 \fi
1058 \def\AfterBabelLanguage{%
1059   \bbl@error
1060     {Too late for \string\AfterBabelLanguage}%
1061     {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.

```

1062 \ifx\bbl@main@language\@undefined
1063   \bbl@info{%
1064     You haven't specified a language. I'll use 'nil'\%
1065     as the main language. Reported}
1066   \bbl@load@language{nil}
1067 \fi
1068 \</package>
1069 \<core>

```

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

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

8.1 Tools

```

1070 \ifx\ldf@quit\@undefined\else
1071 \endinput\fi % Same line!
1072 <<Make sure ProvidesFile is defined>>
1073 \ProvidesFile{babel.def}[\<date>] [\<version>] Babel common definitions]

```

The file `babel.def` expects some definitions made in the $\LaTeX 2_{\epsilon}$ style file. So, In $\LaTeX 2.09$ and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are 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).

```

1074 \ifx\AtBeginDocument\@undefined % TODO. change test.
1075   <<Emulate LaTeX>>
1076   \def\language{english}%
1077   \let\bbl@opt@shorthands\@nnil
1078   \def\bbl@ifshorthand#1#2#3{#2}%
1079   \let\bbl@language@opts\@empty
1080   \ifx\babeloptionstrings\@undefined
1081     \let\bbl@opt@strings\@nnil
1082   \else
1083     \let\bbl@opt@strings\babeloptionstrings
1084   \fi
1085   \def\BabelStringsDefault{generic}
1086   \def\bbl@tempa{normal}
1087   \ifx\babeloptionmath\bbl@tempa
1088     \def\bbl@mathnormal{\noexpand\textormath}
1089   \fi
1090   \def\AfterBabelLanguage#1#2{}
1091   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi

```



```

1092 \let\bbl@afterlang\relax
1093 \def\bbl@opt@safe{BR}
1094 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
1095 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1096 \expandafter\newif\csname ifbbl@single\endcsname
1097 \chardef\bbl@bidimode\z@
1098 \fi

```

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

```

1099 \ifx\bbl@trace\@undefined
1100 \let\LdfInit\endinput
1101 \def\ProvidesLanguage#1{\endinput}
1102 \endinput\fi % Same line!

```

And continue.

9 Multiple languages

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

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.

```

1103 <<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.

```

1104 \def\bbl@version{<<version>>}}
1105 \def\bbl@date{<<date>>}}
1106 \def\adddialect#1#2{%
1107   \global\chardef#1#2\relax
1108   \bbl@usehooks{adddialect}{#1}{#2}}%
1109 \begingroup
1110   \count@#1\relax
1111   \def\bbl@elt##1##2##3##4{%
1112     \ifnum\count@=##2\relax
1113       \bbl@info{\string#1 = using hyphenrules for ##1\\%
1114         (\string\language\the\count@)}}%
1115     \def\bbl@elt####1####2####3####4{%
1116       \fi}%
1117   \bbl@cs{languages}%
1118 \endgroup

```

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

```

1119 \def\bbl@fixname#1{%
1120   \begingroup
1121     \def\bbl@tempe{l@}%
1122     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1123     \bbl@tempd
1124     {\lowercase\expandafter{\bbl@tempd}}%
1125     {\uppercase\expandafter{\bbl@tempd}}%
1126     \@empty

```

```

1127         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1128         \uppercase\expandafter{\bbl@tempd}}}%
1129         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1130         \lowercase\expandafter{\bbl@tempd}}}%
1131     \@empty
1132     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1133 \bbl@tempd
1134 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1135 \def\bbl@iflanguage#1{%
1136   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

1137 \def\bbl@bcpcase#1#2#3#4\@#5{%
1138   \ifx\@empty#3%
1139     \uppercase{\def#5{#1#2}}%
1140   \else
1141     \uppercase{\def#5{#1}}%
1142     \lowercase{\edef#5{#5#2#3#4}}%
1143   \fi}
1144 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1145   \let\bbl@bcp\relax
1146   \lowercase{\def\bbl@tempa{#1}}%
1147   \ifx\@empty#2%
1148     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1149   \else\ifx\@empty#3%
1150     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1151     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1152     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1153   }%
1154   \ifx\bbl@bcp\relax
1155     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1156   \fi
1157   \else
1158     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1159     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1160     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1161     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1162   }%
1163   \ifx\bbl@bcp\relax
1164     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1165     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1166   }%
1167   \fi
1168   \ifx\bbl@bcp\relax
1169     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1170     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1171   }%
1172   \fi
1173   \ifx\bbl@bcp\relax
1174     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1175   \fi
1176   \fi\fi}

```

```

1177 \let\bbl@autoload@options\@empty
1178 \let\bbl@initoload\relax
1179 \def\bbl@provide@locale{%
1180   \ifx\babelprovide\undefined
1181     \bbl@error{For a language to be defined on the fly 'base'\\%
1182               is not enough, and the whole package must be\\%
1183               loaded. Either delete the 'base' option or\\%
1184               request the languages explicitly}%
1185     {See the manual for further details.}%
1186   \fi
1187 % TODO. Option to search if loaded, with \LocaleForEach
1188 \let\bbl@auxname\language % Still necessary. TODO
1189 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1190   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1191 \ifbbl@bcpallowed
1192   \expandafter\ifx\csname date\language\endcsname\relax
1193     \expandafter
1194     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1195     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1196       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1197       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1198       \expandafter\ifx\csname date\language\endcsname\relax
1199         \let\bbl@initoload\bbl@bcp
1200         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1201         \let\bbl@initoload\relax
1202       \fi
1203       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1204     \fi
1205   \fi
1206 \fi
1207 \expandafter\ifx\csname date\language\endcsname\relax
1208   \IfFileExists{babel-\language.tex}%
1209   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
1210   {}%
1211 \fi}

```

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

```

1212 \def\iflanguage#1{%
1213   \bbl@iflanguage{#1}%
1214   \ifnum\csname l@#1\endcsname=\language
1215     \expandafter\@firstoftwo
1216   \else
1217     \expandafter\@secondoftwo
1218   \fi}}

```

9.1 Selecting the language

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

```

1219 \let\bbl@select@type\z@
1220 \edef\selectlanguage{%
1221   \noexpand\protect

```

```
1222 \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`.

```
1223 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1224 \let\xstring\string
```

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

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

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

```
1225 \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`

```
1226 \def\bbl@push@language{%
1227   \ifx\language\@undefined\else
1228     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1229   \fi}
```

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

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

```
1230 \def\bbl@pop@lang#1+#2&#3{%
1231   \edef\language{#1}\xdef#3{#2}}
```

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

```
1232 \let\bbl@ifrestoring\@secondoftwo
1233 \def\bbl@pop@language{%
1234   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1235   \let\bbl@ifrestoring\@firstoftwo
1236   \expandafter\bbl@set@language\expandafter{\language}%
1237   \let\bbl@ifrestoring\@secondoftwo}
```

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

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

```

1238 \chardef\localeid\z@
1239 \def\bbl@id@last{0} % No real need for a new counter
1240 \def\bbl@id@assign{%
1241   \bbl@ifunset{bbl@id@@\language}%
1242   {\count@bbl@id@last\relax
1243    \advance\count@ne
1244    \bbl@csarg\chardef{id@@\language}\count@
1245    \edef\bbl@id@last{\the\count@}%
1246    \ifcase\bbl@engine\or
1247      \directlua{
1248        Babel = Babel or {}
1249        Babel.locale_props = Babel.locale_props or {}
1250        Babel.locale_props[\bbl@id@last] = {}
1251        Babel.locale_props[\bbl@id@last].name = '\language'
1252      }%
1253    \fi}%
1254  }%
1255  \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of `\selectlanguage`.

```

1256 \expandafter\def\csname selectlanguage \endcsname#1{%
1257   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
1258   \bbl@push@language
1259   \aftergroup\bbl@pop@language
1260   \bbl@set@language{#1}}

```

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

```

1261 \def\BabelContentsFiles{toc,lof,lot}
1262 \def\bbl@set@language#1{% from selectlanguage, pop@
1263   % The old buggy way. Preserved for compatibility.
1264   \edef\language{%
1265     \ifnum\escapechar=\expandafter`\string#1\@empty
1266     \else\string#1\@empty\fi}%
1267   \ifcat\relax\noexpand#1%
1268     \expandafter\ifx\csname date\language\endcsname\relax
1269       \edef\language{#1}%
1270       \let\localename\language
1271     \else
1272       \bbl@info{Using '\string\language' instead of 'language' is%%
1273         deprecated. If what you want is to use a%%
1274         macro containing the actual locale, make%%
1275         sure it does not not match any language.%%
1276         Reported}%

```

```

1277 %                I'll\\%
1278 %                try to fix '\string\localename', but I cannot promise\\%
1279 %                anything. Reported}%
1280 \ifx\scantokens\@undefined
1281 \def\localename{??}%
1282 \else
1283 \scantokens\expandafter{\expandafter
1284 \def\expandafter\localename\expandafter{\language}%
1285 \fi
1286 \fi
1287 \else
1288 \def\localename{#1}% This one has the correct catcodes
1289 \fi
1290 \select@language{\language}%
1291 % write to auxs
1292 \expandafter\ifx\csname date\language\endcsname\relax\else
1293 \if@files
1294 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1295 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1296 \fi
1297 \bbl@usehooks{write}{}}%
1298 \fi
1299 \fi}
1300 %
1301 \newif\ifbbl@bcpallowed
1302 \bbl@bcpallowedfalse
1303 \def\select@language#1{% from set@, babel@aux
1304 % set hymap
1305 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1306 % set name
1307 \edef\language{#1}%
1308 \bbl@fixname\language
1309 % TODO. name@map must be here?
1310 \bbl@provide@locale
1311 \bbl@iflanguage\language{%
1312 \expandafter\ifx\csname date\language\endcsname\relax
1313 \bbl@error
1314 {Unknown language '\language'. Either you have\\%
1315 misspelled its name, it has not been installed,\\%
1316 or you requested it in a previous run. Fix its name,\\%
1317 install it or just rerun the file, respectively. In\\%
1318 some cases, you may need to remove the aux file}%
1319 {You may proceed, but expect wrong results}%
1320 \else
1321 % set type
1322 \let\bbl@select@type\z@
1323 \expandafter\bbl@switch\expandafter{\language}%
1324 \fi}}
1325 \def\babel@aux#1#2{%
1326 \select@language{#1}%
1327 \bbl@foreach\BabelContentsFiles{%
1328 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1329 \def\babel@toc#1#2{%
1330 \select@language{#1}}

```

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

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

Then we have to *redefine* `\originalTeX` to compensate for the things that have been

activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

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

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

```

1331 \newif\ifbbl@usedategroup
1332 \def\bbl@switch#1{% from select@, foreign@
1333   % make sure there is info for the language if so requested
1334   \bbl@ensureinfo{#1}%
1335   % restore
1336   \originalTeX
1337   \expandafter\def\expandafter\originalTeX\expandafter{%
1338     \csname noextras#1\endcsname
1339     \let\originalTeX\empty
1340     \babel@beginsave}%
1341   \bbl@usehooks{afterreset}{}%
1342   \languageshorthands{none}%
1343   % set the locale id
1344   \bbl@id@assign
1345   % switch captions, date
1346   \ifcase\bbl@select@type
1347     \ifhmode
1348       \hskip\z@skip % trick to ignore spaces
1349       \csname captions#1\endcsname\relax
1350       \csname date#1\endcsname\relax
1351       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1352     \else
1353       \csname captions#1\endcsname\relax
1354       \csname date#1\endcsname\relax
1355     \fi
1356   \else
1357     \ifhmode
1358       \hskip\z@skip % trick to ignore spaces
1359       \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1360       \ifin@
1361         \csname captions#1\endcsname\relax
1362       \fi
1363       \bbl@xin@{,date,}{, \bbl@select@opts,}%
1364       \ifin@ % if \foreign... within \<lang>date
1365         \csname date#1\endcsname\relax
1366       \fi
1367       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1368     \else
1369       \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1370       \ifin@
1371         \csname captions#1\endcsname\relax
1372       \fi
1373       \bbl@xin@{,date,}{, \bbl@select@opts,}%
1374       \ifin@
1375         \csname date#1\endcsname\relax
1376       \fi
1377     \fi
1378   \fi

```

```

1379 % switch extras
1380 \bbl@usehooks{beforeextras}{}%
1381 \csname extras#1\endcsname\relax
1382 \bbl@usehooks{afterextras}{}%
1383 % > babel-ensure
1384 % > babel-sh-<short>
1385 % > babel-bidi
1386 % > babel-fontspec
1387 % hyphenation - case mapping
1388 \ifcase\bbl@opt@hyphenmap\or
1389 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1390 \ifnum\bbl@hymapsel>4\else
1391 \csname\language @bbl@hyphenmap\endcsname
1392 \fi
1393 \chardef\bbl@opt@hyphenmap\z@
1394 \else
1395 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1396 \csname\language @bbl@hyphenmap\endcsname
1397 \fi
1398 \fi
1399 \global\let\bbl@hymapsel@cclv
1400 % hyphenation - patterns
1401 \bbl@patterns{#1}%
1402 % hyphenation - mins
1403 \babel@savevariable\lefthyphenmin
1404 \babel@savevariable\righthyphenmin
1405 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1406 \set@hyphenmins\tw@\thr@@\relax
1407 \else
1408 \expandafter\expandafter\expandafter\set@hyphenmins
1409 \csname #1hyphenmins\endcsname\relax
1410 \fi}

```

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

```

1411 \long\def\otherlanguage#1{%
1412 \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\thr@@\fi
1413 \csname selectlanguage \endcsname{#1}%
1414 \ignorespaces}

```

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

```

1415 \long\def\endotherlanguage{%
1416 \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`.

```

1417 \expandafter\def\csname otherlanguage*\endcsname{%
1418 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1419 \def\bbl@otherlanguage@s[#1]#2{%
1420 \ifnum\bbl@hymapsel=\cclv\chardef\bbl@hymapsel4\relax\fi
1421 \def\bbl@select@opts{#1}%
1422 \foreign@language{#2}}

```


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

```
1423 \expandafter\let\csname endotherlanguage*\endcsname\relax
```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras⟨lang⟩` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

(3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in `vmode` and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into `hmode` with the surrounding `lang`, and with `\foreignlanguage*` with the new `lang`.

```
1424 \providecommand\bbl@beforeforeign{}
1425 \edef\foreignlanguage{%
1426   \noexpand\protect
1427   \expandafter\noexpand\csname foreignlanguage \endcsname}
1428 \expandafter\def\csname foreignlanguage \endcsname{%
1429   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1430 \providecommand\bbl@foreign@x[3][[]{%
1431   \begingroup
1432     \def\bbl@select@opts{#1}%
1433     \let\BabelText\@firstofone
1434     \bbl@beforeforeign
1435     \foreign@language{#2}%
1436     \bbl@usehooks{foreign}{}%
1437     \BabelText{#3}% Now in horizontal mode!
1438   \endgroup}
1439 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1440   \begingroup
1441     {\par}%
1442     \let\BabelText\@firstofone
1443     \foreign@language{#1}%
1444     \bbl@usehooks{foreign*}{}%
1445     \bbl@dirparastext
1446     \BabelText{#2}% Still in vertical mode!
1447     {\par}%
1448   \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`.

```

1449 \def\foreign@language#1{%
1450 % set name
1451 \edef\language{#1}%
1452 \ifbbl@usedategroup
1453 \bbl@add\bbl@select@opts{,date,}%
1454 \bbl@usedategroupfalse
1455 \fi
1456 \bbl@fixname\language
1457 % TODO. name@map here?
1458 \bbl@provide@locale
1459 \bbl@iflanguage\language{
1460 \expandafter\ifx\csname date\language\endcsname\relax
1461 \bbl@warning % TODO - why a warning, not an error?
1462 {Unknown language `#1'. Either you have\\%
1463 misspelled its name, it has not been installed,\\%
1464 or you requested it in a previous run. Fix its name,\\%
1465 install it or just rerun the file, respectively. In\\%
1466 some cases, you may need to remove the aux file.\\%
1467 I'll proceed, but expect wrong results.\\%
1468 Reported}%
1469 \fi
1470 % set type
1471 \let\bbl@select@type\@ne
1472 \expandafter\bbl@switch\expandafter{\language}}

```

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

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

```

1473 \let\bbl@hyphlist\empty
1474 \let\bbl@hyphenation\relax
1475 \let\bbl@pttnlist\empty
1476 \let\bbl@patterns\relax
1477 \let\bbl@hymapsel=\cclv
1478 \def\bbl@patterns#1{%
1479 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1480 \csname l@#1\endcsname
1481 \edef\bbl@tempa{#1}%
1482 \else
1483 \csname l@#1:\f@encoding\endcsname
1484 \edef\bbl@tempa{#1:\f@encoding}%
1485 \fi
1486 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1487 % > luatex
1488 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1489 \begingroup
1490 \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1491 \ifin\else
1492 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
1493 \hyphenation{%
1494 \bbl@hyphenation@
1495 \@ifundefined{bbl@hyphenation@#1}%
1496 \empty

```

```

1497      {\space\csname bbl@hyphenation@#1\endcsname}}%
1498      \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1499      \fi
1500    \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*`.

```

1501 \def\hyphenrules#1{%
1502   \edef\bbl@tempf{#1}%
1503   \bbl@fixname\bbl@tempf
1504   \bbl@iflanguage\bbl@tempf{%
1505     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1506     \languageshorthands{none}%
1507     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1508       \set@hyphenmins\tw@\thr@@\relax
1509     \else
1510       \expandafter\expandafter\expandafter\set@hyphenmins
1511       \csname\bbl@tempf hyphenmins\endcsname\relax
1512     \fi}}
1513 \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.

```

1514 \def\providehyphenmins#1#2{%
1515   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1516     \@namedef{#1hyphenmins}{#2}%
1517   \fi}

```

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

```

1518 \def\set@hyphenmins#1#2{%
1519   \lefthyphenmin#1\relax
1520   \righthyphenmin#2\relax}

```

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

```

1521 \ifx\ProvidesFile\@undefined
1522   \def\ProvidesLanguage#1[#2 #3 #4]{%
1523     \wlog{Language: #1 #4 #3 <#2>}%
1524   }
1525 \else
1526   \def\ProvidesLanguage#1{%
1527     \begingroup
1528     \catcode`\ 10 %
1529     \@makeother\%
1530     \@ifnextchar[%]
1531       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1532   \def\@provideslanguage#1[#2]{%
1533     \wlog{Language: #1 #2}%
1534     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1535   \endgroup}
1536 \fi

```

`\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`.

```
1537 \ifx\originalTeX\undefined\let\originalTeX\@empty\fi
```

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

```
1538 \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’:

```
1539 \providecommand\setlocale{%
1540   \bbl@error
1541   {Not yet available}%
1542   {Find an armchair, sit down and wait}}
1543 \let\uselocale\setlocale
1544 \let\locale\setlocale
1545 \let\selectlocale\setlocale
1546 \let\localename\setlocale
1547 \let\textlocale\setlocale
1548 \let\textlanguage\setlocale
1549 \let\languagegetext\setlocale
```

9.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about `\PackageError` it must be $\LaTeX 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```
1550 \edef\bbl@nulllanguage{\string\language=0}
1551 \ifx\PackageError\undefined % TODO. Move to Plain
1552   \def\bbl@error#1#2{%
1553     \begingroup
1554       \newlinechar=`^^J
1555       \def\{^^J(babel) }%
1556       \errhelp{#2}\errmessage{\{#1}%
1557     \endgroup}
1558   \def\bbl@warning#1{%
1559     \begingroup
1560       \newlinechar=`^^J
1561       \def\{^^J(babel) }%
1562       \message{\{#1}%
1563     \endgroup}
1564   \let\bbl@infowarn\bbl@warning
1565   \def\bbl@info#1{%
1566     \begingroup
1567       \newlinechar=`^^J
1568     \def\{^^J}%
```

```

1569     \wlog{#1}%
1570   \endgroup}
1571 \fi
1572 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1573 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1574   \global\@namedef{#2}{\textbf{?#1?}}%
1575   \@nameuse{#2}%
1576   \bbl@warning{%
1577     \@backslashchar#2 not set. Please, define\\%
1578     it in the preamble with something like:\\%
1579     \string\renewcommand\@backslashchar#2{..}\\%
1580     Reported}}
1581 \def\bbl@tentative{\protect\bbl@tentative@i}
1582 \def\bbl@tentative@i#1{%
1583   \bbl@warning{%
1584     Some functions for '#1' are tentative.\\%
1585     They might not work as expected and their behavior\\%
1586     could change in the future.\\%
1587     Reported}}
1588 \def\@nolanerr#1{%
1589   \bbl@error
1590   {You haven't defined the language #1\space yet.\\%
1591     Perhaps you misspelled it or your installation\\%
1592     is not complete}%
1593   {Your command will be ignored, type <return> to proceed}}
1594 \def\@nopatterns#1{%
1595   \bbl@warning
1596   {No hyphenation patterns were preloaded for\\%
1597     the language `#1' into the format.\\%
1598     Please, configure your TeX system to add them and\\%
1599     rebuild the format. Now I will use the patterns\\%
1600     preloaded for \bbl@nulllanguage\space instead}}
1601 \let\bbl@usehooks\@gobbletwo
1602 \ifx\bbl@onlyswitch\@empty\endinput\fi
1603 % Here ended switch.def

Here ended switch.def.

1604 \ifx\directlua\@undefined\else
1605   \ifx\bbl@luapatterns\@undefined
1606     \input luababel.def
1607   \fi
1608 \fi
1609 <<Basic macros>>
1610 \bbl@trace{Compatibility with language.def}
1611 \ifx\bbl@languages\@undefined
1612   \ifx\directlua\@undefined
1613     \openin1 = language.def % TODO. Remove hardcoded number
1614     \ifeof1
1615       \closein1
1616       \message{I couldn't find the file language.def}
1617     \else
1618       \closein1
1619     \begingroup
1620       \def\addlanguage#1#2#3#4#5{%
1621         \expandafter\ifx\csname lang@#1\endcsname\relax\else
1622           \global\expandafter\let\csname l@#1\expandafter\endcsname
1623           \csname lang@#1\endcsname
1624         \fi}%
1625       \def\uselanguage#1{%

```

```

1626      \input language.def
1627      \endgroup
1628      \fi
1629      \fi
1630      \chardef\l@english\z@
1631      \fi

```

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

```

1632 \def\addto#1#2{%
1633   \ifx#1\undefined
1634     \def#1{#2}%
1635   \else
1636     \ifx#1\relax
1637       \def#1{#2}%
1638     \else
1639       {\toks@\expandafter{#1#2}%
1640        \xdef#1{\the\toks@}}%
1641     \fi
1642   \fi}

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. TODO. Always used with additional expansions. Move them here? Move the macro to basic?

```

1643 \def\bbl@withactive#1#2{%
1644   \begingroup
1645   \lccode`~=#2\relax
1646   \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). A macro named `\macro` will be saved new control sequences named `\org@macro`.

```

1647 \def\bbl@redefine#1{%
1648   \edef\bbl@tempa{\bbl@stripslash#1}%
1649   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1650   \expandafter\def\csname\bbl@tempa\endcsname{
1651     \@onlypreamble\bbl@redefine

```

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

```

1652 \def\bbl@redefine@long#1{%
1653   \edef\bbl@tempa{\bbl@stripslash#1}%
1654   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1655   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1656     \@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_`.

```

1657 \def\bbl@redefineroast#1{%
1658   \edef\bbl@tempa{\bbl@stripslash#1}%
1659   \bbl@ifunset{\bbl@tempa\space}%
1660     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1661       \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1662     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1663     \@namedef{\bbl@tempa\space}}
1664 \@onlypreamble\bbl@redefineroast

```

9.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by babel to execute hooks defined for an event.

```

1665 \bbl@trace{Hooks}
1666 \newcommand\AddBabelHook[3][{}{%
1667   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1668   \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1669   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1670   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1671     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1672     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1673   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1674 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1675 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1676 \def\bbl@usehooks#1#2{%
1677   \def\bbl@elt##1{%
1678     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1679     \bbl@cs{ev@#1@}%
1680     \ifx\language\@undefined\else % Test required for Plain (?)
1681       \def\bbl@elt##1{%
1682         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1683         \bbl@cl{ev@#1@}%
1684       \fi}

```

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

```

1685 \def\bbl@evargs{,% <- don't delete this comma
1686   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1687   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1688   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1689   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1690   beforestart=0,language=2}

```

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

The macro `\bbl@e@<language>` contains `\bbl@ensure{\include}{\exclude}{\fontenc}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the `fontenc` is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already

contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1691 \bbl@trace{Defining babelensure}
1692 \newcommand\babelensure[2][{}% TODO - revise test files
1693   \AddBabelHook{babel-ensure}{afterextras}{%
1694     \ifcase\bbl@select@type
1695       \bbl@cl{e}%
1696     \fi}%
1697 \begingroup
1698   \let\bbl@ens@include\@empty
1699   \let\bbl@ens@exclude\@empty
1700   \def\bbl@ens@fontenc{\relax}%
1701   \def\bbl@tempb##1{%
1702     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1703   \edef\bbl@tempa{\bbl@tempb##1\@empty}%
1704   \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1705   \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1706   \def\bbl@tempc{\bbl@ensure}%
1707   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1708     \expandafter{\bbl@ens@include}}%
1709   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1710     \expandafter{\bbl@ens@exclude}}%
1711   \toks@\expandafter{\bbl@tempc}%
1712   \bbl@exp{%
1713 \endgroup
1714 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1715 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1716 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1717   \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1718     \edef##1{\noexpand\bbl@nocaption
1719       {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1720   \fi
1721   \ifx##1\@empty\else
1722     \in@{##1}{#2}%
1723     \ifin@ \else
1724       \bbl@ifunset{bbl@ensure@\language\name}%
1725       {\bbl@exp{%
1726         \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1727           \\\foreignlanguage{\language\name}%
1728           {\ifx\relax#3\else
1729             \\\fontencoding{#3}\selectfont
1730           \fi
1731           #####1}}}%
1732       }%
1733       \toks@\expandafter{##1}%
1734       \edef##1{%
1735         \bbl@csarg\noexpand{ensure@\language\name}%
1736         {\the\toks@}}%
1737     \fi
1738     \expandafter\bbl@tempb
1739   \fi}%
1740 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1741 \def\bbl@tempa##1{% elt for include list
1742   \ifx##1\@empty\else
1743     \bbl@csarg\in@{ensure@\language\name\expandafter}\expandafter{##1}%
1744     \ifin@ \else
1745       \bbl@tempb##1\@empty
1746     \fi

```



```

1747     \expandafter\bb1@tempa
1748     \fi}%
1749     \bb1@tempa#1\@empty}
1750 \def\bb1@captionslist{%
1751   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1752   \contentsname\listfigurename\listtablename\indexname\figurename
1753   \tablename\partname\enc1name\ccname\headtoname\pagename\seename
1754   \alsoname\proofname\glossaryname}

```

9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1755 \bb1@trace{Macros for setting language files up}
1756 \def\bb1@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1757   \let\bb1@screset\@empty
1758   \let\BabelStrings\bb1@opt@string
1759   \let\BabelOptions\@empty
1760   \let\BabelLanguages\relax
1761   \ifx\originalTeX\@undefined
1762     \let\originalTeX\@empty
1763   \else
1764     \originalTeX
1765   \fi}
1766 \def\LdfInit#1#2{%
1767   \chardef\atcatcode=\catcode`\@
1768   \catcode`\@=11\relax
1769   \chardef\eqcatcode=\catcode`\=
1770   \catcode`\==12\relax
1771   \expandafter\if\expandafter\@backslashchar
1772     \expandafter\@car\string#2\@nil
1773     \ifx#2\@undefined\else
1774       \ldf@quit{#1}%
1775     \fi
1776   \else
1777     \expandafter\ifx\csname#2\endcsname\relax\else
1778       \ldf@quit{#1}%
1779     \fi
1780   \fi

```

```
1781 \bbl@ldfinit}
```

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

```
1782 \def\ldf@quit#1{%
1783 \expandafter\main@language\expandafter{#1}%
1784 \catcode`\@=\atcatcode \let\atcatcode\relax
1785 \catcode`\==\eqcatcode \let\eqcatcode\relax
1786 \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.

```
1787 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1788 \bbl@afterlang
1789 \let\bbl@afterlang\relax
1790 \let\BabelModifiers\relax
1791 \let\bbl@screset\relax}%
1792 \def\ldf@finish#1{%
1793 \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1794 \loadlocalcfg{#1}%
1795 \fi
1796 \bbl@afterldf{#1}%
1797 \expandafter\main@language\expandafter{#1}%
1798 \catcode`\@=\atcatcode \let\atcatcode\relax
1799 \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 .

```
1800 \@onlypreamble\LdfInit
1801 \@onlypreamble\ldf@quit
1802 \@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.

```
1803 \def\main@language#1{%
1804 \def\bbl@main@language{#1}%
1805 \let\language\main@language % TODO. Set localename
1806 \bbl@id@assign
1807 \bbl@patterns{\language}}
```

We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the `\AtBeginDocument` is executed. Languages do not set `\pagedir`, so we set here for the whole document to the main `\bodydir`.

```
1808 \def\bbl@beforestart{%
1809 \bbl@usehooks{beforestart}{}%
1810 \global\let\bbl@beforestart\relax}
1811 \AtBeginDocument{%
1812 \@nameuse{bbl@beforestart}%
1813 \if@files
1814 \providecommand\babel@aux[2]{}%
1815 \immediate\write\@mainaux{%
1816 \string\providecommand\string\babel@aux[2]{}%
1817 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%}
```

```

1818 \fi
1819 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1820 \ifbbl@single % must go after the line above.
1821 \renewcommand\selectlanguage[1]{}%
1822 \renewcommand\foreignlanguage[2]{#2}%
1823 \global\let\babel@aux\@gobbletwo % Also as flag
1824 \fi
1825 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1826 \def\select@language@x#1{%
1827 \ifcase\bbl@select@type
1828 \bbl@ifsamestring\language#1\{\select@language{#1}}%
1829 \else
1830 \select@language{#1}%
1831 \fi}

```

9.5 Shorthands

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

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

```

1832 \bbl@trace{Shorhands}
1833 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1834 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1835 \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1836 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1837 \begingroup
1838 \catcode`#1\active
1839 \nfss@catcodes
1840 \ifnum\catcode`#1=\active
1841 \endgroup
1842 \bbl@add\nfss@catcodes{\@makeother#1}%
1843 \else
1844 \endgroup
1845 \fi
1846 \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.

```

1847 \def\bbl@remove@special#1{%
1848 \begingroup
1849 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1850 \else\noexpand##1\noexpand##2\fi}%
1851 \def\do{\x\do}%
1852 \def\@makeother{\x\@makeother}%
1853 \edef\x{\endgroup
1854 \def\noexpand\dospecials{\dospecials}%
1855 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1856 \def\noexpand\@sanitize{\@sanitize}%
1857 \fi}%
1858 \x}

```

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char⟨char⟩` to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char⟨char⟩` by default (`⟨char⟩` being the character to be made active). Later its definition can be changed to expand to `\active@char⟨char⟩` by calling `\bbl@activate{⟨char⟩}`.

For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines `"` as `\active@prefix " \active@char` (where the first `"` is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect " or \noexpand "` (ie, with the original `"`); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix " \normal@char`).

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, `\<level>@group`, `<level>@active` and `<next-level>@active` (except in system).

```
1859 \def\bbl@active@def#1#2#3#4{%
1860   \@namedef{#3#1}{%
1861     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1862       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1863     \else
1864       \bbl@afterfi\csname#2@sh@#1\endcsname
1865     \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.

```
1866   \long\@namedef{#3@arg#1}##1{%
1867     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1868       \bbl@afterelse\csname#4#1\endcsname##1%
1869     \else
1870       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1871     \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.

```
1872 \def\@initiate@active@char#1#2#3{%
1873   \bbl@ifunset{active@char\string#1}%
1874   {\bbl@withactive
1875     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1876   {}}
```

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`).

```
1877 \def\@initiate@active@char#1#2#3{%
1878   \bbl@csarg\edef\oricat@#2{\catcode`#2=\the\catcode`#2\relax}%
1879   \ifx#1\@undefined
1880     \bbl@csarg\edef\oridef@#2{\let\noexpand#1\noexpand\@undefined}%
1881   \else
1882     \bbl@csarg\let\oridef@#2#1%
1883     \bbl@csarg\edef\oridef@#2{%
```

```

1884 \let\noexpand#1%
1885 \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1886 \fi

```

If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define `\normal@char⟨char⟩` to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ') the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to "8000 *a posteriori*").

```

1887 \ifx#1#3\relax
1888 \expandafter\let\csname normal@char#2\endcsname#3%
1889 \else
1890 \bbl@info{Making #2 an active character}%
1891 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1892 \@namedef{normal@char#2}{%
1893 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1894 \else
1895 \@namedef{normal@char#2}{#3}%
1896 \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).

```

1897 \bbl@restoreactive{#2}%
1898 \AtBeginDocument{%
1899 \catcode`#2\active
1900 \if@filesw
1901 \immediate\write\@mainaux{\catcode`\string#2\active}%
1902 \fi}%
1903 \expandafter\bbl@add@special\csname#2\endcsname
1904 \catcode`#2\active
1905 \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⟩`).

```

1906 \let\bbl@tempa\@firstoftwo
1907 \if\string^#2%
1908 \def\bbl@tempa{\noexpand\textormath}%
1909 \else
1910 \ifx\bbl@mathnormal\@undefined\else
1911 \let\bbl@tempa\bbl@mathnormal
1912 \fi
1913 \fi
1914 \expandafter\edef\csname active@char#2\endcsname{%
1915 \bbl@tempa
1916 {\noexpand\if@safe@actives
1917 \noexpand\expandafter
1918 \expandafter\noexpand\csname normal@char#2\endcsname
1919 \noexpand\else
1920 \noexpand\expandafter

```

```

1921      \expandafter\noexpand\csname bbl@doactive#2\endcsname
1922      \noexpand\fi}%
1923      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1924      \bbl@csarg\edef{doactive#2}{%
1925      \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

`\active@prefix <char> \normal@char <char>`

(where `\active@char <char>` is *one* control sequence!).

```

1926      \bbl@csarg\edef{active@#2}{%
1927      \noexpand\active@prefix\noexpand#1%
1928      \expandafter\noexpand\csname active@char#2\endcsname}%
1929      \bbl@csarg\edef{normal@#2}{%
1930      \noexpand\active@prefix\noexpand#1%
1931      \expandafter\noexpand\csname normal@char#2\endcsname}%
1932      \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.

```

1933      \bbl@active@def#2\user@group{user@active}{language@active}%
1934      \bbl@active@def#2\language@group{language@active}{system@active}%
1935      \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.

```

1936      \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1937      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1938      \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1939      {\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.

```

1940      \if\string'#2%
1941      \let\prim@s\bbl@prim@s
1942      \let\active@math@prime#1%
1943      \fi
1944      \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}

```

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

```

1945      <<{*More package options}>> ≡
1946      \DeclareOption{math=active}{}
1947      \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1948      <</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*.

```

1949      \ifpackagewith{babel}{KeepShorthandsActive}%

```

```

1950 {\let\bbl@restoreactive\@gobble}%
1951 {\def\bbl@restoreactive#1{%
1952   \bbl@exp{%
1953     \\AfterBabelLanguage\\CurrentOption
1954     {\catcode`#1=\the\catcode`#1\relax}%
1955     \\AtEndOfPackage
1956     {\catcode`#1=\the\catcode`#1\relax}}}%
1957   \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.

```

1958 \def\bbl@sh@select#1#2{%
1959   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1960     \bbl@afterelse\bbl@scndcs
1961   \else
1962     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1963   \fi}

```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protects` the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar`: (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

1964 \begingroup
1965 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1966 {\gdef\active@prefix#1{%
1967   \ifx\protect\@typeset@protect
1968   \else
1969     \ifx\protect\@unexpandable@protect
1970       \noexpand#1%
1971     \else
1972       \protect#1%
1973     \fi
1974     \expandafter\@gobble
1975   \fi}}
1976 {\gdef\active@prefix#1{%
1977   \ifincsname
1978     \string#1%
1979     \expandafter\@gobble
1980   \else
1981     \ifx\protect\@typeset@protect
1982     \else
1983       \ifx\protect\@unexpandable@protect
1984         \noexpand#1%
1985       \else
1986         \protect#1%
1987       \fi
1988       \expandafter\expandafter\expandafter\@gobble
1989     \fi
1990   \fi}}
1991 \endgroup

```

`\if@safe@actives` In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char⟨char⟩`.

```

1992 \newif\if@safe@actives
1993 \@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.

```

1994 \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`.

`\bbl@deactivate`

```

1995 \def\bbl@activate#1{%
1996   \bbl@withactive{\expandafter\let\expandafter}#1%
1997   \csname bbl@active@\string#1\endcsname}
1998 \def\bbl@deactivate#1{%
1999   \bbl@withactive{\expandafter\let\expandafter}#1%
2000   \csname bbl@normal@\string#1\endcsname}

```

`\bbl@firstcs` These macros are used only as a trick when declaring shorthands.

`\bbl@scndcs`

```

2001 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2002 \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.

```

2003 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2004 \def\@decl@short#1#2#3\@nil#4{%
2005   \def\bbl@tempa{#3}%
2006   \ifx\bbl@tempa\@empty
2007     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2008     \bbl@ifunset{#1@sh@\string#2@}{}%
2009     {\def\bbl@tempa{#4}%
2010      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2011      \else
2012        \bbl@info
2013          {Redefining #1 shorthand \string#2\\%
2014           in language \CurrentOption}%
2015        \fi}%
2016     \@namedef{#1@sh@\string#2@}{#4}%
2017   \else
2018     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2019     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2020     {\def\bbl@tempa{#4}%
2021      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2022      \else
2023        \bbl@info
2024          {Redefining #1 shorthand \string#2\string#3\\%
2025           in language \CurrentOption}%
2026        \fi}%
2027     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2028   \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.

```

2029 \def\textormath{%
2030   \ifmmode
2031     \expandafter\@secondoftwo
2032   \else
2033     \expandafter\@firstoftwo
2034   \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’.

```

2035 \def\user@group{user}
2036 \def\language@group{english} % TODO. I don't like defaults
2037 \def\system@group{system}

```

`\useshorthands` This is the user level macro. It initializes and activates the character for use as a shorthand character (ie, it’s active in the preamble). Languages can deactivate shorthands, so a starred version is also provided which activates them always after the language has been switched.

```

2038 \def\useshorthands{%
2039   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
2040 \def\bb1@usesh@s#1{%
2041   \bb1@usesh@x
2042     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
2043     {#1}}
2044 \def\bb1@usesh@x#1#2{%
2045   \bb1@ifshorthand{#2}%
2046     {\def\user@group{user}%
2047       \initiate@active@char{#2}%
2048       #1%
2049       \bb1@activate{#2}}%
2050     {\bb1@error
2051       {Cannot declare a shorthand turned off (\string#2)}
2052       {Sorry, but you cannot use shorthands which have been\\%
2053         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 `\bb1@set@user@generic`); we make also sure {} and \protect are taken into account in this new top level.

```

2054 \def\user@language@group{user@\language@group}
2055 \def\bb1@set@user@generic#1#2{%
2056   \bb1@ifunset{user@generic@active#1}%
2057     {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
2058       \bb1@active@def#1\user@group{user@generic@active}{language@active}%
2059       \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2060         \expandafter\noexpand\csname normal@char#1\endcsname}%
2061       \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2062         \expandafter\noexpand\csname user@active#1\endcsname}}%
2063   \@empty}
2064 \newcommand\defineshorthand[3][user]{%
2065   \edef\bb1@tempa{\zap@space#1 \@empty}%
2066   \bb1@for\bb1@tempb\bb1@tempa{%
2067     \if*\expandafter\@car\bb1@tempb\@nil

```

```

2068 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2069 \@expandtwoargs
2070 \bbl@set@user@generic{\expandafter\string\car#2\@nil}\bbl@tempb
2071 \fi
2072 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

`\languageshorthands` A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing [TODO. Unclear].

```

2073 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

2074 \def\aliasshorthand#1#2{%
2075 \bbl@ifshorthand{#2}%
2076 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2077 \ifx\document\@notprerr
2078 \@notshorthand{#2}%
2079 \else
2080 \initiate@active@char{#2}%
2081 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2082 \csname active@char\string#1\endcsname
2083 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2084 \csname normal@char\string#1\endcsname
2085 \bbl@activate{#2}%
2086 \fi
2087 \fi}%
2088 {\bbl@error
2089 {Cannot declare a shorthand turned off (\string#2)}
2090 {Sorry, but you cannot use shorthands which have been\\%
2091 turned off in the package options}}}

```

`\@notshorthand`

```

2092 \def\@notshorthand#1{%
2093 \bbl@error{%
2094 The character '\string #1' should be made a shorthand character;\\%
2095 add the command \string\usesshorthands\string{#1\string} to
2096 the preamble.\\%
2097 I will ignore your instruction}%
2098 {You may proceed, but expect unexpected results}}

```

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

```

2099 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2100 \DeclareRobustCommand*\shorthandoff{%
2101 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2102 \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.

```

2103 \def\bb@switch@sh#1#2{%
2104   \ifx#2\@nnil\else
2105     \bb@ifunset{bb@active@\string#2}%
2106     {\bb@error
2107       {I cannot switch '\string#2' on or off--not a shorthand}%
2108       {This character is not a shorthand. Maybe you made\\%
2109         a typing mistake? I will ignore your instruction}}}%
2110     {\ifcase#1%
2111       \catcode`#2\relax
2112       \or
2113       \catcode`#2\active
2114       \or
2115       \csname bb@oricat@\string#2\endcsname
2116       \csname bb@oridef@\string#2\endcsname
2117       \fi}%
2118     \bb@afterfi\bb@switch@sh#1%
2119   \fi}

```

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

```

2120 \def\babelshorthand{\active@prefix\babelshorthand\bb@putsh}
2121 \def\bb@putsh#1{%
2122   \bb@ifunset{bb@active@\string#1}%
2123   {\bb@putsh@i#1\@empty\@nnil}%
2124   {\csname bb@active@\string#1\endcsname}}
2125 \def\bb@putsh@i#1#2\@nnil{%
2126   \csname\language\name @sh@\string#1@%
2127     \ifx\@empty#2\else\string#2@\fi\endcsname}
2128 \ifx\bb@opt@shorthands\@nnil\else
2129   \let\bb@s@initiate@active@char\initiate@active@char
2130   \def\initiate@active@char#1{%
2131     \bb@ifshorthand{#1}{\bb@s@initiate@active@char{#1}}{}}
2132   \let\bb@s@switch@sh\bb@switch@sh
2133   \def\bb@switch@sh#1#2{%
2134     \ifx#2\@nnil\else
2135       \bb@afterfi
2136       \bb@ifshorthand{#2}{\bb@s@switch@sh#1{#2}}{\bb@switch@sh#1}%
2137     \fi}
2138   \let\bb@s@activate\bb@activate
2139   \def\bb@activate#1{%
2140     \bb@ifshorthand{#1}{\bb@s@activate{#1}}{}}
2141   \let\bb@s@deactivate\bb@deactivate
2142   \def\bb@deactivate#1{%
2143     \bb@ifshorthand{#1}{\bb@s@deactivate{#1}}{}}
2144 \fi

```

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

```

2145 \newcommand\ifbabelshorthand[3]{\bb@ifunset{bb@active@\string#1}{#3}{#2}}

```

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

```

2146 \def\bbl@prim@s{%
2147   \prime\futurelet\@let@token\bbl@pr@m@s}
2148 \def\bbl@if@primes#1#2{%
2149   \ifx#1\@let@token
2150     \expandafter\@firstoftwo
2151   \else\ifx#2\@let@token
2152     \bbl@afterelse\expandafter\@firstoftwo
2153   \else
2154     \bbl@afterfi\expandafter\@secondoftwo
2155   \fi\fi}
2156 \begingroup
2157   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2158   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
2159   \lowercase{%
2160     \gdef\bbl@pr@m@s{%
2161       \bbl@if@primes"%
2162         \pr@@s
2163         {\bbl@if@primes*\^{\pr@@t\egroup}}}}
2164 \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).

```

2165 \initiate@active@char{~}
2166 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2167 \bbl@activate{~}

```

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

```

2168 \expandafter\def\csname OT1dqpos\endcsname{127}
2169 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

2170 \ifx\f@encoding\@undefined
2171   \def\f@encoding{OT1}
2172 \fi

```

9.6 Language attributes

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

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

```

2173 \bbl@trace{Language attributes}
2174 \newcommand\languageattribute[2]{%
2175   \def\bbl@tempc{#1}%
2176   \bbl@fixname\bbl@tempc
2177   \bbl@iflanguage\bbl@tempc{%
2178     \bbl@vforeach{#2}{%

```

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

```

2179 \if\bbl@known@attrs\@undefined
2180 \in@false
2181 \else
2182 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
2183 \fi
2184 \ifin@
2185 \bbl@warning{%
2186 You have more than once selected the attribute '##1'\%
2187 for language #1. Reported}%
2188 \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.

```

2189 \bbl@exp{%
2190 \\\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
2191 \edef\bbl@tempa{\bbl@tempc-##1}%
2192 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2193 {\csname\bbl@tempc @attr##1\endcsname}%
2194 {\@attrerr{\bbl@tempc}{##1}}%
2195 \fi}}
2196 \@onlypreamble\languageattribute

```

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

```

2197 \newcommand*{\@attrerr}[2]{%
2198 \bbl@error
2199 {The attribute #2 is unknown for language #1.}%
2200 {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}`.

```

2201 \def\bbl@declare@ttribute#1#2#3{%
2202 \bbl@xin@{,#2,}{,\BabelModifiers,}%
2203 \ifin@
2204 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2205 \fi
2206 \bbl@add@list\bbl@attributes{#1-#2}%
2207 \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret \TeX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```

2208 \def\bbl@ifattributeset#1#2#3#4{%

```

```

2209 \ifx\bb1@known@attribs\@undefined
2210 \in@false
2211 \else
2212 \bb1@xin@{,#1-#2,}{,\bb1@known@attribs,}%
2213 \fi
2214 \ifin@
2215 \bb1@afterelse#3%
2216 \else
2217 \bb1@afterfi#4%
2218 \fi
2219 }

```

`\bb1@ifknown@ttrib` An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of `\bb1@tempa` is changed. Finally we execute `\bb1@tempa`.

```

2220 \def\bb1@ifknown@ttrib#1#2{%
2221 \let\bb1@tempa\@secondoftwo
2222 \bb1@loopx\bb1@tempb{#2}{%
2223 \expandafter\in@expandafter{\expandafter,\bb1@tempb,}{,#1,}%
2224 \ifin@
2225 \let\bb1@tempa\@firstoftwo
2226 \else
2227 \fi}%
2228 \bb1@tempa
2229 }

```

`\bb1@clear@ttribs` This macro removes all the attribute code from \LaTeX 's memory at `\begin{document}` time (if any is present).

```

2230 \def\bb1@clear@ttribs{%
2231 \ifx\bb1@attributes\@undefined\else
2232 \bb1@loopx\bb1@tempa{\bb1@attributes}{%
2233 \expandafter\bb1@clear@ttrib\bb1@tempa.
2234 }%
2235 \let\bb1@attributes\@undefined
2236 \fi}
2237 \def\bb1@clear@ttrib#1-#2.{%
2238 \expandafter\let\csname#1@attr#2\endcsname\@undefined}
2239 \AtBeginDocument{\bb1@clear@ttribs}

```

9.7 Support for saving macro definitions

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

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.

`\babel@beginsave`

```

2240 \bb1@trace{Macros for saving definitions}
2241 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```
2242 \newcount\babel@savecnt
2243 \babel@beginsave
```

`\babel@save` The macro `\babel@save⟨csize⟩` saves the current meaning of the control sequence `⟨csize⟩` to `\originalTeX`³¹. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive.

```
2244 \def\babel@save#1{%
2245   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2246   \toks@\expandafter{\originalTeX\let#1=}
2247   \bbl@exp{%
2248     \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}
2249   \advance\babel@savecnt@ne}
2250 \def\babel@savevariable#1{%
2251   \toks@\expandafter{\originalTeX #1=}
2252   \bbl@exp{\def\originalTeX{\the\toks@the#1\relax}}}
```

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

```
2253 \def\bbl@frenchspacing{%
2254   \ifnum\the\sfcode\.\!=\@m
2255     \let\bbl@nonfrenchspacing\relax
2256   \else
2257     \frenchspacing
2258     \let\bbl@nonfrenchspacing\nonfrenchspacing
2259   \fi}
2260 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

9.8 Short tags

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

```
2261 \bbl@trace{Short tags}
2262 \def\babeltags#1{%
2263   \edef\bbl@tempa{\zap@space#1 \@empty}%
2264   \def\bbl@tempb##1=##2\@{}%
2265   \edef\bbl@tempc{%
2266     \noexpand\newcommand
2267     \expandafter\noexpand\csname ##1\endcsname{%
2268       \noexpand\protect
2269       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2270     \noexpand\newcommand
2271     \expandafter\noexpand\csname text##1\endcsname{%
2272       \noexpand\foreignlanguage{##2}}
2273   \bbl@tempc}%
2274   \bbl@for\bbl@tempa\bbl@tempa{%
2275     \expandafter\bbl@tempb\bbl@tempa\@{}}
```

³¹`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

9.9 Hyphens

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

```

2276 \bbl@trace{Hyphens}
2277 \@onlypreamble\babelhyphenation
2278 \AtEndOfPackage{%
2279   \newcommand\babelhyphenation[2][\@empty]{%
2280     \ifx\bbl@hyphenation@ \relax
2281       \let\bbl@hyphenation@ \@empty
2282     \fi
2283     \ifx\bbl@hyphlist \@empty \else
2284       \bbl@warning{%
2285         You must not intermingle \string\selectlanguage\space and\%
2286         \string\babelhyphenation\space or some exceptions will not\%
2287         be taken into account. Reported}%
2288       \fi
2289       \ifx \@empty #1%
2290         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2291       \else
2292         \bbl@vforeach{#1}{%
2293           \def\bbl@tempa{##1}%
2294           \bbl@fixname\bbl@tempa
2295           \bbl@iflanguage\bbl@tempa{%
2296             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2297               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2298               \@empty
2299               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2300               #2}}}%
2301         \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`³².

```

2302 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
2303 \def\bbl@t@one{T1}
2304 \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`.

```

2305 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2306 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2307 \def\bbl@hyphen{%
2308   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2309 \def\bbl@hyphen@i#1#2{%
2310   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2311   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{\#2}}}%
2312   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

³² \TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

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.

```
2313 \def\bbl@usehyphen#1{%
2314   \leavevmode
2315   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2316   \nobreak\hskip\z@skip}
2317 \def\bbl@usehyphen#1{%
2318   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
2319 \def\bbl@hyphenchar{%
2320   \ifnum\hyphenchar\font=\m@ne
2321     \babeinullhyphen
2322   \else
2323     \char\hyphenchar\font
2324   \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.

```
2325 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2326 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2327 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2328 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2329 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2330 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2331 \def\bbl@hy@repeat{%
2332   \bbl@usehyphen{%
2333     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2334 \def\bbl@hy@repeat{%
2335   \bbl@usehyphen{%
2336     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2337 \def\bbl@hy@empty{\hskip\z@skip}
2338 \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.

```
2339 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}
```

9.10 Multiencoding strings

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

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

```
2340 \bbl@trace{Multiencoding strings}
2341 \def\bbl@tglobal#1{\global\let#1#1}
2342 \def\bbl@recatcode#1{% TODO. Used only once?
2343   \@tempcnta="7F
2344   \def\bbl@tempa{%
2345     \ifnum\@tempcnta>"FF\else
2346       \catcode\@tempcnta=#1\relax
2347       \advance\@tempcnta@\ne
2348     \expandafter\bbl@tempa
```

```

2349   \fi}%
2350   \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).

```

2351 \@ifpackagewith{babel}{nocase}%
2352   {\let\bbl@patchuclc\relax}%
2353   {\def\bbl@patchuclc{%
2354     \global\let\bbl@patchuclc\relax
2355     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2356     \gdef\bbl@uclc##1{%
2357       \let\bbl@encoded\bbl@encoded@uclc
2358       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2359       {##1}%
2360       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2361         \csname\language @bbl@uclc\endcsname}%
2362       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2363     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2364     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2365 <<More package options>> ≡
2366 \DeclareOption{nocase}{}
2367 <</More package options>>

```

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

```

2368 <<More package options>> ≡
2369 \let\bbl@opt@strings\@nnil % accept strings=value
2370 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2371 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2372 \def\BabelStringsDefault{generic}
2373 <</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.

```

2374 \@onlypreamble\StartBabelCommands
2375 \def\StartBabelCommands{%
2376   \begingroup
2377   \bbl@recatcode{11}%
2378   <<Macros local to BabelCommands>>
2379   \def\bbl@provstring##1##2{%
2380     \providecommand##1{##2}%
2381     \bbl@tglobal##1}%
2382   \global\let\bbl@scafter\@empty
2383   \let\StartBabelCommands\bbl@startcmds
2384   \ifx\BabelLanguages\relax
2385     \let\BabelLanguages\CurrentOption
2386   \fi

```

```

2387 \begingroup
2388 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2389 \StartBabelCommands}
2390 \def\bbl@startcmds{%
2391   \ifx\bbl@screset\@nnil\else
2392     \bbl@usehooks{stopcommands}{}%
2393   \fi
2394 \endgroup
2395 \begingroup
2396 \@ifstar
2397   {\ifx\bbl@opt@strings\@nnil
2398     \let\bbl@opt@strings\BabelStringsDefault
2399     \fi
2400     \bbl@startcmds@i}%
2401   \bbl@startcmds@i}
2402 \def\bbl@startcmds@i#1#2{%
2403   \edef\bbl@L{\zap@space#1 \@empty}%
2404   \edef\bbl@G{\zap@space#2 \@empty}%
2405   \bbl@startcmds@ii}
2406 \let\bbl@startcommands\StartBabelCommands

```

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

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

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

```

2407 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2408   \let\SetString@gobbletwo
2409   \let\bbl@stringdef@gobbletwo
2410   \let\AfterBabelCommands@gobble
2411   \ifx\@empty#1%
2412     \def\bbl@sc@label{generic}%
2413     \def\bbl@encstring##1##2{%
2414       \ProvideTextCommandDefault##1{##2}%
2415       \bbl@toglobal##1%
2416       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2417     \let\bbl@sctest\in@true
2418   \else
2419     \let\bbl@sc@charset\space % <- zapped below
2420     \let\bbl@sc@fontenc\space % <- " "
2421     \def\bbl@tempa##1=##2\@nil{%
2422       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2423     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2424     \def\bbl@tempa##1 ##2{% space -> comma
2425       ##1%
2426       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2427     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2428     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2429     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2430     \def\bbl@encstring##1##2{%
2431       \bbl@foreach\bbl@sc@fontenc{%
2432         \bbl@ifunset{T@####1}%
2433       }%

```

```

2434         {\ProvideTextCommand###1{####1}{##2}%
2435         \bbl@tglobal##1%
2436         \expandafter
2437         \bbl@tglobal\csname####1\string##1\endcsname}}}%
2438     \def\bbl@sctest{%
2439         \bbl@xin@{\,\bbl@opt@strings,}{\,\bbl@sc@label,\bbl@sc@fontenc,}}%
2440 \fi
2441 \ifx\bbl@opt@strings\@nnil      % ie, no strings key -> defaults
2442 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2443     \let\AfterBabelCommands\bbl@aftercmds
2444     \let\SetString\bbl@setstring
2445     \let\bbl@stringdef\bbl@encstring
2446 \else      % ie, strings=value
2447 \bbl@sctest
2448 \ifin@
2449     \let\AfterBabelCommands\bbl@aftercmds
2450     \let\SetString\bbl@setstring
2451     \let\bbl@stringdef\bbl@provstring
2452 \fi\fi\fi
2453 \bbl@scswitch
2454 \ifx\bbl@G\@empty
2455     \def\SetString##1##2{%
2456         \bbl@error{Missing group for string \string##1}%
2457         {You must assign strings to some category, typically\\%
2458         captions or extras, but you set none}}%
2459 \fi
2460 \ifx\@empty#1%
2461     \bbl@usehooks{defaultcommands}}}%
2462 \else
2463     \@expandtwoargs
2464     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
2465 \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).

```

2466 \def\bbl@forlang#1#2{%
2467     \bbl@for#1\bbl@L{%
2468         \bbl@xin@{,#1,}{\,\BabelLanguages,}%
2469         \ifin@#2\relax\fi}}
2470 \def\bbl@scswitch{%
2471     \bbl@forlang\bbl@tempa{%
2472         \ifx\bbl@G\@empty\else
2473             \ifx\SetString\@gobbletwo\else
2474                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2475                 \bbl@xin@{\,\bbl@GL,}{\,\bbl@screset,}%
2476             \ifin@\else
2477                 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2478                 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2479             \fi
2480         \fi
2481     \fi}}

```

```

2482 \AtEndOfPackage{%
2483   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
2484   \let\bbl@scswitch\relax}
2485 \@onlypreamble\EndBabelCommands
2486 \def\EndBabelCommands{%
2487   \bbl@usehooks{stopcommands}{}%
2488   \endgroup
2489   \endgroup
2490   \bbl@scafter}
2491 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

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

```

2492 \def\bbl@setstring#1#2{%
2493   \bbl@forlang\bbl@tempa{%
2494     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2495     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2496     {\global\expandafter % TODO - con \bbl@exp ?
2497       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2498       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
2499     }%
2500   \def\BabelString{\#2}%
2501   \bbl@usehooks{stringprocess}{}%
2502   \expandafter\bbl@stringdef
2503   \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.

```

2504 \ifx\bbl@opt@strings\relax
2505   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2506   \bbl@patchuclc
2507   \let\bbl@encoded\relax
2508   \def\bbl@encoded@uclc#1{%
2509     \@inmathwarn#1%
2510     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2511       \expandafter\ifx\csname ?\string#1\endcsname\relax
2512         \TextSymbolUnavailable#1%
2513       \else
2514         \csname ?\string#1\endcsname
2515       \fi
2516     \else
2517       \csname\cf@encoding\string#1\endcsname
2518     \fi}
2519 \else
2520   \def\bbl@scset#1#2{\def#1{\#2}}
2521 \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.

```

2522 <<*Macros local to BabelCommands>> ≡
2523 \def\SetStringLoop##1##2{%
2524   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2525   \count@\z@
2526   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2527     \advance\count@ \@ne
2528     \toks@\expandafter{\bbl@tempa}%
2529     \bbl@exp{%
2530       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2531       \count@=\the\count@\relax}}}%
2532 <</Macros local to BabelCommands>>

```

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

```

2533 \def\bbl@aftercmds#1{%
2534   \toks@\expandafter{\bbl@scafter#1}%
2535   \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.

```

2536 <<*Macros local to BabelCommands>> ≡
2537 \newcommand\SetCase[3][{}{%
2538   \bbl@patchuclc
2539   \bbl@forlang\bbl@tempa{%
2540     \expandafter\bbl@encstring
2541     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2542     \expandafter\bbl@encstring
2543     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2544     \expandafter\bbl@encstring
2545     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2546 <</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.

```

2547 <<*Macros local to BabelCommands>> ≡
2548 \newcommand\SetHyphenMap[1][{}{%
2549   \bbl@forlang\bbl@tempa{%
2550     \expandafter\bbl@stringdef
2551     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2552 <</Macros local to BabelCommands>>

```

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

```

2553 \newcommand\BabelLower[2]{% one to one.
2554   \ifnum\lccode#1=#2\else
2555     \babel@savevariable{\lccode#1}%
2556     \lccode#1=#2\relax
2557   \fi}
2558 \newcommand\BabelLowerMM[4]{% many-to-many
2559   \@tempcnta=#1\relax
2560   \@tempcntb=#4\relax
2561   \def\bbl@tempa{%
2562     \ifnum\@tempcnta>#2\else
2563       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2564       \advance\@tempcnta#3\relax
2565       \advance\@tempcntb#3\relax
2566       \expandafter\bbl@tempa

```

```

2567 \fi}%
2568 \bbl@tempa}
2569 \newcommand\BabelLowerM0[4]{% many-to-one
2570 \@tempcnta=#1\relax
2571 \def\bbl@tempa{%
2572 \ifnum\@tempcnta>#2\else
2573 \expandafter\BabelLower{\the\@tempcnta}{#4}%
2574 \advance\@tempcnta#3
2575 \expandafter\bbl@tempa
2576 \fi}%
2577 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2578 <<More package options>> ≡
2579 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2580 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2581 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2582 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2583 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2584 <</More package options>>

```

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

```

2585 \AtEndOfPackage{%
2586 \ifx\bbl@opt@hyphenmap\undefined
2587 \bbl@xin@{,}{\bbl@language@opts}%
2588 \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
2589 \fi}

```

9.11 Macros common to a number of languages

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

```

2590 \bbl@trace{Macros related to glyphs}
2591 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2592 \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2593 \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.

```

2594 \def\save@sf@q#1{\leavevmode
2595 \begingroup
2596 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2597 \endgroup}

```

9.12 Making glyphs available

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

9.12.1 Quotation marks

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

```

2598 \ProvideTextCommand{\quotedblbase}{OT1}{%
2599 \save@sf@q{\set@low@box{\textquotedblright\}%
2600 \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.

```
2601 \ProvideTextCommandDefault{\quotedblbase}{%
2602   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2603 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2604   \save@sf@q{\set@low@box{\textquoteright\}%
2605     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
2606 \ProvideTextCommandDefault{\quotesinglbase}{%
2607   \UseTextSymbol{OT1}{\quotesinglbase}}
```

`\guillemetleft` The guillemet characters are not available in OT1 encoding. They are faked. (Wrong names
`\guillemetright` with o preserved for compatibility.)

```
2608 \ProvideTextCommand{\guillemetleft}{OT1}{%
2609   \ifmmode
2610     \ll
2611   \else
2612     \save@sf@q{\nobreak
2613       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2614   \fi}
2615 \ProvideTextCommand{\guillemetright}{OT1}{%
2616   \ifmmode
2617     \gg
2618   \else
2619     \save@sf@q{\nobreak
2620       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2621   \fi}
2622 \ProvideTextCommand{\guillemotleft}{OT1}{%
2623   \ifmmode
2624     \ll
2625   \else
2626     \save@sf@q{\nobreak
2627       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2628   \fi}
2629 \ProvideTextCommand{\guillemotright}{OT1}{%
2630   \ifmmode
2631     \gg
2632   \else
2633     \save@sf@q{\nobreak
2634       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2635   \fi}
```

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

```
2636 \ProvideTextCommandDefault{\guillemetleft}{%
2637   \UseTextSymbol{OT1}{\guillemetleft}}
2638 \ProvideTextCommandDefault{\guillemetright}{%
2639   \UseTextSymbol{OT1}{\guillemetright}}
2640 \ProvideTextCommandDefault{\guillemotleft}{%
2641   \UseTextSymbol{OT1}{\guillemotleft}}
2642 \ProvideTextCommandDefault{\guillemotright}{%
2643   \UseTextSymbol{OT1}{\guillemotright}}
```


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

```
2644 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2645   \ifmmode
2646     <%
2647   \else
2648     \save@sf@q{\nobreak
2649       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2650   \fi}
2651 \ProvideTextCommand{\guilsinglright}{OT1}{%
2652   \ifmmode
2653     >%
2654   \else
2655     \save@sf@q{\nobreak
2656       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2657   \fi}
```

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

```
2658 \ProvideTextCommandDefault{\guilsinglleft}{%
2659   \UseTextSymbol{OT1}{\guilsinglleft}}
2660 \ProvideTextCommandDefault{\guilsinglright}{%
2661   \UseTextSymbol{OT1}{\guilsinglright}}
```

9.12.2 Letters

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

```
2662 \DeclareTextCommand{\ij}{OT1}{%
2663   i\kern-0.02em\bbl@allowhyphens j}
2664 \DeclareTextCommand{\IJ}{OT1}{%
2665   I\kern-0.02em\bbl@allowhyphens J}
2666 \DeclareTextCommand{\ij}{T1}{\char188}
2667 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2668 \ProvideTextCommandDefault{\ij}{%
2669   \UseTextSymbol{OT1}{\ij}}
2670 \ProvideTextCommandDefault{\IJ}{%
2671   \UseTextSymbol{OT1}{\IJ}}
```

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

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

```
2672 \def\crrtic@{\hrule height0.1ex width0.3em}
2673 \def\crrtic@{\hrule height0.1ex width0.33em}
2674 \def\ddj@{%
2675   \setbox0\hbox{d}\dimen@=\ht0
2676   \advance\dimen@1ex
2677   \dimen@.45\dimen@
2678   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2679   \advance\dimen@ii.5ex
2680   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2681 \def\DDJ@{%
2682   \setbox0\hbox{D}\dimen@=.55\ht0
2683   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
```

```

2684 \advance\dimen@ii.15ex % correction for the dash position
2685 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2686 \dimen\thr@@\expandafter\rem\pt\the\fontdimen7\font\dimen@
2687 \leavevmode\rlap{\raise\dimen@hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2688 %
2689 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2690 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2691 \ProvideTextCommandDefault{\dj}{%
2692 \UseTextSymbol{OT1}{\dj}}
2693 \ProvideTextCommandDefault{\DJ}{%
2694 \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.

```

2695 \DeclareTextCommand{\SS}{OT1}{SS}
2696 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

9.12.3 Shorthands for quotation marks

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

\glq The ‘german’ single quotes.

```

\grq 2697 \ProvideTextCommandDefault{\glq}{%
2698 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2699 \ProvideTextCommand{\grq}{T1}{%
2700 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2701 \ProvideTextCommand{\grq}{TU}{%
2702 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2703 \ProvideTextCommand{\grq}{OT1}{%
2704 \save@sf@q{\kern-.0125em
2705 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2706 \kern.07em\relax}}
2707 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 2708 \ProvideTextCommandDefault{\glqq}{%
2709 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2710 \ProvideTextCommand{\grqq}{T1}{%
2711 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2712 \ProvideTextCommand{\grqq}{TU}{%
2713 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2714 \ProvideTextCommand{\grqq}{OT1}{%
2715 \save@sf@q{\kern-.07em
2716 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2717 \kern.07em\relax}}
2718 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.
`\frq` 2719 \ProvideTextCommandDefault{\flq}{%
 2720 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
 2721 \ProvideTextCommandDefault{\frq}{%
 2722 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

`\flqq` The ‘french’ double guillemets.
`\frqq` 2723 \ProvideTextCommandDefault{\flqq}{%
 2724 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
 2725 \ProvideTextCommandDefault{\frqq}{%
 2726 \textormath{\guillemetright}{\mbox{\guillemetright}}}

9.12.4 Umlauts and tremas

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

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

```
2727 \def\umlauthigh{%
2728   \def\bbl@umlauta##1{\leavevmode\bgroup%
2729     \expandafter\accent\csname\f@encoding dqpos\endcsname
2730     ##1\bbl@allowhyphens\egroup}%
2731   \let\bbl@umlaute\bbl@umlauta}
2732 \def\umlautlow{%
2733   \def\bbl@umlauta{\protect\lower@umlaut}}
2734 \def\umlautelow{%
2735   \def\bbl@umlaute{\protect\lower@umlaut}}
2736 \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.

```
2737 \expandafter\ifx\csname U@D\endcsname\relax
2738   \csname newdimen\endcsname\U@D
2739 \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.

```
2740 \def\lower@umlaut#1{%
2741   \leavevmode\bgroup
2742   \U@D 1ex%
2743   {\setbox\z@\hbox{%
2744     \expandafter\char\csname\f@encoding dqpos\endcsname}%
2745     \dimen@ -.45ex\advance\dimen@\ht\z@
2746     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2747   \expandafter\accent\csname\f@encoding dqpos\endcsname
```

```

2748 \fontdimen5\font\U@D #1%
2749 \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).

```

2750 \AtBeginDocument{%
2751 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2752 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2753 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2754 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2755 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2756 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2757 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2758 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2759 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2760 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2761 \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty `\language` is defined. Currently used in Amharic.

```

2762 \ifx\l@english\@undefined
2763 \chardef\l@english\z@
2764 \fi
2765 % The following is used to cancel rules in ini files (see Amharic).
2766 \ifx\l@babelnohyphens\@undefined
2767 \newlanguage\l@babelnohyphens
2768 \fi

```

9.13 Layout

Work in progress.

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

```

2769 \bbl@trace{Bidi layout}
2770 \providecommand\IfBabelLayout[3]{#3}%
2771 \newcommand\BabelPatchSection[1]{%
2772 \@ifundefined{#1}{%
2773 \bbl@exp{\let\<bbl@ss@#1>\<#1>%
2774 \@namedef{#1}{%
2775 \@ifstar{\bbl@presec@s{#1}}%
2776 {\@dblarg{\bbl@presec@x{#1}}}}}%
2777 \def\bbl@presec@x#1[#2]#3{%
2778 \bbl@exp{%
2779 \\\select@language@x{\bbl@main@language}%
2780 \\\bbl@cs{sspre@#1}%
2781 \\\bbl@cs{ss@#1}%
2782 [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2783 {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2784 \\\select@language@x{\language}}}%
2785 \def\bbl@presec@s#1#2{%
2786 \bbl@exp{%
2787 \\\select@language@x{\bbl@main@language}%

```

```

2788   \\\bbl@cs{sspre@#1}%
2789   \\\bbl@cs{ss@#1}*%
2790   {\foreignlanguage{\language}{\unexpanded{#2}}}%
2791   \select@language@x{\language}}}%
2792 \IfBabelLayout{sectioning}%
2793   {\BabelPatchSection{part}%
2794    \BabelPatchSection{chapter}%
2795    \BabelPatchSection{section}%
2796    \BabelPatchSection{subsection}%
2797    \BabelPatchSection{subsubsection}%
2798    \BabelPatchSection{paragraph}%
2799    \BabelPatchSection{subparagraph}%
2800    \def\babel@toc#1{%
2801      \select@language@x{\bbl@main@language}}}%
2802 \IfBabelLayout{captions}%
2803   {\BabelPatchSection{caption}}}%

```

9.14 Load engine specific macros

```

2804 \bbl@trace{Input engine specific macros}
2805 \ifcase\bbl@engine
2806   \input txtbabel.def
2807 \or
2808   \input luababel.def
2809 \or
2810   \input xebabel.def
2811 \fi

```

9.15 Creating and modifying languages

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

```

2812 \bbl@trace{Creating languages and reading ini files}
2813 \newcommand\babelprovide[2][{}]{%
2814   \let\bbl@savelangname\language
2815   \edef\bbl@savelocaleid{\the\localeid}%
2816   % Set name and locale id
2817   \edef\language{#2}%
2818   % \global\@namedef{\bbl@lcname@#2}{#2}%
2819   \bbl@id@assign
2820   \let\bbl@KVP@captions\@nil
2821   \let\bbl@KVP@import\@nil
2822   \let\bbl@KVP@main\@nil
2823   \let\bbl@KVP@script\@nil
2824   \let\bbl@KVP@language\@nil
2825   \let\bbl@KVP@hyphenrules\@nil % only for provide@new
2826   \let\bbl@KVP@mapfont\@nil
2827   \let\bbl@KVP@maparabic\@nil
2828   \let\bbl@KVP@mapdigits\@nil
2829   \let\bbl@KVP@intraspace\@nil
2830   \let\bbl@KVP@intrapenalty\@nil
2831   \let\bbl@KVP@onchar\@nil
2832   \let\bbl@KVP@alph\@nil
2833   \let\bbl@KVP@Alph\@nil
2834   \let\bbl@KVP@labels\@nil
2835   \bbl@csarg\let{KVP@labels*}\@nil
2836   \bbl@forkv{#1}{% TODO - error handling

```

```

2837 \in@/{/}{##1}%
2838 \ifin@
2839 \bbl@renewinikey##1\@{##2}%
2840 \else
2841 \bbl@csarg\def{KVP@##1}{##2}%
2842 \fi}%
2843 % == import, captions ==
2844 \ifx\bbl@KVP@import\@nil\else
2845 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2846 {\ifx\bbl@initoload\relax
2847 \begingroup
2848 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2849 \bbl@input@ini{##2}%
2850 \endgroup
2851 \else
2852 \xdef\bbl@KVP@import{\bbl@initoload}%
2853 \fi}%
2854 {}%
2855 \fi
2856 \ifx\bbl@KVP@captions\@nil
2857 \let\bbl@KVP@captions\bbl@KVP@import
2858 \fi
2859 % Load ini
2860 \bbl@ifunset{date#2}%
2861 {\bbl@provide@new{##2}}%
2862 {\bbl@ifblank{##1}%
2863 {\bbl@error
2864 {If you want to modify `#2' you must tell how in\\%
2865 the optional argument. See the manual for the\\%
2866 available options.}%
2867 {Use this macro as documented}}%
2868 {\bbl@provide@renew{##2}}}%
2869 % Post tasks
2870 \bbl@exp{\bbl@babelensure[exclude=\\today]{##2}}%
2871 \bbl@ifunset{\bbl@ensure@\language}%
2872 {\bbl@exp{%
2873 \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2874 \\\foreignlanguage{\language}%
2875 {###1}}}%
2876 {}%
2877 \bbl@exp{%
2878 \\\bbl@toglobal\<bbl@ensure@\language>%
2879 \\\bbl@toglobal\<bbl@ensure@\language\space>%
2880 % At this point all parameters are defined if 'import'. Now we
2881 % execute some code depending on them. But what about if nothing was
2882 % imported? We just load the very basic parameters.
2883 \bbl@load@basic{##2}%
2884 % == script, language ==
2885 % Override the values from ini or defines them
2886 \ifx\bbl@KVP@script\@nil\else
2887 \bbl@csarg\edef{sname@##2}{\bbl@KVP@script}%
2888 \fi
2889 \ifx\bbl@KVP@language\@nil\else
2890 \bbl@csarg\edef{lname@##2}{\bbl@KVP@language}%
2891 \fi
2892 % == onchar ==
2893 \ifx\bbl@KVP@onchar\@nil\else
2894 \bbl@luahyphenate
2895 \directlua{

```

```

2896     if Babel.locale_mapped == nil then
2897         Babel.locale_mapped = true
2898         Babel.linebreaking.add_before(Babel.locale_map)
2899         Babel.loc_to_scr = {}
2900         Babel.chr_to_loc = Babel.chr_to_loc or {}
2901     end}%
2902 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2903 \ifin@
2904     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2905         \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2906     \fi
2907     \bbl@exp{\bbl@add\bbl@starthyphens
2908         {\bbl@patterns@lua{\language}}}%
2909     % TODO - error/warning if no script
2910     \directlua{
2911         if Babel.script_blocks['\bbl@cl{sbc}'] then
2912             Babel.loc_to_scr[\the\localeid] =
2913                 Babel.script_blocks['\bbl@cl{sbc}']
2914             Babel.locale_props[\the\localeid].lc = \the\localeid\space
2915             Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@language}\space
2916         end
2917     }%
2918 \fi
2919 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2920 \ifin@
2921     \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}{}%
2922     \bbl@ifunset{bbl@wdir@language}{\bbl@provide@dirs{language}}{}%
2923     \directlua{
2924         if Babel.script_blocks['\bbl@cl{sbc}'] then
2925             Babel.loc_to_scr[\the\localeid] =
2926                 Babel.script_blocks['\bbl@cl{sbc}']
2927         end}%
2928     \ifx\bbl@mapselect\@undefined
2929         \AtBeginDocument{%
2930             \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
2931             {\selectfont}}%
2932         \def\bbl@mapselect{%
2933             \let\bbl@mapselect\relax
2934             \edef\bbl@prefontid{\fontid\font}}%
2935         \def\bbl@mapdir##1{%
2936             {\def\language{##1}%
2937             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2938             \bbl@switchfont
2939             \directlua{
2940                 Babel.locale_props[\the\csname bbl@id@##1\endcsname]
2941                     ['\bbl@prefontid'] = \fontid\font\space}}}%
2942         \fi
2943         \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{language}}}%
2944     \fi
2945     % TODO - catch non-valid values
2946 \fi
2947 % == mapfont ==
2948 % For bidi texts, to switch the font based on direction
2949 \ifx\bbl@KVP@mapfont\@nil\else
2950     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2951     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
2952         mapfont. Use 'direction'.%
2953         {See the manual for details.}}}%
2954     \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}{}%

```

```

2955 \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2956 \ifx\bbl@mapselect\@undefined
2957 \AtBeginDocument{%
2958 \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2959 {\selectfont}}%
2960 \def\bbl@mapselect{%
2961 \let\bbl@mapselect\relax
2962 \edef\bbl@prefontid{\fontid\font}}%
2963 \def\bbl@mapdir##1{%
2964 {\def\languagename{##1}%
2965 \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2966 \bbl@switchfont
2967 \directlua{Babel.fontmap
2968 [\the\csname bbl@wdir@##1\endcsname]%
2969 [\bbl@prefontid]=\fontid\font}}}%
2970 \fi
2971 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
2972 \fi
2973 % == intraspace, intrapenalty ==
2974 % For CJK, East Asian, Southeast Asian, if interspace in ini
2975 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2976 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2977 \fi
2978 \bbl@provide@intraspace
2979 % == hyphenate.other.locale ==
2980 \bbl@ifunset{\bbl@hyotl@\languagename}{}%
2981 {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{ },}%
2982 \bbl@startcommands*\languagename}{}%
2983 \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2984 \ifcase\bbl@engine
2985 \ifnum##1<257
2986 \SetHyphenMap{\BabelLower{##1}{##1}}%
2987 \fi
2988 \else
2989 \SetHyphenMap{\BabelLower{##1}{##1}}%
2990 \fi}%
2991 \bbl@endcommands}%
2992 % == hyphenate.other.script ==
2993 \bbl@ifunset{\bbl@hyots@\languagename}{}%
2994 {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{ },}%
2995 \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2996 \ifcase\bbl@engine
2997 \ifnum##1<257
2998 \global\lccode##1=##1\relax
2999 \fi
3000 \else
3001 \global\lccode##1=##1\relax
3002 \fi}}%
3003 % == maparabic ==
3004 % Native digits, if provided in ini (TeX level, xe and lua)
3005 \ifcase\bbl@engine\else
3006 \bbl@ifunset{\bbl@dgnat@\languagename}{}%
3007 {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
3008 \expandafter\expandafter\expandafter
3009 \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
3010 \ifx\bbl@KVP@maparabic\@nil\else
3011 \ifx\bbl@latinarabic\@undefined
3012 \expandafter\let\expandafter\@arabic
3013 \csname bbl@counter@\languagename\endcsname

```



```

3014         \else      % ie, if layout=counters, which redefines \@arabic
3015             \expandafter\let\expandafter\bbl@latinarabic
3016             \csname bbl@counter@\language\endcsname
3017         \fi
3018     \fi
3019 \fi}%
3020 \fi
3021 % == mapdigits ==
3022 % Native digits (lua level).
3023 \ifodd\bbl@engine
3024     \ifx\bbl@KVP@mapdigits\@nil\else
3025         \bbl@ifunset{\bbl@dgnat@\language\endcsname}{}%
3026         {\RequirePackage{luatexbase}%
3027         \bbl@activate@preotf
3028         \directlua{
3029             Babel = Babel or {}  %% -> presets in luababel
3030             Babel.digits_mapped = true
3031             Babel.digits = Babel.digits or {}
3032             Babel.digits[\the\localeid] =
3033             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3034             if not Babel.numbers then
3035                 function Babel.numbers(head)
3036                     local LOCALE = luatexbase.registernumber'\bbl@attr@locale'
3037                     local GLYPH = node.id'glyph'
3038                     local inmath = false
3039                     for item in node.traverse(head) do
3040                         if not inmath and item.id == GLYPH then
3041                             local temp = node.get_attribute(item, LOCALE)
3042                             if Babel.digits[temp] then
3043                                 local chr = item.char
3044                                 if chr > 47 and chr < 58 then
3045                                     item.char = Babel.digits[temp][chr-47]
3046                                 end
3047                             end
3048                             elseif item.id == node.id'math' then
3049                                 inmath = (item.subtype == 0)
3050                             end
3051                         end
3052                     end
3053                     return head
3054                 end
3055             }%
3056         \fi
3057     \fi
3058 % == alph, Alph ==
3059 % What if extras<lang> contains a \babel@save\@alph? It won't be
3060 % restored correctly when exiting the language, so we ignore
3061 % this change with the \bbl@alph@saved trick.
3062 \ifx\bbl@KVP@alph\@nil\else
3063     \toks@\expandafter\expandafter\expandafter{%
3064     \csname extras\language\endcsname}%
3065     \bbl@exp{%
3066         \def\<extras\language>{%
3067             \let\\bbl@alph@saved\\@alph
3068             \the\toks@
3069             \let\\@alph\\bbl@alph@saved
3070             \\babel@save\\@alph
3071             \let\\@alph<bbl@cntr@\bbl@KVP@alph @\language>}}%
3072 \fi

```

```

3073 \ifx\bb1@KVP@Alph\@nil\else
3074 \toks@{\expandafter\expandafter\expandafter{%
3075 \csname extras\language\endcsname}%
3076 \bb1@exp{%
3077 \def\<extras\language>{%
3078 \let\bb1@Alph@saved\bb1@Alph
3079 \the\toks@
3080 \let\bb1@Alph\bb1@Alph@saved
3081 \bb1@babel@save\bb1@Alph
3082 \let\bb1@Alph\bb1@cntr@bb1@KVP@Alph @\language>}}%
3083 \fi
3084 % == require.babel in ini ==
3085 % To load or reload the babel-*.tex, if require.babel in ini
3086 \bb1@ifunset{bb1@rtex@\language}{}%
3087 {\expandafter\ifx\csname bb1@rtex@\language\endcsname\@empty\else
3088 \let\BabelBeforeIni@gobbletwo
3089 \chardef\atcatcode=\catcode\@
3090 \catcode\@=11\relax
3091 \bb1@input@ini{\bb1@cs{rtex@\language}}%
3092 \catcode\@=\atcatcode
3093 \let\atcatcode\relax
3094 \fi}%
3095 % == main ==
3096 \ifx\bb1@KVP@main\@nil % Restore only if not 'main'
3097 \let\language\bb1@savelangname
3098 \chardef\localeid\bb1@savelocaleid\relax
3099 \fi}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T_EX. Non-digits characters are kept. The first macro is the generic “localized” command.

```

3100 % TODO. Merge with \localnumeral:
3101 \newcommand\localedigits{\@nameuse{\language digits}}
3102 \def\bb1@setdigits#1#2#3#4#5{%
3103 \bb1@exp{%
3104 \def\<\language digits>####1{% ie, \langdigits
3105 \<bb1@digits@\language>####1\@nil}%
3106 \let\bb1@cntr@digits@\language\<\language digits>%
3107 \def\<\language counter>####1{% ie, \langcounter
3108 \expandafter\bb1@counter@\language>%
3109 \csname c@####1\endcsname}%
3110 \def\<bb1@counter@\language>####1{% ie, \bb1@counter@lang
3111 \expandafter\bb1@digits@\language>%
3112 \number####1\@nil}}%
3113 \def\bb1@tempa##1##2##3##4##5{%
3114 \bb1@exp{% Wow, quite a lot of hashes! :-(
3115 \def\<bb1@digits@\language>#####1{%
3116 \ifx#####1\@nil % ie, \bb1@digits@lang
3117 \else
3118 \ifx0#####1#1%
3119 \else\ifx1#####1#2%
3120 \else\ifx2#####1#3%
3121 \else\ifx3#####1#4%
3122 \else\ifx4#####1#5%
3123 \else\ifx5#####1#1%
3124 \else\ifx6#####1#2%
3125 \else\ifx7#####1#3%
3126 \else\ifx8#####1#4%
3127 \else\ifx9#####1#5%

```



```

3185 \bbl@savedate
3186 \EndBabelCommands
3187 \fi
3188 % == hyphenrules ==
3189 \bbl@provide@hyphens{#1}%
3190 % Load the basic parameters (ids, typography, counters, and a few
3191 % more), while captions and dates are left out. But it may happen some
3192 % data has been loaded before automatically, so we first discard the
3193 % saved values.
3194 \def\bbl@load@basic#1{%
3195 \bbl@ifunset{\bbl@inidata@\language}\relax
3196 {\getlocaleproperty\bbl@tempa{\language}{identification/load.level}%
3197 \ifcase\bbl@tempa\else
3198 \bbl@csarg\let{lname@\language}\relax
3199 \fi}%
3200 \bbl@ifunset{\bbl@lname@#1}%
3201 {\def\BabelBeforeIni##1##2{%
3202 \begingroup
3203 \catcode\`[=12 \catcode\`=12 \catcode\`=12
3204 \catcode\`=12 \catcode\`=12 %
3205 \let\bbl@ini@captions@aux\@gobbletwo
3206 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3207 \bbl@read@ini{##1}0%
3208 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3209 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3210 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3211 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3212 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3213 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3214 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3215 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3216 \bbl@exportkey{chrng}{characters.ranges}{}%
3217 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3218 \ifx\bbl@initoload\relax\endinput\fi
3219 \endgroup}%
3220 \begingroup % boxed, to avoid extra spaces:
3221 \ifx\bbl@initoload\relax
3222 \bbl@input@ini{#1}%
3223 \else
3224 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3225 \fi
3226 \endgroup}%
3227 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3228 \def\bbl@provide@hyphens#1{%
3229 \let\bbl@tempa\relax
3230 \ifx\bbl@KVP@hyphenrules\@nil\else
3231 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3232 \bbl@foreach\bbl@KVP@hyphenrules{%
3233 \ifx\bbl@tempa\relax % if not yet found
3234 \bbl@ifsamestring{##1}{+}%
3235 {\bbl@exp{\addlanguage\<#1>}}}%
3236 {}%
3237 \bbl@ifunset{l@#1}%
3238 {}%
3239 {\bbl@exp{\let\bbl@tempa\<#1>}}}%
3240 \fi}%
3241 \fi

```

```

3242 \ifx\bbbl@tempa\relax %           if no opt or no language in opt found
3243 \ifx\bbbl@KVP@import\@nil
3244 \ifx\bbbl@initoload\relax\else
3245 \bbbl@exp{%                        and hyphenrules is not empty
3246 \\\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3247 }%
3248 {\let\\bbbl@tempa<l@\bbbl@cl{hyphr}>}}%
3249 \fi
3250 \else % if importing
3251 \bbbl@exp{%                        and hyphenrules is not empty
3252 \\\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3253 }%
3254 {\let\\bbbl@tempa<l@\bbbl@cl{hyphr}>}}%
3255 \fi
3256 \fi
3257 \bbbl@ifunset{\bbbl@tempa}%        ie, relax or undefined
3258 {\bbbl@ifunset{l@#1}%              no hyphenrules found - fallback
3259 {\bbbl@exp{\\adddialect<l@#1>\language}}%
3260 }%                                  so, l@<lang> is ok - nothing to do
3261 {\bbbl@exp{\\adddialect<l@#1>\bbbl@tempa}}% found in opt list or ini
3262

```

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.

```

3263 \ifx\bbbl@readstream\@undefined
3264 \csname newread\endcsname\bbbl@readstream
3265 \fi
3266 \def\bbbl@input@ini#1{%
3267 \bbbl@exp{\lowercase{\InputIfFileExists{babel-#1.tex}}{}}}%
3268 \def\bbbl@inipreread#1=#2\@{%
3269 \bbbl@trim@def\bbbl@tempa{#1}% Redundant below !!
3270 \bbbl@trim\toks@{#2}%
3271 % Move trims here ??
3272 \bbbl@ifunset{\bbbl@KVP@\bbbl@section/\bbbl@tempa}%
3273 {\bbbl@exp{%
3274 \\\g@addto@macro\\bbbl@inidata{%
3275 \\\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3276 \expandafter\bbbl@inireader\bbbl@tempa=#2\@}%
3277 }%
3278 \def\bbbl@read@ini#1#2{%
3279 \bbbl@csarg\edef{lini@language}{#1}%
3280 \openin\bbbl@readstream=babel-#1.ini
3281 \ifeof\bbbl@readstream
3282 \bbbl@error
3283 {There is no ini file for the requested language\\%
3284 (#1). Perhaps you misspelled it or your installation\\%
3285 is not complete.}%
3286 {Fix the name or reinstall babel.}%
3287 \else
3288 \bbbl@exp{\def\\bbbl@inidata{%
3289 \\\bbbl@elt{identification}{tag.ini}{#1}%
3290 \\\bbbl@elt{identification}{load.level}{#2}}}%
3291 \let\bbbl@section\@empty
3292 \let\bbbl@savestrings\@empty
3293 \let\bbbl@savetoday\@empty
3294 \let\bbbl@savestate\@empty
3295 \let\bbbl@inireader\bbbl@iniskip
3296 \bbbl@info{Importing
3297 \ifcase#2 \or font and identification \or basic \fi

```

```

3298             data for \language\%
3299             from babel-#1.ini. Reported}%
3300     \loop
3301     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3302     \endlinechar\m@ne
3303     \read\bbl@readstream to \bbl@line
3304     \endlinechar`\^^M
3305     \ifx\bbl@line\@empty\else
3306     \expandafter\bbl@inline\bbl@line\bbl@inline
3307     \fi
3308     \repeat
3309     \bbl@foreach\bbl@renewlist{%
3310     \bbl@ifunset\bbl@renew@##1\{\}\bbl@inisec[##1]\@}%
3311     \global\let\bbl@renewlist\empty
3312     % Ends last section. See \bbl@inisec
3313     \def\bbl@elt##1##2{\bbl@inireader##1=##2\@}%
3314     \bbl@cs{renew@\bbl@section}%
3315     \global\bbl@csarg\let{renew@\bbl@section}\relax
3316     \bbl@cs{secpost@\bbl@section}%
3317     \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3318     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
3319     \bbl@tglobal\bbl@ini@loaded
3320     \fi}
3321 \def\bbl@inline#1\bbl@inline{%
3322 \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

3323 \def\bbl@iniskip#1\@{%          if starts with ;
3324 \def\bbl@inisec[#1]#2\@{%      if starts with opening bracket
3325 \def\bbl@elt##1##2{%
3326     \expandafter\toks@\expandafter{%
3327     \expandafter{\bbl@section}{##1}{##2}}%
3328     \bbl@exp{%
3329     \g@addto@macro\bbl@inidata{\bbl@elt\the\toks@}%
3330     \bbl@inireader##1=##2\@}%
3331     \bbl@cs{renew@\bbl@section}%
3332     \global\bbl@csarg\let{renew@\bbl@section}\relax
3333     \bbl@cs{secpost@\bbl@section}%
3334     % The previous code belongs to the previous section.
3335     % -----
3336     % Now start the current one.
3337     \in@{=date.}{#1}%
3338     \ifin@
3339     \lowercase{\def\bbl@tempa{=##1}}%
3340     \bbl@replace\bbl@tempa{=date.gregorian}{}%
3341     \bbl@replace\bbl@tempa{=date.}{}%
3342     \in@{.licr=}{#1}%
3343     \ifin@
3344     \ifcase\bbl@engine
3345     \bbl@replace\bbl@tempa{.licr=}{}%
3346     \else
3347     \let\bbl@tempa\relax
3348     \fi
3349     \fi
3350     \ifx\bbl@tempa\relax\else

```

```

3351 \bbl@replace\bbl@tempa{=}{}%
3352 \bbl@exp{%
3353 \def<\bbl@inikv@#1>####1=####2\\@@{%
3354 \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3355 \fi
3356 \fi
3357 \def\bbl@section{#1}%
3358 \def\bbl@elt##1##2{%
3359 \@namedef{\bbl@KVP@#1/##1}{}}%
3360 \bbl@cs{renew@#1}%
3361 \bbl@cs{secpre@#1}% pre-section `hook'
3362 \bbl@ifunset{\bbl@inikv@#1}%
3363 {\let\bbl@inireader\bbl@iniskip}%
3364 {\bbl@exp{\let\\bbl@inireader<\bbl@inikv@#1>}}
3365 \let\bbl@renewlist@empty
3366 \def\bbl@renewinikey#1/#2\\@@#3{%
3367 \bbl@ifunset{\bbl@renew@#1}%
3368 {\bbl@add@list\bbl@renewlist{#1}}%
3369 {}}%
3370 \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}

```

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

```

3371 \def\bbl@inikv#1=#2\\@@{% key=value
3372 \bbl@trim@def\bbl@tempa{#1}%
3373 \bbl@trim\toks@{#2}%
3374 \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.

```

3375 \def\bbl@exportkey#1#2#3{%
3376 \bbl@ifunset{\bbl@kv@#2}%
3377 {\bbl@csarg\gdef{#1@\language}\{#3}}%
3378 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\empty
3379 \bbl@csarg\gdef{#1@\language}\{#3}}%
3380 \else
3381 \bbl@exp{\global\let<\bbl@#1@\language>\<\bbl@kv@#2>}%
3382 \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@secpost@identification is called always (via \bbl@inise), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3383 \def\bbl@iniwarning#1{%
3384 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3385 {\bbl@warning{%
3386 From babel-\bbl@cs{lini@\language}.ini:\\%
3387 \bbl@cs{@kv@identification.warning#1}\\%
3388 Reported }}}
3389 \let\bbl@inikv@identification\bbl@inikv
3390 \def\bbl@secpost@identification{%
3391 \bbl@iniwarning}%
3392 \ifcase\bbl@engine
3393 \bbl@iniwarning{.pdf\latex}%
3394 \or
3395 \bbl@iniwarning{.lua\latex}%
3396 \or
3397 \bbl@iniwarning{.x\latex}%
3398 \fi%
3399 \bbl@exportkey{elname}{identification.name.english}{}}%

```

```

3400 \bbl@exp{\bbbl@exportkey{lname}{identification.name.opentype}%
3401   {\csname bbl@elname@language\endcsname}}%
3402 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{%} TODO
3403 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3404 \bbl@exportkey{esname}{identification.script.name}{%}
3405 \bbl@exp{\bbbl@exportkey{sname}{identification.script.name.opentype}%
3406   {\csname bbl@esname@language\endcsname}}%
3407 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{%}
3408 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3409 \ifbbl@bcptoname
3410   \bbl@csarg\xdef{bcp@map@bbl@cl{lbcpr}}{\language}%
3411 \fi}
3412 \let\bbl@inikv@typography\bbl@inikv
3413 \let\bbl@inikv@characters\bbl@inikv
3414 \let\bbl@inikv@numbers\bbl@inikv
3415 \def\bbl@inikv@counters#1=#2\@@{%
3416   \bbl@ifsamestring{#1}{digits}%
3417   {\bbl@error{The counter name 'digits' is reserved for mapping\%
3418     decimal digits}%
3419     {Use another name.}}%
3420   }%
3421 \def\bbl@tempc{#1}%
3422 \bbl@trim@def{\bbl@tempb*}{#2}%
3423 \in@{.1$}{#1$}%
3424 \ifin@
3425   \bbl@replace\bbl@tempc{.1}{%}
3426   \bbl@csarg\protected@xdef{cntr@bbl@tempc @language}{%
3427     \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3428 \fi
3429 \in@{.F.}{#1}%
3430 \ifin@else\in@{.S.}{#1}\fi
3431 \ifin@
3432   \bbl@csarg\protected@xdef{cntr@#1@language}{\bbl@tempb*}%
3433 \else
3434   \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3435   \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3436   \bbl@csarg{\global\expandafter\let}{cntr@#1@language}\bbl@tempa
3437 \fi}
3438 \def\bbl@after@ini{%
3439   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3440   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3441   \bbl@exportkey{prehc}{typography.prehyphenchar}{%}
3442   \bbl@exportkey{lncbrk}{typography.linebreaking}{h}%
3443   \bbl@exportkey{hyphr}{typography.hyphenrules}{%}
3444   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{%}
3445   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{%}
3446   \bbl@exportkey{intsp}{typography.intraspace}{%}
3447   \bbl@exportkey{jstfy}{typography.justify}{w}%
3448   \bbl@exportkey{chrng}{characters.ranges}{%}
3449   \bbl@exportkey{dgnat}{numbers.digits.native}{%}
3450   \bbl@exportkey{rqtex}{identification.require.babel}{%}
3451   \bbl@toggle\bbl@savetoday
3452   \bbl@toggle\bbl@savetoday

```

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.

```

3453 \ifcase\bbl@engine
3454   \bbl@csarg\def{inikv@captions.licr}{#1=#2\@@{%

```



```

3455 \bbl@ini@captions@aux{#1}{#2}}
3456 \else
3457 \def\bbl@inikv@captions#1=#2\@@{%
3458 \bbl@ini@captions@aux{#1}{#2}}
3459 \fi

```

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

```

3460 \def\bbl@ini@captions@aux#1#2{%
3461 \bbl@trim@def\bbl@tempa{#1}%
3462 \bbl@ifblank{#2}%
3463 {\bbl@exp{%
3464 \toks@{\bbl@nocaption{\bbl@tempa}{\language\language\bbl@tempa name}}}%
3465 {\bbl@trim\toks@{#2}}}%
3466 \bbl@exp{%
3467 \bbl@add\bbl@savestrings{%
3468 \SetString\<\bbl@tempa name>{\the\toks@}}}%

```

TODO. Document

```

3469 % Arguments are _not_ protected.
3470 \let\bbl@calendar\@empty
3471 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3472 \def\bbl@cased{% TODO. Move
3473 \ifx\oe\OE
3474 \expandafter\in@\expandafter
3475 {\expandafter\OE\expandafter}\expandafter{\oe}%
3476 \ifin@
3477 \bbl@afterelse\expandafter\MakeUppercase
3478 \else
3479 \bbl@afterfi\expandafter\MakeLowercase
3480 \fi
3481 \else
3482 \expandafter\@firstofone
3483 \fi}
3484 \def\bbl@localedate#1#2#3#4{%
3485 \begingroup
3486 \ifx\@empty#1\@empty\else
3487 \let\bbl@ld@calendar\@empty
3488 \let\bbl@ld@variant\@empty
3489 \edef\bbl@tempa{\zap@space#1 \@empty}%
3490 \def\bbl@tempb##1=##2\@@{\@namedef{\bbl@ld##1}{##2}}%
3491 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3492 \edef\bbl@calendar{%
3493 \bbl@ld@calendar
3494 \ifx\bbl@ld@variant\@empty\else
3495 .\bbl@ld@variant
3496 \fi}%
3497 \bbl@replace\bbl@calendar{gregorian}{}}%
3498 \fi
3499 \bbl@cased
3500 {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3501 \endgroup}
3502 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3503 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3504 \bbl@trim@def\bbl@tempa{#1.#2}%
3505 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3506 {\bbl@trim@def\bbl@tempa{#3}%
3507 \bbl@trim\toks@{#5}%
3508 \@temptokena\expandafter{\bbl@savestate}%
3509 \bbl@exp{% Reverse order - in ini last wins

```

```

3510 \def\\bbl@savestate{%
3511   \\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}%
3512   \the\temptokena}}}%
3513 {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3514   {\lowercase{\def\bbl@tempb{#6}}}%
3515   \bbl@trim@def\bbl@toreplace{#5}%
3516   \bbl@TG@date
3517   \bbl@ifunset{\bbl@date@\language @}%
3518   {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3519   % TODO. Move to a better place.
3520   \bbl@exp{%
3521     \gdef\<\language date>\protect\<\language date >}%
3522     \gdef\<\language date >####1####2####3{%
3523       \\bbl@usedategrouptue
3524       \<bbl@ensure@\language >{%
3525         \\localedate{####1}{####2}{####3}}}%
3526       \\bbl@add\\bbl@savetoday{%
3527         \\SetString\\today{%
3528           \<\language date>%
3529           {\the\year}{\the\month}{\the\day}}}%
3530       }%
3531   \ifx\bbl@tempb\@empty\else
3532     \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3533   \fi}%
3534   {}}

```

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.

```

3535 \let\bbl@calendar\@empty
3536 \newcommand\BabelDateSpace{\nobreakspace}
3537 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3538 \newcommand\BabelDated[1]{\number#1}
3539 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3540 \newcommand\BabelDateM[1]{\number#1}
3541 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3542 \newcommand\BabelDateMMM[1]{%
3543   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3544 \newcommand\BabelDatey[1]{\number#1}%
3545 \newcommand\BabelDateyy[1]{%
3546   \ifnum#1<10 0\number#1 %
3547   \else\ifnum#1<100 \number#1 %
3548   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3549   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3550   \else
3551     \bbl@error
3552     {Currently two-digit years are restricted to the\
3553     range 0-9999.}%
3554     {There is little you can do. Sorry.}%
3555   \fi\fi\fi\fi}}
3556 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3557 \def\bbl@replace@finish@iii#1{%
3558   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3559 \def\bbl@TG@date{%
3560   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3561   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3562   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3563   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3564   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%

```

```

3565 \bbl@replace\bbl@toreplace{[MM]}\BabelDateMM{####2}%
3566 \bbl@replace\bbl@toreplace{[MMMM]}\BabelDateMMMM{####2}%
3567 \bbl@replace\bbl@toreplace{[y]}\BabelDatey{####1}%
3568 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{####1}%
3569 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}%
3570 \bbl@replace\bbl@toreplace{[y]}\bbl@datecctr[####1]|}%
3571 \bbl@replace\bbl@toreplace{[m]}\bbl@datecctr[####2]|}%
3572 \bbl@replace\bbl@toreplace{[d]}\bbl@datecctr[####3]|}%
3573 % Note after \bbl@replace \toks@ contains the resulting string.
3574 % TODO - Using this implicit behavior doesn't seem a good idea.
3575 \bbl@replace@finish@iii\bbl@toreplace}
3576 \def\bbl@datecctr\expandafter\bbl@xdatecctr\expandafter}
3577 \def\bbl@xdatecctr[#1|#2]\localenumeral{#2}{#1}}

```

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

```

3578 \def\bbl@provide@lsys#1{%
3579   \bbl@ifunset{bbl@lname@#1}%
3580     {\bbl@ini@basic{#1}}%
3581   }%
3582   \bbl@csarg\let{lsys@#1}\empty
3583   \bbl@ifunset{bbl@sname@#1}\bbl@csarg\gdef{sname@#1}{Default}}}%
3584   \bbl@ifunset{bbl@sotf@#1}\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
3585   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}}%
3586   \bbl@ifunset{bbl@lname@#1}}}%
3587     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3588   \ifcase\bbl@engine\or\or
3589     \bbl@ifunset{bbl@prehc@#1}}}%
3590     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}}%
3591     }%
3592     {\ifx\bbl@xenohyph\undefined
3593       \let\bbl@xenohyph\bbl@xenohyph@d
3594       \ifx\AtBeginDocument\@notprerr
3595         \expandafter\@secondoftwo % to execute right now
3596       \fi
3597       \AtBeginDocument{%
3598         \expandafter\bbl@add
3599         \csname selectfont \endcsname{\bbl@xenohyph}%
3600         \expandafter\selectlanguage\expandafter{\languagename}%
3601         \expandafter\bbl@tglobal\csname selectfont \endcsname}%
3602       \fi}}%
3603   \fi
3604   \bbl@csarg\bbl@tglobal{lsys@#1}}
3605 \def\bbl@ifset#1#2#3{% TODO. Move to the correct place.
3606   \bbl@ifunset{#1}{#3}\bbl@exp{\bbl@ifblank{#1}}{#3}{#2}}
3607 \def\bbl@xenohyph@d{%
3608   \bbl@ifset{bbl@prehc@languagename}%
3609     {\ifnum\hyphenchar\font=\defaultshyphenchar
3610       \iffontchar\font\bbl@cl{prehc}\relax
3611       \hyphenchar\font\bbl@cl{prehc}\relax
3612     \else\iffontchar\font"200B
3613       \hyphenchar\font"200B
3614     \else
3615       \bbl@error
3616       {Neither 0 nor ZERO WIDTH SPACE are available\\%
3617        in the current font, and therefore the hyphen\\%
3618        will be printed. Try with 'HyphenChar', but be\\%
3619        aware this setting is not safe (see the manual).}%
3620       {See the manual.}%

```

```

3621      \hyphenchar\font\defaultshyphenchar
3622      \fi\fi
3623      \fi}%
3624      {\hyphenchar\font\defaultshyphenchar}}
3625      % \fi}

```

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

```

3626 \def\bbl@ini@basic#1{%
3627   \def\BabelBeforeIni##1##2{%
3628     \begingroup
3629     \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
3630     \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3631     \catcode`\;=12 \catcode`\|=12 %
3632     \bbl@read@ini{##1}1%
3633     \endinput          % babel- .tex may contain onlypreamble's
3634     \endgroup}%        boxed, to avoid extra spaces:
3635     {\bbl@input@ini{##1}}

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3636 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3637   \ifx\\#1%           % \\ before, in case #1 is multiletter
3638     \bbl@exp{%
3639       \def\\bbl@tempa####1{%
3640         \ifcase>####1\space\the\toks@<\else>\\@ctrerr<\fi>}}%
3641       \else
3642         \toks@<\expandafter\the\toks@<\or #1>%
3643         \expandafter\bbl@buildifcase
3644       \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as a special case, for a fixed form (see babel-he.ini, for example).

```

3645 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3646 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3647 \newcommand\localecounter[2]{%
3648   \expandafter\bbl@localecntr
3649   \expandafter{\number\csname c@#2\endcsname}{#1}}
3650 \def\bbl@alphnumeral#1#2{%
3651   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3652 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3653   \ifcase\car#8\@nil\or % Currenty <10000, but prepared for bigger
3654     \bbl@alphnumeral@ii{#9}000000#1\or
3655     \bbl@alphnumeral@ii{#9}00000#1#2\or
3656     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3657     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3658     \bbl@alphnum@invalid{>9999}%
3659   \fi}
3660 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3661   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3662   {\bbl@cs{cntr@#1.4@\languagename}{#5}
3663     \bbl@cs{cntr@#1.3@\languagename}{#6}

```

```

3664 \bbl@cs{cntr@#1.2@\language}#7%
3665 \bbl@cs{cntr@#1.1@\language}#8%
3666 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3667 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3668 {\bbl@cs{cntr@#1.S.321@\language}}%
3669 \fi}%
3670 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}
3671 \def\bbl@alphnum@invalid#1{%
3672 \bbl@error{Alphabetic numeral too large (#1)}%
3673 {Currently this is the limit.}}

```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

3674 \newcommand\localeinfo[1]{%
3675 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
3676 {\bbl@error{I've found no info for the current locale.\%
3677 The corresponding ini file has not been loaded\%
3678 Perhaps it doesn't exist}%
3679 {See the manual for details.}}%
3680 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}
3681 % \namedef{bbl@info@name.locale}{lname}
3682 \namedef{bbl@info@tag.ini}{lini}
3683 \namedef{bbl@info@name.english}{elname}
3684 \namedef{bbl@info@name.opentype}{lname}
3685 \namedef{bbl@info@tag.bcp47}{lbc} % TODO
3686 \namedef{bbl@info@tag.opentype}{lotf}
3687 \namedef{bbl@info@script.name}{esname}
3688 \namedef{bbl@info@script.name.opentype}{sname}
3689 \namedef{bbl@info@script.tag.bcp47}{sbcp}
3690 \namedef{bbl@info@script.tag.opentype}{sotf}
3691 \let\bbl@ensureinfo\@gobble
3692 \newcommand\BabelEnsureInfo{%
3693 \ifx\InputIfFileExists\undefined\else
3694 \def\bbl@ensureinfo##1{%
3695 \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}}%
3696 \fi
3697 \bbl@foreach\bbl@loaded{%
3698 \def\language{##1}%
3699 \bbl@ensureinfo{##1}}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3700 \newcommand\getlocaleproperty[3]{%
3701 \let#1\relax
3702 \def\bbl@elt##1##2##3{%
3703 \bbl@ifsamestring{##1/##2}{##3}%
3704 {\providecommand#1{##3}%
3705 \def\bbl@elt####1####2####3{}}%
3706 {}}%
3707 \bbl@cs{inidata@#2}%
3708 \ifx#1\relax
3709 \bbl@error
3710 {Unknown key for locale '#2':\%
3711 #3\%
3712 \string#1 will be set to \relax}%
3713 {Perhaps you misspelled it.}%
3714 \fi}
3715 \let\bbl@ini@loaded\@empty

```

```
3716 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
```

10 Adjusting the Babel bahavior

A generic high level inteface is provided to adjust some global and general settings.

```
3717 \newcommand\babeladjust[1]{% TODO. Error handling.
3718   \bbl@forkv{#1}{%
3719     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3720     {\bbl@cs{ADJ@##1}{##2}}%
3721     {\bbl@cs{ADJ@##1@##2}}}
3722 %
3723 \def\bbl@adjust@lua#1#2{%
3724   \ifvmode
3725     \ifnum\currentgrouplevel=\z@
3726       \directlua{ Babel.#2 }%
3727       \expandafter\expandafter\expandafter\@gobble
3728     \fi
3729   \fi
3730   {\bbl@error   % The error is gobbled if everything went ok.
3731     {Currently, #1 related features can be adjusted only\\%
3732       in the main vertical list.}%
3733     {Maybe things change in the future, but this is what it is.}}}
3734 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3735   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3736 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3737   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3738 \@namedef{\bbl@ADJ@bidi.text@on}{%
3739   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3740 \@namedef{\bbl@ADJ@bidi.text@off}{%
3741   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3742 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3743   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3744 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3745   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3746 %
3747 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3748   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3749 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3750   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3751 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
3752   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3753 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
3754   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3755 %
3756 \def\bbl@adjust@layout#1{%
3757   \ifvmode
3758     #1%
3759     \expandafter\@gobble
3760   \fi
3761   {\bbl@error   % The error is gobbled if everything went ok.
3762     {Currently, layout related features can be adjusted only\\%
3763       in vertical mode.}%
3764     {Maybe things change in the future, but this is what it is.}}}
3765 \@namedef{\bbl@ADJ@layout.tabular@on}{%
3766   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3767 \@namedef{\bbl@ADJ@layout.tabular@off}{%
3768   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
```

```

3769 \@namedef{bbl@ADJ@layout.lists@on}{%
3770   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3771 \@namedef{bbl@ADJ@layout.lists@on}{%
3772   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3773 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3774   \bbl@activateposthyphen}
3775 %
3776 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3777   \bbl@bcpallowedtrue}
3778 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3779   \bbl@bcpallowedfalse}
3780 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3781   \def\bbl@bcp@prefix{#1}}
3782 \def\bbl@bcp@prefix{bcp47-}
3783 \@namedef{bbl@ADJ@autoload.options#1}{%
3784   \def\bbl@autoload@options{#1}}
3785 \let\bbl@autoload@bcptoptions\@empty
3786 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3787   \def\bbl@autoload@bcptoptions{#1}}
3788 \newif\ifbbl@bcptoname
3789 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3790   \bbl@bcptonametrue}
3791 \BabelEnsureInfo
3792 }
3793 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3794   \bbl@bcptonamefalse}
3795 % TODO: use babel name, override
3796 %
3797 % As the final task, load the code for lua.
3798 %
3799 \ifx\directlua\@undefined\else
3800   \ifx\bbl@luapatterns\@undefined
3801     \input luababel.def
3802   \fi
3803 \fi
3804 </core>

```

A proxy file for switch.def

```

3805 <*kernel>
3806 \let\bbl@onlyswitch\@empty
3807 \input babel.def
3808 \let\bbl@onlyswitch\@undefined
3809 </kernel>
3810 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by \LaTeX because it should instruct \TeX to read hyphenation patterns. To this end the `docstrip` option `patterns` can be used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

To make sure that \LaTeX 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```

3811 <<Make sure ProvidesFile is defined>>
3812 \ProvidesFile{hyphen.cfg}[\<<date>>] \<<version>> Babel hyphens]
3813 \xdef\bbbl@format{\jobname}
3814 \def\bbbl@version{\<<version>>}
3815 \def\bbbl@date{\<<date>>}
3816 \ifx\AtBeginDocument\@undefined
3817   \def\@empty{}
3818   \let\orig@dump\dump
3819   \def\dump{%
3820     \ifx\@ztryfc\@undefined
3821       \else
3822         \toks0=\expandafter{\@preamblecmds}%
3823         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3824         \def\@begindocumenthook{}%
3825       \fi
3826       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3827 \fi
3828 <<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.

```

3829 \def\process@line#1#2 #3 #4 {%
3830   \ifx=#1%
3831     \process@synonym{#2}%
3832   \else
3833     \process@language{#1#2}{#3}{#4}%
3834   \fi
3835   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an =. It needs an empty token register to begin with. `\bbbl@languages` is also set to empty.

```

3836 \toks@{}
3837 \def\bbbl@languages{}

```

When no languages have been loaded yet, the name following the = will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.)

Otherwise the name will be a synonym for the language loaded last.

We also need to copy the `hyphenmin` parameters for the synonym.

```

3838 \def\process@synonym#1{%
3839   \ifnum\last@language=\m@ne
3840     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3841   \else
3842     \expandafter\chardef\csname l@#1\endcsname\last@language
3843     \wlog{\string\l@#1=\string\language\the\last@language}%
3844     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
3845     \csname\language\endcsname hyphenmins\endcsname
3846     \let\bbbl@elt\relax
3847     \edef\bbbl@languages{\bbbl@languages\bbbl@elt{#1}{\the\last@language}}{}%
3848   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the

‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

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

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

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

3849 \def\process@language#1#2#3{%
3850   \expandafter\addlanguage\csname l@#1\endcsname
3851   \expandafter\language\csname l@#1\endcsname
3852   \edef\language#1}%
3853   \bbl@hook@everylanguage{#1}%
3854   % > luatex
3855   \bbl@get@enc#1::@@@
3856   \begingroup
3857     \lefthyphenmin\m@ne
3858     \bbl@hook@loadpatterns{#2}%
3859     % > luatex
3860     \ifnum\lefthyphenmin=\m@ne
3861     \else
3862       \expandafter\xdef\csname #1hyphenmins\endcsname{%
3863         \the\lefthyphenmin\the\righthyphenmin}%
3864       \fi
3865   \endgroup
3866   \def\bbl@tempa{#3}%
3867   \ifx\bbl@tempa\@empty\else
3868     \bbl@hook@loadexceptions{#3}%
3869     % > luatex
3870   \fi
3871   \let\bbl@elt\relax
3872   \edef\bbl@languages{%
3873     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3874   \ifnum\the\language=\z@
3875     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3876       \set@hyphenmins\tw@\thr@@\relax

```

```

3877 \else
3878 \expandafter\expandafter\expandafter\set@hyphenmins
3879 \csname #1hyphenmins\endcsname
3880 \fi
3881 \the\toks@
3882 \toks@{}%
3883 \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.

```

3884 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. loadkernel currently loads nothing, but define some basic macros instead.

```

3885 \def\bbl@hook@everylanguage#1{%
3886 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3887 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3888 \def\bbl@hook@loadkernel#1{%
3889 \def\addlanguage{\csname newlanguage\endcsname}%
3890 \def\adddialect##1##2{%
3891 \global\chardef##1##2\relax
3892 \wlog{\string##1 = a dialect from \string\language##2}}%
3893 \def\iflanguage##1{%
3894 \expandafter\ifx\csname l@##1\endcsname\relax
3895 \@nolanerr{##1}%
3896 \else
3897 \ifnum\csname l@##1\endcsname=\language
3898 \expandafter\expandafter\expandafter\@firstoftwo
3899 \else
3900 \expandafter\expandafter\expandafter\@secondoftwo
3901 \fi
3902 \fi}%
3903 \def\providehyphenmins##1##2{%
3904 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
3905 \@namedef{##1hyphenmins}{##2}%
3906 \fi}%
3907 \def\set@hyphenmins##1##2{%
3908 \lefthyphenmin##1\relax
3909 \righthyphenmin##2\relax}%
3910 \def\selectlanguage{%
3911 \errhelp{Selecting a language requires a package supporting it}%
3912 \errmessage{Not loaded}}%
3913 \let\foreignlanguage\selectlanguage
3914 \let\otherlanguage\selectlanguage
3915 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
3916 \def\bbl@usehooks##1##2{% TODO. Temporary!!
3917 \def\setlocale{%
3918 \errhelp{Find an armchair, sit down and wait}%
3919 \errmessage{Not yet available}}%
3920 \let\uselocale\setlocale
3921 \let\locale\setlocale
3922 \let\selectlocale\setlocale
3923 \let\localename\setlocale
3924 \let\textlocale\setlocale
3925 \let\textlanguage\setlocale
3926 \let\languagetext\setlocale}
3927 \begingroup

```

```

3928 \def\AddBabelHook#1#2{%
3929   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3930     \def\next{\toks1}%
3931   \else
3932     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
3933   \fi
3934   \next}
3935 \ifx\directlua\undefined
3936   \ifx\XeTeXinputencoding\undefined\else
3937     \input xebabel.def
3938   \fi
3939 \else
3940   \input luababel.def
3941 \fi
3942 \openin1 = babel-\bbl@format.cfg
3943 \ifeof1
3944 \else
3945   \input babel-\bbl@format.cfg\relax
3946 \fi
3947 \closein1
3948 \endgroup
3949 \bbl@hook@loadkernel{switch.def}

```

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

```

3950 \openin1 = language.dat

```

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

```

3951 \def\language{english}%
3952 \ifeof1
3953   \message{I couldn't find the file language.dat,\space
3954     I will try the file hyphen.tex}
3955   \input hyphen.tex\relax
3956   \chardef\l@english\z@
3957 \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`.

```

3958 \last@language\m@ne

```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```

3959 \loop
3960   \endlinechar\m@ne
3961   \read1 to \bbl@line
3962   \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.

```

3963   \if T\ifeof1F\fi T\relax
3964   \ifx\bbl@line\empty\else
3965     \edef\bbl@line{\bbl@line\space\space\space}%
3966     \expandafter\process@line\bbl@line\relax
3967   \fi
3968 \repeat

```

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

```
3969 \begingroup
3970   \def\bbl@elt#1#2#3#4{%
3971     \global\language=#2\relax
3972     \gdef\language{#1}%
3973     \def\bbl@elt##1##2##3##4{}}%
3974   \bbl@languages
3975 \endgroup
3976 \fi
3977 \closein1
```

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

```
3978 \if/\the\toks@/\else
3979   \errhelp{language.dat loads no language, only synonyms}
3980   \errmessage{Orphan language synonym}
3981 \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.

```
3982 \let\bbl@line\@undefined
3983 \let\process@line\@undefined
3984 \let\process@synonym\@undefined
3985 \let\process@language\@undefined
3986 \let\bbl@get@enc\@undefined
3987 \let\bbl@hyph@enc\@undefined
3988 \let\bbl@tempa\@undefined
3989 \let\bbl@hook@loadkernel\@undefined
3990 \let\bbl@hook@everylanguage\@undefined
3991 \let\bbl@hook@loadpatterns\@undefined
3992 \let\bbl@hook@loadexceptions\@undefined
3993 \patterns
```

Here the code for `iniTEX` ends.

12 Font handling with fontspec

Add the bidi handler just before `luaotfload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
3994 <<{*More package options}>> ≡
3995 \chardef\bbl@bidimode\z@
3996 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
3997 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
3998 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
3999 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4000 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4001 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4002 <</More package options>>
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\. .family` by the corresponding macro `\. .default`.

```
4003 <<{*Font selection}>> ≡
4004 \bbl@trace{Font handling with fontspec}
```

```

4005 \@onlypreamble\babelfont
4006 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4007   \bbl@foreach{#1}{\%
4008     \expandafter\ifx\csname date##1\endcsname\relax
4009     \IfFileExists{babel-##1.tex}%
4010       {\babelprovide{##1}}%
4011       }{}%
4012   \fi}%
4013 \edef\bbl@tempa{#1}%
4014 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4015 \ifx\fontspec\@undefined
4016   \usepackage{fontspec}%
4017 \fi
4018 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4019 \bbl@bblfont}
4020 \newcommand\bbl@bblfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
4021   \bbl@ifunset{\bbl@tempb family}%
4022   {\bbl@providedefam{\bbl@tempb}}%
4023   {\bbl@exp{%
4024     \\\bbl@sreplace\<\bbl@tempb family >%
4025     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4026   % For the default font, just in case:
4027   \bbl@ifunset{\bbl@lsys\<\languagenam>}{\bbl@provide@lsys{\languagenam}}{}%
4028   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4029   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4030   \bbl@exp{%
4031     \let\<bbl@\bbl@tempb dflt@\languagenam>\<bbl@\bbl@tempb dflt@>%
4032     \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagenam>%
4033     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4034   {\bbl@foreach\bbl@tempa{\% ie bbl@rmdflt@lang / *scrt
4035     \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}}}%

```

If the family in the previous command does not exist, it must be defined. Here is how:

```

4036 \def\bbl@providedefam#1{\%
4037   \bbl@exp{%
4038     \\\newcommand\<#1default>{}% Just define it
4039     \\\bbl@add@list\\\bbl@font@fams{#1}%
4040     \\\DeclareRobustCommand\<#1family>{}%
4041     \\\not@math@alphabet\<#1family>\relax
4042     \\\fontfamily\<#1default>\selectfont}%
4043     \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}%

```

The following macro is activated when the hook babel-fontspec is enabled. But before we define a macro for a warning, which sets a flag to avoid duplicate them.

```

4044 \def\bbl@nostdfont#1{\%
4045   \bbl@ifunset{\bbl@WFF@\f@family}%
4046   {\bbl@csarg\gdef{WFF@\f@family}}{}% Flag, to avoid dupl warns
4047   \bbl@infowarn{The current font is not a babel standard family:\%
4048     #1%
4049     \fontname\font\\%
4050     There is nothing intrinsically wrong with this warning, and\\%
4051     you can ignore it altogether if you do not need these\\%
4052     families. But if they are used in the document, you should be\\%
4053     aware 'babel' will no set Script and Language for them, so\\%
4054     you may consider defining a new family with \string\babelfont.\\%
4055     See the manual for further details about \string\babelfont.\\%
4056     Reported}}
4057   }{}%
4058 \gdef\bbl@switchfont{%

```

```

4059 \bbl@ifunset{\bbl@sys@\language}\bbl@provide@sys{\language}}{}%
4060 \bbl@exp{% eg Arabic -> arabic
4061 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4062 \bbl@foreach\bbl@font@fams{%
4063   \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4064   {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4065     {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4066       {}% 123=F - nothing!
4067       {\bbl@exp{% 3=T - from generic
4068         \global\let<\bbl@##1dflt@\language>%
4069         \<\bbl@##1dflt@>}}}%
4070       {\bbl@exp{% 2=T - from script
4071         \global\let<\bbl@##1dflt@\language>%
4072         \<\bbl@##1dflt@*\bbl@tempa>}}}%
4073       {}}% 1=T - language, already defined
4074 \def\bbl@tempa{\bbl@nostdfont{}}%
4075 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4076   \bbl@ifunset{\bbl@##1dflt@\language}%
4077   {\bbl@cs{famrst@##1}%
4078     \global\bbl@csarg\let{famrst@##1}\relax}%
4079   {\bbl@exp{% order is relevant
4080     \\bbl@add\\originalTeX{%
4081       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4082       \<##1default>\<##1family>{##1}}}%
4083     \\bbl@font@set<\bbl@##1dflt@\language>% the main part!
4084     \<##1default>\<##1family>}}}%
4085 \bbl@ifrestoring{ }\bbl@tempa}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

4086 \ifx\f@family\undefined\else % if latex
4087 \ifcase\bbl@engine % if pdftex
4088 \let\bbl@ckeckstdfonts\relax
4089 \else
4090 \def\bbl@ckeckstdfonts{%
4091   \begingroup
4092   \global\let\bbl@ckeckstdfonts\relax
4093   \let\bbl@tempa\@empty
4094   \bbl@foreach\bbl@font@fams{%
4095     \bbl@ifunset{\bbl@##1dflt@}%
4096     {\@nameuse{##1family}%
4097       \bbl@csarg\gdef{WFF@f@family}}}% Flag
4098     \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4099       \space\space\fontname\font\\}%
4100     \bbl@csarg\xdef{##1dflt@}{\f@family}%
4101     \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4102     {}}%
4103   \ifx\bbl@tempa\@empty\else
4104     \bbl@infowarn{The following font families will use the default\\%
4105       settings for all or some languages:\\%
4106       \bbl@tempa
4107       There is nothing intrinsically wrong with it, but\\%
4108       'babel' will no set Script and Language, which could\\%
4109       be relevant in some languages. If your document uses\\%
4110       these families, consider redefining them with \string\babelfont.\\%
4111       Reported}%
4112   \fi
4113 \endgroup}
4114 \fi

```

4115 \fi

Now the macros defining the font with fontspec.

When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.

```

4116 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4117   \bbl@xin@{<>}{#1}%
4118   \ifin@
4119     \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4120   \fi
4121   \bbl@exp{%
4122     \def\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4123     \bbl@ifsamestring{#2}{\f@family}{\#3\let\bbl@tempa\relax}{}}%
4124 %      TODO - next should be global?, but even local does its job. I'm
4125 %      still not sure -- must investigate:
4126 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4127   \let\bbl@tempa\bbl@mapselect
4128   \let\bbl@mapselect\relax
4129   \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4130   \let#4\@empty           %      Make sure \renewfontfamily is valid
4131   \bbl@exp{%
4132     \let\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4133     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4134     {\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4135     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4136     {\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4137     \bbl@renewfontfamily\#4%
4138     [\bbl@cs{lsys@\language\name},#2]{#3}% ie \bbl@exp{..}{#3}
4139   \begingroup
4140     #4%
4141     \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4142   \endgroup
4143   \let#4\bbl@temp@fam
4144   \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4145   \let\bbl@mapselect\bbl@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.

```

4146 \def\bbl@font@rst#1#2#3#4{%
4147   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```

4148 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4149 \newcommand\babelFSstore[2][{%
4150   \bbl@ifblank{#1}%
4151   {\bbl@csarg\def{sname@#2}{Latin}}%
4152   {\bbl@csarg\def{sname@#2}{#1}}%
4153   \bbl@provide@dirs{#2}%
4154   \bbl@csarg\ifnum{wdir@#2}>\z@
4155     \let\bbl@beforeforeign\leavevmode
4156     \EnableBabelHook{babel-bidi}%

```

```

4157 \fi
4158 \bbl@foreach{#2}{%
4159   \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4160   \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4161   \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4162 \def\bbl@FSstore#1#2#3#4{%
4163   \bbl@csarg\edef{#2default#1}{#3}%
4164   \expandafter\addto\csname extras#1\endcsname{%
4165     \let#4#3%
4166     \ifx#3\f@family
4167       \edef#3{\csname bbl@#2default#1\endcsname}%
4168       \fontfamily{#3}\selectfont
4169     \else
4170       \edef#3{\csname bbl@#2default#1\endcsname}%
4171     \fi}%
4172   \expandafter\addto\csname noextras#1\endcsname{%
4173     \ifx#3\f@family
4174       \fontfamily{#4}\selectfont
4175     \fi
4176     \let#3#4}}
4177 \let\bbl@langfeatures\@empty
4178 \def\babelFSfeatures{% make sure \fontspec is redefined once
4179   \let\bbl@ori@fontspec\fontspec
4180   \renewcommand\fontspec[1][{}]{%
4181     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4182   \let\babelFSfeatures\bbl@FSfeatures
4183   \babelFSfeatures}
4184 \def\bbl@FSfeatures#1#2{%
4185   \expandafter\addto\csname extras#1\endcsname{%
4186     \babel@save\bbl@langfeatures
4187     \edef\bbl@langfeatures{#2,}}
4188 <</Font selection>>

```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to `utf8`, which seems a sensible default.

```

4189 <<{*Footnote changes}>> ≡
4190 \bbl@trace{Bidi footnotes}
4191 \ifnum\bbl@bidimode>\z@
4192   \def\bbl@footnote#1#2#3{%
4193     \@ifnextchar[%
4194       {\bbl@footnote@o{#1}{#2}{#3}}%
4195       {\bbl@footnote@x{#1}{#2}{#3}}}
4196   \def\bbl@footnote@x#1#2#3#4{%
4197     \bgroup
4198     \select@language@x{\bbl@main@language}%
4199     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4200     \egroup}
4201   \def\bbl@footnote@o#1#2#3[#4]#5{%
4202     \bgroup
4203     \select@language@x{\bbl@main@language}%
4204     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4205     \egroup}
4206   \def\bbl@footnotetext#1#2#3{%

```



```

4207 \@ifnextchar[%
4208   {\bbl@footnotetext@o{#1}{#2}{#3}}%
4209   {\bbl@footnotetext@x{#1}{#2}{#3}}}%
4210 \def\bbl@footnotetext@x#1#2#3#4{%
4211   \bgroup
4212   \select@language@x{\bbl@main@language}%
4213   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4214   \egroup}
4215 \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4216   \bgroup
4217   \select@language@x{\bbl@main@language}%
4218   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4219   \egroup}
4220 \def\BabelFootnote#1#2#3#4{%
4221   \ifx\bbl@fn@footnote\undefined
4222     \let\bbl@fn@footnote\footnote
4223   \fi
4224   \ifx\bbl@fn@footnotetext\undefined
4225     \let\bbl@fn@footnotetext\footnotetext
4226   \fi
4227   \bbl@ifblank{#2}%
4228   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4229    \@namedef{\bbl@stripslash#1text}%
4230    {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4231   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4232    \@namedef{\bbl@stripslash#1text}%
4233    {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4234 \fi
4235 <</Footnote changes>>

```

Now, the code.

```

4236 (*xetex)
4237 \def\BabelStringsDefault{unicode}
4238 \let\xebbl@stop\relax
4239 \AddBabelHook{xetex}{encodedcommands}{%
4240   \def\bbl@tempa{#1}%
4241   \ifx\bbl@tempa\empty
4242     \XeTeXinputencoding"bytes"%
4243   \else
4244     \XeTeXinputencoding"#1"%
4245   \fi
4246   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4247 \AddBabelHook{xetex}{stopcommands}{%
4248   \xebbl@stop
4249   \let\xebbl@stop\relax}
4250 \def\bbl@intraspace#1 #2 #3\@@{%
4251   \bbl@csarg\gdef{\xeisp@language}%
4252   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4253 \def\bbl@intrapenalty#1\@@{%
4254   \bbl@csarg\gdef{\xeipn@language}%
4255   {\XeTeXlinebreakpenalty #1\relax}}
4256 \def\bbl@provide@intraspace{%
4257   \bbl@xin@{\bbl@cl{\lnbrk}}{s}%
4258   \ifin@else\bbl@xin@{\bbl@cl{\lnbrk}}{c}\fi
4259   \ifin@
4260     \bbl@ifunset{\bbl@intsp@language}%
4261     {\expandafter\ifx\csname\bbl@intsp@language\endcsname\empty\else
4262       \ifx\bbl@KVP@intraspace\@nil
4263         \bbl@exp{%

```

```

4264      \\bbl@intraspace\bbl@cl{intsp}\\@}%
4265      \fi
4266      \ifx\bbl@KVP@intrapenalty\@nil
4267        \bbl@intrapenalty0\@@
4268      \fi
4269    \fi
4270    \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4271      \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4272    \fi
4273    \ifx\bbl@KVP@intrapenalty\@nil\else
4274      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4275    \fi
4276    \bbl@exp{%
4277      \\bbl@add\<extras\language>{%
4278        \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
4279        \<bbl@xeisp@\language>%
4280        \<bbl@xeipn@\language>}%
4281      \\bbl@toglobal\<extras\language>%
4282      \\bbl@add\<noextras\language>{%
4283        \XeTeXlinebreaklocale "en"%
4284        \\bbl@toglobal\<noextras\language>}%
4285      \ifx\bbl@ispace\@undefined
4286        \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4287        \ifx\AtBeginDocument\@notprerr
4288          \expandafter\@secondoftwo % to execute right now
4289        \fi
4290        \AtBeginDocument{%
4291          \expandafter\bbl@add
4292          \csname selectfont \endcsname{\bbl@ispace}%
4293          \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4294        \fi}%
4295    \fi}
4296 \ifx\DisableBabelHook\@undefined\endinput\fi
4297 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4298 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4299 \DisableBabelHook{babel-fontspec}
4300 <<Font selection>>
4301 \input txtbabel.def
4302 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T_EX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdfTeX and xetex.

```

4303 <*texxet>
4304 \providecommand\bbl@provide@intraspace{}
4305 \bbl@trace{Redefinitions for bidi layout}
4306 \def\bbl@sspre@caption{%
4307   \bbl@exp{\everyhbox{\bbl@texdir\bbl@cs{wdir@\bbl@main@language}}}}
4308 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4309 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}

```

```

4310 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4311 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4312   \def\@hangfrom#1{%
4313     \setbox\@tempboxa\hbox{{#1}}%
4314     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4315     \noindent\box\@tempboxa}
4316 \def\raggedright{%
4317   \let\\\@centercr
4318   \bbl@startskip\z@skip
4319   \@rightskip\@flushglue
4320   \bbl@endskip\@rightskip
4321   \parindent\z@
4322   \parfillskip\bbl@startskip}
4323 \def\raggedleft{%
4324   \let\\\@centercr
4325   \bbl@startskip\@flushglue
4326   \bbl@endskip\z@skip
4327   \parindent\z@
4328   \parfillskip\bbl@endskip}
4329 \fi
4330 \IfBabelLayout{lists}
4331   {\bbl@sreplace\list
4332     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4333     \def\bbl@listleftmargin{%
4334       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4335     \ifcase\bbl@engine
4336       \def\labelenumii{}\theenumii{}% pdfTeX doesn't reverse ()
4337       \def\p@enumiii{\p@enumii}\theenumii{}%
4338     \fi
4339     \bbl@sreplace\@verbatim
4340       {\leftskip\@totalleftmargin}%
4341       {\bbl@startskip\textwidth
4342         \advance\bbl@startskip-\linewidth}%
4343     \bbl@sreplace\@verbatim
4344       {\rightskip\z@skip}%
4345       {\bbl@endskip\z@skip}}%
4346   {}
4347 \IfBabelLayout{contents}
4348   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4349     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4350   {}
4351 \IfBabelLayout{columns}
4352   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4353     \def\bbl@outputbox#1{%
4354       \hb@xt@\textwidth{%
4355         \hskip\columnwidth
4356         \hfil
4357         {\normalcolor\vrule \@width\columnseprule}%
4358         \hfil
4359         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4360         \hskip-\textwidth
4361         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4362         \hskip\columnsep
4363         \hskip\columnwidth}}}%
4364   {}
4365 <<Footnote changes>>
4366 \IfBabelLayout{footnotes}%
4367   {\BabelFootnote\footnote\language\language}%
4368   {\BabelFootnote\localfootnote\language\language}%

```

```

4369 \BabelFootnote\mainfootnote{}{}{}
4370 {}

```

Implicitly reverses sectioning labels in bidi=basic, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

4371 \IfBabelLayout{counters}%
4372 {\let\bbl@latinarabic=@arabic
4373 \def@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4374 \let\bbl@asciroman=@roman
4375 \def@roman#1{\babelsublr{\ensureascii\bbl@asciroman#1}}%
4376 \let\bbl@asciiRoman=@Roman
4377 \def@Roman#1{\babelsublr{\ensureascii\bbl@asciiRoman#1}}}%
4378 </texet>

```

13.3 LuaTeX

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

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

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

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

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

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

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4379 <*\luatex>
4380 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4381 \bbl@trace{Read language.dat}
4382 \ifx\bbl@readstream\undefined
4383 \csname newread\endcsname\bbl@readstream

```

```

4384 \fi
4385 \beginngroup
4386 \toks@{}
4387 \count@ \z@ % 0=start, 1=0th, 2=normal
4388 \def\bbl@process@line#1#2 #3 #4 {%
4389 \ifx=#1%
4390 \bbl@process@synonym{#2}%
4391 \else
4392 \bbl@process@language{#1#2}{#3}{#4}%
4393 \fi
4394 \ignorespaces}
4395 \def\bbl@manylang{%
4396 \ifnum\bbl@last>\@ne
4397 \bbl@info{Non-standard hyphenation setup}%
4398 \fi
4399 \let\bbl@manylang\relax}
4400 \def\bbl@process@language#1#2#3{%
4401 \ifcase\count@
4402 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4403 \or
4404 \count@\tw@
4405 \fi
4406 \ifnum\count@=\tw@
4407 \expandafter\addlanguage\csname l@#1\endcsname
4408 \language\allocationnumber
4409 \chardef\bbl@last\allocationnumber
4410 \bbl@manylang
4411 \let\bbl@elt\relax
4412 \xdef\bbl@languages{%
4413 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4414 \fi
4415 \the\toks@
4416 \toks@{}}
4417 \def\bbl@process@synonym@aux#1#2{%
4418 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4419 \let\bbl@elt\relax
4420 \xdef\bbl@languages{%
4421 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4422 \def\bbl@process@synonym#1{%
4423 \ifcase\count@
4424 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4425 \or
4426 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4427 \else
4428 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4429 \fi}
4430 \ifx\bbl@languages@\undefined % Just a (sensible?) guess
4431 \chardef\l@english\z@
4432 \chardef\l@USenglish\z@
4433 \chardef\bbl@last\z@
4434 \global\namedef{bbl@hyphendata@0}{\hyphen.tex}}
4435 \gdef\bbl@languages{%
4436 \bbl@elt{english}{0}{\hyphen.tex}}%
4437 \bbl@elt{USenglish}{0}{}}
4438 \else
4439 \global\let\bbl@languages@format\bbl@languages
4440 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4441 \ifnum#2>\z@\else
4442 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%

```

```

4443     \fi}%
4444     \xdef\bbl@languages{\bbl@languages}%
4445 \fi
4446 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4447 \bbl@languages
4448 \openin\bbl@readstream=language.dat
4449 \ifeof\bbl@readstream
4450     \bbl@warning{I couldn't find language.dat. No additional\\%
4451         patterns loaded. Reported}%
4452 \else
4453     \loop
4454         \endlinechar\m@ne
4455         \read\bbl@readstream to \bbl@line
4456         \endlinechar\^^M
4457         \if \T\ifeof\bbl@readstream F\fi T\relax
4458         \ifx\bbl@line\@empty\else
4459             \edef\bbl@line{\bbl@line\space\space\space}%
4460             \expandafter\bbl@process@line\bbl@line\relax
4461         \fi
4462     \repeat
4463 \fi
4464 \endgroup
4465 \bbl@trace{Macros for reading patterns files}
4466 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4467 \ifx\babelcatcodetablenum\@undefined
4468     \ifx\newcatcodetable\@undefined
4469         \def\babelcatcodetablenum{5211}
4470     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4471 \else
4472     \newcatcodetable\babelcatcodetablenum
4473     \newcatcodetable\bbl@pattcodes
4474 \fi
4475 \else
4476     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4477 \fi
4478 \def\bbl@luapatterns#1#2{%
4479     \bbl@get@enc#1::\@@@
4480     \setbox\z@\hbox\bgroup
4481     \begingroup
4482         \savecatcodetable\babelcatcodetablenum\relax
4483         \initcatcodetable\bbl@pattcodes\relax
4484         \catcodetable\bbl@pattcodes\relax
4485         \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
4486         \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
4487         \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4488         \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4489         \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4490         \catcode\'=12 \catcode\'=12 \catcode\'=12
4491         \input #1\relax
4492     \catcodetable\babelcatcodetablenum\relax
4493 \endgroup
4494 \def\bbl@tempa{#2}%
4495 \ifx\bbl@tempa\@empty\else
4496     \input #2\relax
4497 \fi
4498 \egroup}%
4499 \def\bbl@patterns@lua#1{%
4500     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4501     \csname l@#1\endcsname

```

```

4502 \edef\bbl@tempa{#1}%
4503 \else
4504 \csname l@#1:\f@encoding\endcsname
4505 \edef\bbl@tempa{#1:\f@encoding}%
4506 \fi\relax
4507 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4508 \@ifundefined{bbl@hyphendata@the\language}%
4509 {\def\bbl@elt##1##2##3##4{%
4510 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4511 \def\bbl@tempb{##3}%
4512 \ifx\bbl@tempb@empty\else % if not a synonymous
4513 \def\bbl@tempc{##3}{##4}%
4514 \fi
4515 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4516 \fi}%
4517 \bbl@languages
4518 \@ifundefined{bbl@hyphendata@the\language}%
4519 {\bbl@info{No hyphenation patterns were set for\%
4520 language '\bbl@tempa'. Reported}}%
4521 {\expandafter\expandafter\expandafter\bbl@luapatterns
4522 \csname bbl@hyphendata@the\language\endcsname}}}%
4523 \endinput\fi
4524 % Here ends \ifx\AddBabelHook\@undefined
4525 % A few lines are only read by hyphen.cfg
4526 \ifx\DisableBabelHook\@undefined
4527 \AddBabelHook{luatex}{everylanguage}{%
4528 \def\process@language##1##2##3{%
4529 \def\process@line####1####2 ####3 ####4 {}}}
4530 \AddBabelHook{luatex}{loadpatterns}{%
4531 \input #1\relax
4532 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4533 {{#1}}}%
4534 \AddBabelHook{luatex}{loadexceptions}{%
4535 \input #1\relax
4536 \def\bbl@tempb##1##2{{#1}{#1}}%
4537 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4538 {\expandafter\expandafter\expandafter\bbl@tempb
4539 \csname bbl@hyphendata@the\language\endcsname}}
4540 \endinput\fi
4541 % Here stops reading code for hyphen.cfg
4542 % The following is read the 2nd time it's loaded
4543 \begingroup
4544 \catcode`\%=12
4545 \catcode`\'=12
4546 \catcode`\ "=12
4547 \catcode`\:=12
4548 \directlua{
4549 Babel = Babel or {}
4550 function Babel.bytes(line)
4551 return line:gsub("(.)",
4552 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4553 end
4554 function Babel.begin_process_input()
4555 if luatexbase and luatexbase.add_to_callback then
4556 luatexbase.add_to_callback('process_input_buffer',
4557 Babel.bytes, 'Babel.bytes')
4558 else
4559 Babel.callback = callback.find('process_input_buffer')
4560 callback.register('process_input_buffer',Babel.bytes)

```

```

4561     end
4562 end
4563 function Babel.end_process_input ()
4564     if luatexbase and luatexbase.remove_from_callback then
4565         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4566     else
4567         callback.register('process_input_buffer', Babel.callback)
4568     end
4569 end
4570 function Babel.addpatterns(pp, lg)
4571     local lg = lang.new(lg)
4572     local pats = lang.patterns(lg) or ''
4573     lang.clear_patterns(lg)
4574     for p in pp:gmatch('[^%s]+') do
4575         ss = ''
4576         for i in string.utfcharacters(p:gsub('%d', '')) do
4577             ss = ss .. '%d?' .. i
4578         end
4579         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4580         ss = ss:gsub('%.%%d%?$', '%%.')
4581         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4582         if n == 0 then
4583             tex.sprint(
4584                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4585                 .. p .. [[{}]])
4586             pats = pats .. ' ' .. p
4587         else
4588             tex.sprint(
4589                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4590                 .. p .. [[{}]])
4591         end
4592     end
4593     lang.patterns(lg, pats)
4594 end
4595 }
4596 \endgroup
4597 \ifx\newattribute\undefined\else
4598     \newattribute\bbl@attr@locale
4599     \directlua{Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4600     \AddBabelHook{luatex}{beforeextras}{%
4601         \setattribute\bbl@attr@locale\localeid}
4602 \fi
4603 \def\BabelStringsDefault{unicode}
4604 \let\luabbl@stop\relax
4605 \AddBabelHook{luatex}{encodedcommands}{%
4606     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4607     \ifx\bbl@tempa\bbl@tempb\else
4608         \directlua{Babel.begin_process_input()}%
4609         \def\luabbl@stop{%
4610             \directlua{Babel.end_process_input()}}%
4611     \fi}%
4612 \AddBabelHook{luatex}{stopcommands}{%
4613     \luabbl@stop
4614     \let\luabbl@stop\relax}
4615 \AddBabelHook{luatex}{patterns}{%
4616     \@ifundefined{bbl@hyphendata@the\language}%
4617     {\def\bbl@elt##1##2##3##4{%
4618         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
4619         \def\bbl@tempb{##3}%

```



```

4620         \ifx\bb1@tempb\@empty\else % if not a synonymous
4621         \def\bb1@tempc{##3}##4}%
4622         \fi
4623         \bb1@csarg\xdef{hyphendata@##2}{\bb1@tempc}%
4624     \fi}%
4625 \bb1@languages
4626 \@ifundefined{bb1@hyphendata@the\language}%
4627     {\bb1@info{No hyphenation patterns were set for\%
4628         language '#2'. Reported}}%
4629     {\expandafter\expandafter\expandafter\bb1@luapatterns
4630         \csname bb1@hyphendata@the\language\endcsname}}}%
4631 \@ifundefined{bb1@patterns@}{}%
4632 \begingroup
4633     \bb1@xin@{, \number\language,}{, \bb1@pttnlist}%
4634     \ifin\else
4635         \ifx\bb1@patterns@\@empty\else
4636             \directlua{ Babel.addpatterns(
4637                 [[\bb1@patterns@]], \number\language) }%
4638         \fi
4639         \@ifundefined{bb1@patterns@#1}%
4640             \@empty
4641             {\directlua{ Babel.addpatterns(
4642                 [[\space\csname bb1@patterns@#1\endcsname]],
4643                 \number\language) }}%
4644         \xdef\bb1@pttnlist{\bb1@pttnlist\number\language,}%
4645     \fi
4646 \endgroup}%
4647 \bb1@exp{%
4648     \bb1@ifunset{bb1@prehc@\language\name}{}%
4649     {\bb1@ifblank{\bb1@cs{prehc@\language\name}}}%
4650     {\prehyphenchar=\bb1@c1{prehc}\relax}}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bb1@patterns@` for the global ones and `\bb1@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4651 \@onlypreamble\babelpatterns
4652 \AtEndOfPackage{%
4653     \newcommand\babelpatterns[2][\@empty]{%
4654         \ifx\bb1@patterns@\relax
4655             \let\bb1@patterns@\@empty
4656         \fi
4657         \ifx\bb1@pttnlist\@empty\else
4658             \bb1@warning{%
4659                 You must not intermingle \string\selectlanguage\space and\%
4660                 \string\babelpatterns\space or some patterns will not\%
4661                 be taken into account. Reported}%
4662             \fi
4663             \ifx\@empty#1%
4664                 \protected@edef\bb1@patterns@{\bb1@patterns@\space#2}%
4665             \else
4666                 \edef\bb1@tempb{\zap@space#1 \@empty}%
4667                 \bb1@for\bb1@tempa\bb1@tempb{%
4668                     \bb1@fixname\bb1@tempa
4669                     \bb1@iflanguage\bb1@tempa{%
4670                         \bb1@csarg\protected@edef{patterns@\bb1@tempa}{%
4671                             \@ifundefined{bb1@patterns@\bb1@tempa}%
4672                             \@empty
4673                             {\csname bb1@patterns@\bb1@tempa\endcsname\space}%

```

```

4674         #2}}}%
4675     \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

In progress. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4676 \directlua{
4677   Babel = Babel or {}
4678   Babel.linebreaking = Babel.linebreaking or {}
4679   Babel.linebreaking.before = {}
4680   Babel.linebreaking.after = {}
4681   Babel.locale = {} % Free to use, indexed with \localeid
4682   function Babel.linebreaking.add_before(func)
4683     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4684     table.insert(Babel.linebreaking.before, func)
4685   end
4686   function Babel.linebreaking.add_after(func)
4687     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4688     table.insert(Babel.linebreaking.after, func)
4689   end
4690 }
4691 \def\bbl@intraspace#1 #2 #3\@@{%
4692   \directlua{
4693     Babel = Babel or {}
4694     Babel.intraspaces = Babel.intraspaces or {}
4695     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
4696       {b = #1, p = #2, m = #3}
4697     Babel.locale_props[\the\localeid].intraspace = %
4698       {b = #1, p = #2, m = #3}
4699   }}
4700 \def\bbl@intrapenalty#1\@@{%
4701   \directlua{
4702     Babel = Babel or {}
4703     Babel.intrapenalties = Babel.intrapenalties or {}
4704     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4705     Babel.locale_props[\the\localeid].intrapenalty = #1
4706   }}
4707 \begingroup
4708 \catcode`\%=12
4709 \catcode`\^=14
4710 \catcode`\'=12
4711 \catcode`\~=12
4712 \gdef\bbl@seaintraspace{^
4713   \let\bbl@seaintraspace\relax
4714   \directlua{
4715     Babel = Babel or {}
4716     Babel.sea_enabled = true
4717     Babel.sea_ranges = Babel.sea_ranges or {}
4718     function Babel.set_chranges (script, chrng)
4719       local c = 0
4720       for s, e in string.gmatch(chrng..' ', '(.)%%.(.)%s') do
4721         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4722         c = c + 1
4723       end

```

```

4724 end
4725 function Babel.sea_disc_to_space (head)
4726     local sea_ranges = Babel.sea_ranges
4727     local last_char = nil
4728     local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4729     for item in node.traverse(head) do
4730         local i = item.id
4731         if i == node.id'glyph' then
4732             last_char = item
4733         elseif i == 7 and item.subtype == 3 and last_char
4734             and last_char.char > 0x0C99 then
4735             quad = font.getfont(last_char.font).size
4736             for lg, rg in pairs(sea_ranges) do
4737                 if last_char.char > rg[1] and last_char.char < rg[2] then
4738                     lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
4739                     local intraspace = Babel.intraspaces[lg]
4740                     local intrapenalty = Babel.intrapenalties[lg]
4741                     local n
4742                     if intrapenalty ~= 0 then
4743                         n = node.new(14, 0) ^^ penalty
4744                         n.penalty = intrapenalty
4745                         node.insert_before(head, item, n)
4746                     end
4747                     n = node.new(12, 13) ^^ (glue, spaceskip)
4748                     node.setglue(n, intraspace.b * quad,
4749                                 intraspace.p * quad,
4750                                 intraspace.m * quad)
4751                     node.insert_before(head, item, n)
4752                     node.remove(head, item)
4753                 end
4754             end
4755         end
4756     end
4757 end
4758 }^^
4759 \bbl@luahyphenate}
4760 \catcode`\%=14
4761 \gdef\bbl@cjk intraspace{%
4762 \let\bbl@cjk intraspace\relax
4763 \directlua{
4764     Babel = Babel or {}
4765     require'babel-data-cjk.lua'
4766     Babel.cjk_enabled = true
4767     function Babel.cjk_linebreak(head)
4768         local GLYPH = node.id'glyph'
4769         local last_char = nil
4770         local quad = 655360      % 10 pt = 655360 = 10 * 65536
4771         local last_class = nil
4772         local last_lang = nil
4773
4774         for item in node.traverse(head) do
4775             if item.id == GLYPH then
4776
4777                 local lang = item.lang
4778
4779                 local LOCALE = node.get_attribute(item,
4780                     luatexbase.registernumber'bbl@attr@locale')
4781                 local props = Babel.locale_props[LOCALE]
4782

```

```

4783     local class = Babel.cjk_class[item.char].c
4784
4785     if class == 'cp' then class = 'cl' end % ]] as CL
4786     if class == 'id' then class = 'I' end
4787
4788     local br = 0
4789     if class and last_class and Babel.cjk_breaks[last_class][class] then
4790         br = Babel.cjk_breaks[last_class][class]
4791     end
4792
4793     if br == 1 and props.linebreak == 'c' and
4794         lang ~= \the\l@nohyphenation\space and
4795         last_lang ~= \the\l@nohyphenation then
4796         local intrapenalty = props.intrapenalty
4797         if intrapenalty ~= 0 then
4798             local n = node.new(14, 0)      % penalty
4799             n.penalty = intrapenalty
4800             node.insert_before(head, item, n)
4801         end
4802         local intraspace = props.intraspace
4803         local n = node.new(12, 13)        % (glue, spaceskip)
4804         node.setglue(n, intraspace.b * quad,
4805                       intraspace.p * quad,
4806                       intraspace.m * quad)
4807         node.insert_before(head, item, n)
4808     end
4809
4810     quad = font.getfont(item.font).size
4811     last_class = class
4812     last_lang = lang
4813     else % if penalty, glue or anything else
4814         last_class = nil
4815     end
4816 end
4817 lang.hyphenate(head)
4818 end
4819 }%
4820 \bbl@luahyphenate}
4821 \gdef\bbl@luahyphenate{%
4822 \let\bbl@luahyphenate\relax
4823 \directlua{
4824     luatexbase.add_to_callback('hyphenate',
4825     function (head, tail)
4826         if Babel.linebreaking.before then
4827             for k, func in ipairs(Babel.linebreaking.before) do
4828                 func(head)
4829             end
4830         end
4831         if Babel.cjk_enabled then
4832             Babel.cjk_linebreak(head)
4833         end
4834         lang.hyphenate(head)
4835         if Babel.linebreaking.after then
4836             for k, func in ipairs(Babel.linebreaking.after) do
4837                 func(head)
4838             end
4839         end
4840         if Babel.sea_enabled then
4841             Babel.sea_disc_to_space(head)

```

```

4842     end
4843 end,
4844 'Babel.hyphenate')
4845 }
4846 }
4847 \endgroup
4848 \def\bbl@provide@intraspace{%
4849   \bbl@ifunset{\bbl@intsp@{language}}{%
4850     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
4851       \bbl@xin@{\bbl@cl{lnbrk}}{c}%
4852       \ifin@           % cjk
4853       \bbl@cjk@intraspace
4854       \directlua{
4855         Babel = Babel or {}
4856         Babel.locale_props = Babel.locale_props or {}
4857         Babel.locale_props[\the\localeid].linebreak = 'c'
4858       }%
4859       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4860       \ifx\bbl@KVP@intrapenalty\@nil
4861         \bbl@intrapenalty0\@@
4862       \fi
4863     \else           % sea
4864       \bbl@seaintraspace
4865       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4866       \directlua{
4867         Babel = Babel or {}
4868         Babel.sea_ranges = Babel.sea_ranges or {}
4869         Babel.set_chranges('\bbl@cl{sbcpr}',
4870                           '\bbl@cl{chrng}')
4871       }%
4872       \ifx\bbl@KVP@intrapenalty\@nil
4873         \bbl@intrapenalty0\@@
4874       \fi
4875     \fi
4876   \fi
4877   \ifx\bbl@KVP@intrapenalty\@nil\else
4878     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4879   \fi}}

```

13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

Work in progress.

Common stuff.

```

4880 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4881 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4882 \DisableBabelHook{babel-fontspec}
4883 <<Font selection>>

```

13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
4884 \directlua{
4885 Babel.script_blocks = {
4886   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4887             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4888   ['Armn'] = {{0x0530, 0x058F}},
4889   ['Beng'] = {{0x0980, 0x09FF}},
4890   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
4891   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4892   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4893             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4894   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4895   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4896             {0xAB00, 0xAB2F}},
4897   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4898   % Don't follow strictly Unicode, which places some Coptic letters in
4899   % the 'Greek and Coptic' block
4900   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4901   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4902             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4903             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4904             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4905             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4906             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4907   ['Hebr'] = {{0x0590, 0x05FF}},
4908   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4909             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4910   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4911   ['Knda'] = {{0x0C80, 0x0CFF}},
4912   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4913             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4914             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4915   ['Lao0'] = {{0x0E80, 0x0EFF}},
4916   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4917             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4918             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4919   ['Mahj'] = {{0x11150, 0x1117F}},
4920   ['Mlym'] = {{0x0D00, 0x0D7F}},
4921   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4922   ['Orya'] = {{0x0B00, 0x0B7F}},
4923   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4924   ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
4925   ['Taml'] = {{0x0B80, 0x0BFF}},
4926   ['Telu'] = {{0x0C00, 0x0C7F}},
4927   ['Tfng'] = {{0x2D30, 0x2D7F}},
4928   ['Thai'] = {{0x0E00, 0x0E7F}},
4929   ['Tibt'] = {{0x0F00, 0x0FFF}},
4930   ['Vaii'] = {{0xA500, 0xA63F}},
4931   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
```

```

4932 }
4933
4934 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
4935 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4936 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4937
4938 function Babel.locale_map(head)
4939   if not Babel.locale_mapped then return head end
4940
4941   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
4942   local GLYPH = node.id('glyph')
4943   local inmath = false
4944   local toloc_save
4945   for item in node.traverse(head) do
4946     local toloc
4947     if not inmath and item.id == GLYPH then
4948       % Optimization: build a table with the chars found
4949       if Babel.chr_to_loc[item.char] then
4950         toloc = Babel.chr_to_loc[item.char]
4951       else
4952         for lc, maps in pairs(Babel.loc_to_scr) do
4953           for _, rg in pairs(maps) do
4954             if item.char >= rg[1] and item.char <= rg[2] then
4955               Babel.chr_to_loc[item.char] = lc
4956               toloc = lc
4957               break
4958             end
4959           end
4960         end
4961       end
4962       % Now, take action, but treat composite chars in a different
4963       % fashion, because they 'inherit' the previous locale. Not yet
4964       % optimized.
4965       if not toloc and
4966         (item.char >= 0x0300 and item.char <= 0x036F) or
4967         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4968         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
4969         toloc = toloc_save
4970       end
4971       if toloc and toloc > -1 then
4972         if Babel.locale_props[toloc].lg then
4973           item.lang = Babel.locale_props[toloc].lg
4974           node.set_attribute(item, LOCALE, toloc)
4975         end
4976         if Babel.locale_props[toloc]['/'..item.font] then
4977           item.font = Babel.locale_props[toloc]['/'..item.font]
4978         end
4979         toloc_save = toloc
4980       end
4981     elseif not inmath and item.id == 7 then
4982       item.replace = item.replace and Babel.locale_map(item.replace)
4983       item.pre = item.pre and Babel.locale_map(item.pre)
4984       item.post = item.post and Babel.locale_map(item.post)
4985     elseif item.id == node.id'math' then
4986       inmath = (item.subtype == 0)
4987     end
4988   end
4989   return head
4990 end

```

4991 }

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

4992 \newcommand\babelcharproperty[1]{%
4993   \count@=#1\relax
4994   \ifvmode
4995     \expandafter\babel@chprop
4996   \else
4997     \babel@error{\string\babelcharproperty\space can be used only in\%
4998       vertical mode (preamble or between paragraphs)}%
4999     {See the manual for futher info}%
5000   \fi}
5001 \newcommand\babel@chprop[3][\the\count@]{%
5002   \@tempcnta=#1\relax
5003   \babel@ifunset{\babel@chprop@#2}%
5004   {\babel@error{No property named '#2'. Allowed values are\%
5005     direction (bc), mirror (bmg), and linebreak (lb)}%
5006     {See the manual for futher info}}%
5007   }%
5008   \loop
5009     \babel@cs{\babel@chprop@#2}{#3}%
5010     \ifnum\count@<\@tempcnta
5011       \advance\count@\@ne
5012     \repeat}
5013 \def\babel@chprop@direction#1{%
5014   \directlua{
5015     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5016     Babel.characters[\the\count@]['d'] = '#1'
5017   }}
5018 \let\babel@chprop@bc\babel@chprop@direction
5019 \def\babel@chprop@mirror#1{%
5020   \directlua{
5021     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5022     Babel.characters[\the\count@]['m'] = '\number#1'
5023   }}
5024 \let\babel@chprop@bmg\babel@chprop@mirror
5025 \def\babel@chprop@linebreak#1{%
5026   \directlua{
5027     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5028     Babel.cjk_characters[\the\count@]['c'] = '#1'
5029   }}
5030 \let\babel@chprop@lb\babel@chprop@linebreak
5031 \def\babel@chprop@locale#1{%
5032   \directlua{
5033     Babel.chr_to_loc = Babel.chr_to_loc or {}
5034     Babel.chr_to_loc[\the\count@] =
5035       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
5036   }}

```

Post-handling hyphenation patterns for non-standard rules, like `ff` to `ff-f`. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as

explained in the luatex manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

5037 \begingroup
5038 \catcode`\#=12
5039 \catcode`\%=12
5040 \catcode`\&=14
5041 \directlua{
5042   Babel.linebreaking.post_replacements = {}
5043   Babel.linebreaking.pre_replacements = {}
5044
5045   function Babel.str_to_nodes(fn, matches, base)
5046     local n, head, last
5047     if fn == nil then return nil end
5048     for s in string.utfvalues(fn(matches)) do
5049       if base.id == 7 then
5050         base = base.replace
5051       end
5052       n = node.copy(base)
5053       n.char = s
5054       if not head then
5055         head = n
5056       else
5057         last.next = n
5058       end
5059       last = n
5060     end
5061     return head
5062   end
5063
5064   function Babel.fetch_word(head, funct)
5065     local word_string = ''
5066     local word_nodes = {}
5067     local lang
5068     local item = head
5069     local inmath = false
5070
5071     while item do
5072
5073       if item.id == 29
5074         and not(item.char == 124) && ie, not |
5075         and not(item.char == 61) && ie, not =
5076         and not inmath
5077         and (item.lang == lang or lang == nil) then
5078         lang = lang or item.lang
5079         word_string = word_string .. unicode.utf8.char(item.char)
5080         word_nodes[#word_nodes+1] = item
5081
5082       elseif item.id == 7 and item.subtype == 2 and not inmath then
5083         word_string = word_string .. '='
5084         word_nodes[#word_nodes+1] = item
5085
5086       elseif item.id == 7 and item.subtype == 3 and not inmath then
5087         word_string = word_string .. '|'
5088         word_nodes[#word_nodes+1] = item
5089
5090       elseif item.id == 11 and item.subtype == 0 then

```

```

5091         inmath = true
5092
5093     elseif word_string == '' then
5094         %% pass
5095
5096     else
5097         return word_string, word_nodes, item, lang
5098     end
5099
5100     item = item.next
5101 end
5102 end
5103
5104 function Babel.post_hyphenate_replace(head)
5105     local u = unicode.utf8
5106     local lbkr = Babel.linebreaking.post_replacements
5107     local word_head = head
5108
5109     while true do
5110         local w, wn, nw, lang = Babel.fetch_word(word_head)
5111         if not lang then return head end
5112
5113         if not lbkr[lang] then
5114             break
5115         end
5116
5117         for k=1, #lbkr[lang] do
5118             local p = lbkr[lang][k].pattern
5119             local r = lbkr[lang][k].replace
5120
5121             while true do
5122                 local matches = { u.match(w, p) }
5123                 if #matches < 2 then break end
5124
5125                 local first = table.remove(matches, 1)
5126                 local last = table.remove(matches, #matches)
5127
5128                 %% Fix offsets, from bytes to unicode.
5129                 first = u.len(w:sub(1, first-1)) + 1
5130                 last = u.len(w:sub(1, last-1))
5131
5132                 local new %% used when inserting and removing nodes
5133                 local changed = 0
5134
5135                 %% This loop traverses the replace list and takes the
5136                 %% corresponding actions
5137                 for q = first, last do
5138                     local crep = r[q-first+1]
5139                     local char_node = wn[q]
5140                     local char_base = char_node
5141
5142                     if crep and crep.data then
5143                         char_base = wn[crep.data+first-1]
5144                     end
5145
5146                     if crep == {} then
5147                         break
5148                     elseif crep == nil then
5149                         changed = changed + 1

```

```

5150         node.remove(head, char_node)
5151     elseif crep and (crep.pre or crep.no or crep.post) then
5152         changed = changed + 1
5153         d = node.new(7, 0)    %% (disc, discretionary)
5154         d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5155         d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5156         d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5157         d.attr = char_base.attr
5158         if crep.pre == nil then    %% TeXbook p96
5159             d.penalty = crep.penalty or tex.hyphenpenalty
5160         else
5161             d.penalty = crep.penalty or tex.exhyphenpenalty
5162         end
5163         head, new = node.insert_before(head, char_node, d)
5164         node.remove(head, char_node)
5165         if q == 1 then
5166             word_head = new
5167         end
5168     elseif crep and crep.string then
5169         changed = changed + 1
5170         local str = crep.string(matches)
5171         if str == '' then
5172             if q == 1 then
5173                 word_head = char_node.next
5174             end
5175             head, new = node.remove(head, char_node)
5176         elseif char_node.id == 29 and u.len(str) == 1 then
5177             char_node.char = string.utfvalue(str)
5178         else
5179             local n
5180             for s in string.utfvalues(str) do
5181                 if char_node.id == 7 then
5182                     log('Automatic hyphens cannot be replaced, just removed.')
5183                 else
5184                     n = node.copy(char_base)
5185                 end
5186                 n.char = s
5187                 if q == 1 then
5188                     head, new = node.insert_before(head, char_node, n)
5189                     word_head = new
5190                 else
5191                     node.insert_before(head, char_node, n)
5192                 end
5193             end
5194             node.remove(head, char_node)
5195         end    %% string length
5196     end    %% if char and char.string
5197 end    %% for char in match
5198 if changed > 20 then
5199     texio.write('Too many changes. Ignoring the rest.')
5200 elseif changed > 0 then
5201     w, wn, nw = Babel.fetch_word(word_head)
5202 end
5203 end
5204
5205     end    %% for match
5206 end    %% for patterns
5207 word_head = nw
5208 end    %% for words

```

```

5209     return head
5210 end
5211
5212 &%%&
5213 &%% Preliminary code for \babelprehyphenation
5214 &%% TODO. Copypaste pattern. Merge with fetch_word
5215 function Babel.fetch_subtext(head, funct)
5216     local word_string = ''
5217     local word_nodes = {}
5218     local lang
5219     local item = head
5220     local inmath = false
5221
5222     while item do
5223
5224         if item.id == 29 then
5225             local locale = node.get_attribute(item, Babel.attr_locale)
5226
5227             if not(item.char == 124) &%% ie, not | = space
5228                 and not inmath
5229                 and (locale == lang or lang == nil) then
5230                 lang = lang or locale
5231                 word_string = word_string .. unicode.utf8.char(item.char)
5232                 word_nodes[#word_nodes+1] = item
5233             end
5234
5235             if item == node.tail(head) then
5236                 item = nil
5237                 return word_string, word_nodes, item, lang
5238             end
5239
5240             elseif item.id == 12 and item.subtype == 13 and not inmath then
5241                 word_string = word_string .. '|'
5242                 word_nodes[#word_nodes+1] = item
5243
5244                 if item == node.tail(head) then
5245                     item = nil
5246                     return word_string, word_nodes, item, lang
5247                 end
5248
5249             elseif item.id == 11 and item.subtype == 0 then
5250                 inmath = true
5251
5252             elseif word_string == '' then
5253                 &%% pass
5254
5255             else
5256                 return word_string, word_nodes, item, lang
5257             end
5258
5259             item = item.next
5260         end
5261     end
5262
5263 &%% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5264 function Babel.pre_hyphenate_replace(head)
5265     local u = unicode.utf8
5266     local lbkr = Babel.linebreaking.pre_replacements
5267     local word_head = head

```

```

5268
5269 while true do
5270     local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5271     if not lang then return head end
5272
5273     if not lbkr[lang] then
5274         break
5275     end
5276
5277     for k=1, #lbkr[lang] do
5278         local p = lbkr[lang][k].pattern
5279         local r = lbkr[lang][k].replace
5280
5281         while true do
5282             local matches = { u.match(w, p) }
5283             if #matches < 2 then break end
5284
5285             local first = table.remove(matches, 1)
5286             local last = table.remove(matches, #matches)
5287
5288             %% Fix offsets, from bytes to unicode.
5289             first = u.len(w:sub(1, first-1)) + 1
5290             last = u.len(w:sub(1, last-1))
5291
5292             local new %% used when inserting and removing nodes
5293             local changed = 0
5294
5295             %% This loop traverses the replace list and takes the
5296             %% corresponding actions
5297             for q = first, last do
5298                 local crep = r[q-first+1]
5299                 local char_node = wn[q]
5300                 local char_base = char_node
5301
5302                 if crep and crep.data then
5303                     char_base = wn[crep.data+first-1]
5304                 end
5305
5306                 if crep == {} then
5307                     break
5308                 elseif crep == nil then
5309                     changed = changed + 1
5310                     node.remove(head, char_node)
5311                 elseif crep and crep.string then
5312                     changed = changed + 1
5313                     local str = crep.string(matches)
5314                     if str == '' then
5315                         if q == 1 then
5316                             word_head = char_node.next
5317                         end
5318                         head, new = node.remove(head, char_node)
5319                     elseif char_node.id == 29 and u.len(str) == 1 then
5320                         char_node.char = string.utfvalue(str)
5321                     else
5322                         local n
5323                         for s in string.utfvalues(str) do
5324                             if char_node.id == 7 then
5325                                 log('Automatic hyphens cannot be replaced, just removed.')
5326                             else

```

```

5327         n = node.copy(char_base)
5328     end
5329     n.char = s
5330     if q == 1 then
5331         head, new = node.insert_before(head, char_node, n)
5332         word_head = new
5333     else
5334         node.insert_before(head, char_node, n)
5335     end
5336 end
5337
5338     node.remove(head, char_node)
5339 end %% string length
5340 end %% if char and char.string
5341 end %% for char in match
5342 if changed > 20 then
5343     texio.write('Too many changes. Ignoring the rest.')
5344 elseif changed > 0 then
5345     %% For one-to-one can we modify directly the
5346     %% values without re-fetching? Very likely.
5347     w, wn, nw = Babel.fetch_subtext(word_head)
5348 end
5349
5350 end %% for match
5351 end %% for patterns
5352 word_head = nw
5353 end %% for words
5354 return head
5355 end
5356 %%% end of preliminary code for \babelprehyphenation
5357
5358 %% The following functions belong to the next macro
5359
5360 %% This table stores capture maps, numbered consecutively
5361 Babel.capture_maps = {}
5362
5363 function Babel.capture_func(key, cap)
5364     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..["] .. "]"
5365     ret = ret:gsub('{{[0-9]}|([^\]|+)|(-)}', Babel.capture_func_map)
5366     ret = ret:gsub("%[%]%%.", '')
5367     ret = ret:gsub("%.%[%]%%", '')
5368     return key .. "[=function(m) return ]] .. ret .. [ end]"
5369 end
5370
5371 function Babel.capt_map(from, mapno)
5372     return Babel.capture_maps[mapno][from] or from
5373 end
5374
5375 %% Handle the {n|abc|ABC} syntax in captures
5376 function Babel.capture_func_map(capno, from, to)
5377     local froms = {}
5378     for s in string.utfcharacters(from) do
5379         table.insert(froms, s)
5380     end
5381     local cnt = 1
5382     table.insert(Babel.capture_maps, {})
5383     local mlen = table.getn(Babel.capture_maps)
5384     for s in string.utfcharacters(to) do
5385         Babel.capture_maps[mlen][froms[cnt]] = s

```

```

5386     cnt = cnt + 1
5387 end
5388 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5389     (mlen) .. " ).. " .. "["
5390 end
5391 }

```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ - becomes `function(m) return m[1]..m[1]..'-' end`, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to $m[1]$. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load - save the code as string in a \TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).`

```

5392 \catcode`\#=6
5393 \gdef\babelposthyphenation#1#2#3{&%
5394   \bbl@activateposthyphen
5395   \begingroup
5396     \def\babeltempa{\bbl@add@list\babeltempb}&%
5397     \let\babeltempb\@empty
5398     \bbl@foreach{#3}{&%
5399       \bbl@ifsamestring{##1}{remove}&%
5400       {\bbl@add@list\babeltempb{nil}}&%
5401       {\directlua{
5402         local rep = [[##1]]
5403         rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5404         rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5405         rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5406         rep = rep:gsub( '(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5407         tex.print([[string\babeltempa{}} .. rep .. {}]])
5408       }}&%
5409     \directlua{
5410       local lbkr = Babel.linebreaking.post_replacements
5411       local u = unicode.utf8
5412       &% Convert pattern:
5413       local patt = string.gsub([==[#2]==], '%s', '')
5414       if not u.find(patt, '()', nil, true) then
5415         patt = '()' .. patt .. '()'
5416       end
5417       patt = u.gsub(patt, '{(.)}',
5418         function (n)
5419           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5420         end)
5421       lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5422       table.insert(lbkr[\the\csname l@#1\endcsname],
5423         { pattern = patt, replace = { \babeltempb } })
5424     }&%
5425   \endgroup}
5426 % TODO. Working !!! Copypaste pattern.
5427 \gdef\babelprehyphenation#1#2#3{&%
5428   \bbl@activateprehyphen
5429   \begingroup
5430     \def\babeltempa{\bbl@add@list\babeltempb}&%
5431     \let\babeltempb\@empty
5432     \bbl@foreach{#3}{&%

```

```

5433 \bbl@ifsamestring{##1}{remove}&%
5434 {\bbl@add@list\babeltempb{nil}}&%
5435 {\directlua{
5436     local rep = [[##1]]
5437     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5438     tex.print([[string\babeltempa{}}] .. rep .. [[]]])
5439 }}&%
5440 \directlua{
5441     local lbkr = Babel.linebreaking.pre_replacements
5442     local u = unicode.utf8
5443     &% Convert pattern:
5444     local patt = string.gsub([==[#2]==], '%s', '')
5445     if not u.find(patt, '()', nil, true) then
5446         patt = '()' .. patt .. '()'
5447     end
5448     patt = u.gsub(patt, '{(.)}',
5449         function (n)
5450             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5451         end)
5452     lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5453     table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5454         { pattern = patt, replace = { \babeltempb } })
5455 }&%
5456 \endgroup}
5457 \endgroup
5458 \def\bbl@activateposthyphen{%
5459 \let\bbl@activateposthyphen\relax
5460 \directlua{
5461     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5462 }}
5463 % TODO. Working !!!
5464 \def\bbl@activateprehyphen{%
5465 \let\bbl@activateprehyphen\relax
5466 \directlua{
5467     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5468 }}

```

13.7 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5469 \bbl@trace{Redefinitions for bidi layout}
5470 \ifx\@eqnnum\undefined\else
5471 \ifx\bbl@attr@dir\undefined\else
5472 \edef\@eqnnum{%
5473     \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5474     \unexpanded\expandafter{\@eqnnum}}

```



```

5475 \fi
5476 \fi
5477 \ifx\bbbl@opt@layout\@nnil\endinput\fi % if no layout
5478 \ifnum\bbbl@bidimode>\z@
5479 \def\bbbl@nextfake#1{% non-local changes, use always inside a group!
5480 \bbbl@exp{%
5481 \mathdir\the\bodydir
5482 #1% Once entered in math, set boxes to restore values
5483 \<ifmmode>%
5484 \everyvbox{%
5485 \the\everyvbox
5486 \bodydir\the\bodydir
5487 \mathdir\the\mathdir
5488 \everyhbox{\the\everyhbox}%
5489 \everyvbox{\the\everyvbox}}%
5490 \everyhbox{%
5491 \the\everyhbox
5492 \bodydir\the\bodydir
5493 \mathdir\the\mathdir
5494 \everyhbox{\the\everyhbox}%
5495 \everyvbox{\the\everyvbox}}%
5496 \<fi>}}%
5497 \def\@hangfrom#1{%
5498 \setbox\@tempboxa\hbox{\{#1\}}%
5499 \hangindent\wd\@tempboxa
5500 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
5501 \shapemode\@ne
5502 \fi
5503 \noindent\box\@tempboxa}
5504 \fi
5505 \IfBabelLayout{tabular}
5506 {\let\bbbl@OL@tabular\@tabular
5507 \bbbl@replace\@tabular{\$}\bbbl@nextfake$}%
5508 \let\bbbl@NL@tabular\@tabular
5509 \AtBeginDocument{%
5510 \ifx\bbbl@NL@tabular\@tabular\else
5511 \bbbl@replace\@tabular{\$}\bbbl@nextfake$}%
5512 \let\bbbl@NL@tabular\@tabular
5513 \fi}}
5514 {}
5515 \IfBabelLayout{lists}
5516 {\let\bbbl@OL@list\list
5517 \bbbl@sreplace\list{\parshape}\bbbl@listparshape}%
5518 \let\bbbl@NL@list\list
5519 \def\bbbl@listparshape#1#2#3{%
5520 \parshape #1 #2 #3 %
5521 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
5522 \shapemode\tw@
5523 \fi}}
5524 {}
5525 \IfBabelLayout{graphics}
5526 {\let\bbbl@pictresetdir\relax
5527 \def\bbbl@pictsetdir{%
5528 \ifcase\bbbl@thetextdir
5529 \let\bbbl@pictresetdir\relax
5530 \else
5531 \textdir TLT\relax
5532 \def\bbbl@pictresetdir{\textdir TRT\relax}%
5533 \fi}%

```


13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

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

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

In some sense, there are two bidi modes, one for numbers, and the other for text.

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

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

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

```
5583 (*basic-r)
5584 Babel = Babel or {}
5585
5586 Babel.bidi_enabled = true
5587
5588 require('babel-data-bidi.lua')
5589
5590 local characters = Babel.characters
5591 local ranges = Babel.ranges
5592
5593 local DIR = node.id("dir")
5594
5595 local function dir_mark(head, from, to, outer)
5596   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
```

```

5597 local d = node.new(DIR)
5598 d.dir = '+' .. dir
5599 node.insert_before(head, from, d)
5600 d = node.new(DIR)
5601 d.dir = '-' .. dir
5602 node.insert_after(head, to, d)
5603 end
5604
5605 function Babel.bidi(head, ispar)
5606   local first_n, last_n      -- first and last char with nums
5607   local last_es              -- an auxiliary 'last' used with nums
5608   local first_d, last_d      -- first and last char in L/R block
5609   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):

```

5610 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5611 local strong_lr = (strong == 'l') and 'l' or 'r'
5612 local outer = strong
5613
5614 local new_dir = false
5615 local first_dir = false
5616 local inmath = false
5617
5618 local last_lr
5619
5620 local type_n = ''
5621
5622 for item in node.traverse(head) do
5623
5624   -- three cases: glyph, dir, otherwise
5625   if item.id == node.id'glyph'
5626     or (item.id == 7 and item.subtype == 2) then
5627
5628     local itemchar
5629     if item.id == 7 and item.subtype == 2 then
5630       itemchar = item.replace.char
5631     else
5632       itemchar = item.char
5633     end
5634     local chardata = characters[itemchar]
5635     dir = chardata and chardata.d or nil
5636     if not dir then
5637       for nn, et in ipairs(ranges) do
5638         if itemchar < et[1] then
5639           break
5640         elseif itemchar <= et[2] then
5641           dir = et[3]
5642           break
5643         end
5644       end
5645     end
5646     dir = dir or 'l'
5647     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).

```

5648     if new_dir then
5649         attr_dir = 0
5650         for at in node.traverse(item.attr) do
5651             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5652                 attr_dir = at.value % 3
5653             end
5654         end
5655         if attr_dir == 1 then
5656             strong = 'r'
5657         elseif attr_dir == 2 then
5658             strong = 'al'
5659         else
5660             strong = 'l'
5661         end
5662         strong_lr = (strong == 'l') and 'l' or 'r'
5663         outer = strong_lr
5664         new_dir = false
5665     end
5666
5667     if dir == 'nsm' then dir = strong end -- W1

```

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

```

5668     dir_real = dir -- We need dir_real to set strong below
5669     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:

```

5670     if strong == 'al' then
5671         if dir == 'en' then dir = 'an' end -- W2
5672         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5673         strong_lr = 'r' -- W3
5674     end

```

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

```

5675     elseif item.id == node.id'dir' and not inmath then
5676         new_dir = true
5677         dir = nil
5678     elseif item.id == node.id'math' then
5679         inmath = (item.subtype == 0)
5680     else
5681         dir = nil -- Not a char
5682     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 `texdir` 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>`.

```

5683     if dir == 'en' or dir == 'an' or dir == 'et' then
5684         if dir ~= 'et' then
5685             type_n = dir
5686         end
5687         first_n = first_n or item
5688         last_n = last_es or item

```

```

5689     last_es = nil
5690   elseif dir == 'es' and last_n then -- W3+W6
5691     last_es = item
5692   elseif dir == 'cs' then           -- it's right - do nothing
5693   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5694     if strong_lr == 'r' and type_n ~= '' then
5695       dir_mark(head, first_n, last_n, 'r')
5696     elseif strong_lr == 'l' and first_d and type_n == 'an' then
5697       dir_mark(head, first_n, last_n, 'r')
5698       dir_mark(head, first_d, last_d, outer)
5699       first_d, last_d = nil, nil
5700     elseif strong_lr == 'l' and type_n ~= '' then
5701       last_d = last_n
5702     end
5703     type_n = ''
5704     first_n, last_n = nil, nil
5705   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:

```

5706   if dir == 'l' or dir == 'r' then
5707     if dir ~= outer then
5708       first_d = first_d or item
5709       last_d = item
5710     elseif first_d and dir ~= strong_lr then
5711       dir_mark(head, first_d, last_d, outer)
5712       first_d, last_d = nil, nil
5713     end
5714   end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

5715   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5716     item.char = characters[item.char] and
5717       characters[item.char].m or item.char
5718   elseif (dir or new_dir) and last_lr ~= item then
5719     local mir = outer .. strong_lr .. (dir or outer)
5720     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5721       for ch in node.traverse(node.next(last_lr)) do
5722         if ch == item then break end
5723         if ch.id == node.id'glyph' and characters[ch.char] then
5724           ch.char = characters[ch.char].m or ch.char
5725         end
5726       end
5727     end
5728   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).

```

5729   if dir == 'l' or dir == 'r' then
5730     last_lr = item
5731     strong = dir_real           -- Don't search back - best save now
5732     strong_lr = (strong == 'l') and 'l' or 'r'

```

```

5733     elseif new_dir then
5734         last_lr = nil
5735     end
5736 end

```

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

```

5737 if last_lr and outer == 'r' then
5738     for ch in node.traverse_id(node.id('glyph', node.next(last_lr))) do
5739         if characters[ch.char] then
5740             ch.char = characters[ch.char].m or ch.char
5741         end
5742     end
5743 end
5744 if first_n then
5745     dir_mark(head, first_n, last_n, outer)
5746 end
5747 if first_d then
5748     dir_mark(head, first_d, last_d, outer)
5749 end

```

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

```

5750 return node.prev(head) or head
5751 end
5752 </basic-r>

```

And here the Lua code for bidi=basic:

```

5753 (*basic)
5754 Babel = Babel or {}
5755
5756 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5757
5758 Babel.fontmap = Babel.fontmap or {}
5759 Babel.fontmap[0] = {}      -- l
5760 Babel.fontmap[1] = {}      -- r
5761 Babel.fontmap[2] = {}      -- al/an
5762
5763 Babel.bidi_enabled = true
5764 Babel.mirroring_enabled = true
5765
5766 require('babel-data-bidi.lua')
5767
5768 local characters = Babel.characters
5769 local ranges = Babel.ranges
5770
5771 local DIR = node.id('dir')
5772 local GLYPH = node.id('glyph')
5773
5774 local function insert_implicit(head, state, outer)
5775     local new_state = state
5776     if state.sim and state.eim and state.sim ~= state.eim then
5777         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5778         local d = node.new(DIR)
5779         d.dir = '+' .. dir
5780         node.insert_before(head, state.sim, d)
5781         local d = node.new(DIR)
5782         d.dir = '-' .. dir
5783         node.insert_after(head, state.eim, d)
5784     end

```

```

5785 new_state.sim, new_state.eim = nil, nil
5786 return head, new_state
5787 end
5788
5789 local function insert_numeric(head, state)
5790   local new
5791   local new_state = state
5792   if state.san and state.ean and state.san ~= state.ean then
5793     local d = node.new(DIR)
5794     d.dir = '+TLT'
5795     _, new = node.insert_before(head, state.san, d)
5796     if state.san == state.sim then state.sim = new end
5797     local d = node.new(DIR)
5798     d.dir = '-TLT'
5799     _, new = node.insert_after(head, state.ean, d)
5800     if state.ean == state.eim then state.eim = new end
5801   end
5802   new_state.san, new_state.ean = nil, nil
5803   return head, new_state
5804 end
5805
5806 -- TODO - \hbox with an explicit dir can lead to wrong results
5807 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5808 -- was made to improve the situation, but the problem is the 3-dir
5809 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5810 -- well.
5811
5812 function Babel.bidi(head, ispar, hdir)
5813   local d -- d is used mainly for computations in a loop
5814   local prev_d = ''
5815   local new_d = false
5816
5817   local nodes = {}
5818   local outer_first = nil
5819   local inmath = false
5820
5821   local glue_d = nil
5822   local glue_i = nil
5823
5824   local has_en = false
5825   local first_et = nil
5826
5827   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5828
5829   local save_outer
5830   local temp = node.get_attribute(head, ATDIR)
5831   if temp then
5832     temp = temp % 3
5833     save_outer = (temp == 0 and 'l') or
5834                  (temp == 1 and 'r') or
5835                  (temp == 2 and 'al')
5836   elseif ispar then -- Or error? Shouldn't happen
5837     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5838   else -- Or error? Shouldn't happen
5839     save_outer = ('TRT' == hdir) and 'r' or 'l'
5840   end
5841   -- when the callback is called, we are just _after_ the box,
5842   -- and the textdir is that of the surrounding text
5843   -- if not ispar and hdir ~= tex.textdir then

```



```

5844 -- save_outer = ('TRT' == hdir) and 'r' or 'l'
5845 -- end
5846 local outer = save_outer
5847 local last = outer
5848 -- 'al' is only taken into account in the first, current loop
5849 if save_outer == 'al' then save_outer = 'r' end
5850
5851 local fontmap = Babel.fontmap
5852
5853 for item in node.traverse(head) do
5854
5855     -- In what follows, #node is the last (previous) node, because the
5856     -- current one is not added until we start processing the neutrals.
5857
5858     -- three cases: glyph, dir, otherwise
5859     if item.id == GLYPH
5860         or (item.id == 7 and item.subtype == 2) then
5861
5862         local d_font = nil
5863         local item_r
5864         if item.id == 7 and item.subtype == 2 then
5865             item_r = item.replace -- automatic discs have just 1 glyph
5866         else
5867             item_r = item
5868         end
5869         local chardata = characters[item_r.char]
5870         d = chardata and chardata.d or nil
5871         if not d or d == 'nsm' then
5872             for nn, et in ipairs(ranges) do
5873                 if item_r.char < et[1] then
5874                     break
5875                 elseif item_r.char <= et[2] then
5876                     if not d then d = et[3]
5877                     elseif d == 'nsm' then d_font = et[3]
5878                 end
5879                 break
5880             end
5881         end
5882         d = d or 'l'
5883
5884         -- A short 'pause' in bidi for mapfont
5885         d_font = d_font or d
5886         d_font = (d_font == 'l' and 0) or
5887             (d_font == 'nsm' and 0) or
5888             (d_font == 'r' and 1) or
5889             (d_font == 'al' and 2) or
5890             (d_font == 'an' and 2) or nil
5891         if d_font and fontmap and fontmap[d_font][item_r.font] then
5892             item_r.font = fontmap[d_font][item_r.font]
5893         end
5894
5895         if new_d then
5896             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5897             if inmath then
5898                 attr_d = 0
5899             else
5900                 attr_d = node.get_attribute(item, ATDIR)
5901                 attr_d = attr_d % 3

```

```

5903         end
5904         if attr_d == 1 then
5905             outer_first = 'r'
5906             last = 'r'
5907         elseif attr_d == 2 then
5908             outer_first = 'r'
5909             last = 'al'
5910         else
5911             outer_first = 'l'
5912             last = 'l'
5913         end
5914         outer = last
5915         has_en = false
5916         first_et = nil
5917         new_d = false
5918     end
5919
5920     if glue_d then
5921         if (d == 'l' and 'l' or 'r') ~= glue_d then
5922             table.insert(nodes, {glue_i, 'on', nil})
5923         end
5924         glue_d = nil
5925         glue_i = nil
5926     end
5927
5928     elseif item.id == DIR then
5929         d = nil
5930         new_d = true
5931
5932     elseif item.id == node.id'glue' and item.subtype == 13 then
5933         glue_d = d
5934         glue_i = item
5935         d = nil
5936
5937     elseif item.id == node.id'math' then
5938         inmath = (item.subtype == 0)
5939
5940     else
5941         d = nil
5942     end
5943
5944     -- AL <= EN/ET/ES      -- W2 + W3 + W6
5945     if last == 'al' and d == 'en' then
5946         d = 'an'          -- W3
5947     elseif last == 'al' and (d == 'et' or d == 'es') then
5948         d = 'on'          -- W6
5949     end
5950
5951     -- EN + CS/ES + EN      -- W4
5952     if d == 'en' and #nodes >= 2 then
5953         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5954             and nodes[#nodes-1][2] == 'en' then
5955             nodes[#nodes][2] = 'en'
5956         end
5957     end
5958
5959     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
5960     if d == 'an' and #nodes >= 2 then
5961         if (nodes[#nodes][2] == 'cs')

```

```

5962         and nodes[#nodes-1][2] == 'an' then
5963             nodes[#nodes][2] = 'an'
5964         end
5965     end
5966
5967     -- ET/EN          -- W5 + W7->1 / W6->on
5968     if d == 'et' then
5969         first_et = first_et or (#nodes + 1)
5970     elseif d == 'en' then
5971         has_en = true
5972         first_et = first_et or (#nodes + 1)
5973     elseif first_et then      -- d may be nil here !
5974         if has_en then
5975             if last == 'l' then
5976                 temp = 'l'      -- W7
5977             else
5978                 temp = 'en'     -- W5
5979             end
5980         else
5981             temp = 'on'         -- W6
5982         end
5983         for e = first_et, #nodes do
5984             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5985         end
5986         first_et = nil
5987         has_en = false
5988     end
5989
5990     if d then
5991         if d == 'al' then
5992             d = 'r'
5993             last = 'al'
5994         elseif d == 'l' or d == 'r' then
5995             last = d
5996         end
5997         prev_d = d
5998         table.insert(nodes, {item, d, outer_first})
5999     end
6000
6001     outer_first = nil
6002
6003 end
6004
6005 -- TODO -- repeated here in case EN/ET is the last node. Find a
6006 -- better way of doing things:
6007 if first_et then      -- dir may be nil here !
6008     if has_en then
6009         if last == 'l' then
6010             temp = 'l'      -- W7
6011         else
6012             temp = 'en'     -- W5
6013         end
6014     else
6015         temp = 'on'         -- W6
6016     end
6017     for e = first_et, #nodes do
6018         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6019     end
6020 end

```

```

6021
6022 -- dummy node, to close things
6023 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6024
6025 ----- NEUTRAL -----
6026
6027 outer = save_outer
6028 last = outer
6029
6030 local first_on = nil
6031
6032 for q = 1, #nodes do
6033     local item
6034
6035     local outer_first = nodes[q][3]
6036     outer = outer_first or outer
6037     last = outer_first or last
6038
6039     local d = nodes[q][2]
6040     if d == 'an' or d == 'en' then d = 'r' end
6041     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6042
6043     if d == 'on' then
6044         first_on = first_on or q
6045     elseif first_on then
6046         if last == d then
6047             temp = d
6048         else
6049             temp = outer
6050         end
6051         for r = first_on, q - 1 do
6052             nodes[r][2] = temp
6053             item = nodes[r][1] -- MIRRORING
6054             if Babel.mirroring_enabled and item.id == GLYPH
6055                 and temp == 'r' and characters[item.char] then
6056                 local font_mode = font.fonts[item.font].properties.mode
6057                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6058                     item.char = characters[item.char].m or item.char
6059                 end
6060             end
6061         end
6062         first_on = nil
6063     end
6064
6065     if d == 'r' or d == 'l' then last = d end
6066 end
6067
6068 ----- IMPLICIT, REORDER -----
6069
6070 outer = save_outer
6071 last = outer
6072
6073 local state = {}
6074 state.has_r = false
6075
6076 for q = 1, #nodes do
6077
6078     local item = nodes[q][1]
6079

```

```

6080   outer = nodes[q][3] or outer
6081
6082   local d = nodes[q][2]
6083
6084   if d == 'nsm' then d = last end           -- W1
6085   if d == 'en' then d = 'an' end
6086   local isdir = (d == 'r' or d == 'l')
6087
6088   if outer == 'l' and d == 'an' then
6089     state.san = state.san or item
6090     state.ean = item
6091   elseif state.san then
6092     head, state = insert_numeric(head, state)
6093   end
6094
6095   if outer == 'l' then
6096     if d == 'an' or d == 'r' then         -- im -> implicit
6097       if d == 'r' then state.has_r = true end
6098       state.sim = state.sim or item
6099       state.eim = item
6100     elseif d == 'l' and state.sim and state.has_r then
6101       head, state = insert_implicit(head, state, outer)
6102     elseif d == 'l' then
6103       state.sim, state.eim, state.has_r = nil, nil, false
6104     end
6105   else
6106     if d == 'an' or d == 'l' then
6107       if nodes[q][3] then -- nil except after an explicit dir
6108         state.sim = item -- so we move sim 'inside' the group
6109       else
6110         state.sim = state.sim or item
6111       end
6112       state.eim = item
6113     elseif d == 'r' and state.sim then
6114       head, state = insert_implicit(head, state, outer)
6115     elseif d == 'r' then
6116       state.sim, state.eim = nil, nil
6117     end
6118   end
6119
6120   if isdir then
6121     last = d -- Don't search back - best save now
6122   elseif d == 'on' and state.san then
6123     state.san = state.san or item
6124     state.ean = item
6125   end
6126
6127 end
6128
6129 return node.prev(head) or head
6130 end
6131 </basic>

```

14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

For the meaning of these codes, see the Unicode standard.

15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available. The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the @ sign, etc.

```
6132 ⟨*nil⟩
6133 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
6134 \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.

```
6135 \ifx\l@nil\undefined
6136 \newlanguage\l@nil
6137 \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
6138 \let\bbl@elt\relax
6139 \edef\bbl@languages{% Add it to the list of languages
6140 \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}
6141 \fi
```

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

```
6142 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil
6143 \let\captionnil\@empty
6144 \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.

```
6145 \ldf@finish{nil}
6146 ⟨/nil⟩
```

16 Support for Plain T_EX (plain.def)

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

People can have a file `locallyhyphen.tex` or whatever they like, but they mustn't diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

```

6147 (*bplain | blplain)
6148 \catcode`\{=1 % left brace is begin-group character
6149 \catcode`\}=2 % right brace is end-group character
6150 \catcode`\#=6 % hash mark is macro parameter character

```

```
6151 \openin 0 hyphen.cfg
6152 \ifeof0
6153 \else
6154   \let\input
```

```

6155 \def\input #1 {%
6156     \let\input\@
6157     \a hyphen.cfg
6158     \let\@undefined
6159 }
6160 \fi
6161 </bplain | bplain>

```

```
6162 <bplain>\a plain.tex
6163 <blplain>\a lplain.tex
```

```
6164 <bplain>\def\fmtname{babel-plain}
6165 <blplain>\def\fmtname{babel-lplain}
```

16.2 Emulating some L^AT_EX features

```

6166 \langle\langle *Emulate LaTeX \rangle\rangle \equiv
6167 % == Code for plain ==
6168 \def\@empty{}
6169 \def\loadlocalcfg#1{%
6170   \openin0#1.cfg

```

```

6171 \ifeof0
6172 \closein0
6173 \else
6174 \closein0
6175 {\immediate\write16{*****}%
6176 \immediate\write16{* Local config file #1.cfg used}%
6177 \immediate\write16{*}%
6178 }
6179 \input #1.cfg\relax
6180 \fi
6181 \@endofldf}

```

16.3 General tools

A number of \LaTeX macro's that are needed later on.

```

6182 \long\def\@firstofone#1{#1}
6183 \long\def\@firstoftwo#1#2{#1}
6184 \long\def\@secondoftwo#1#2{#2}
6185 \def\@nnil{\@nil}
6186 \def\@gobbletwo#1#2{}
6187 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6188 \def\@star@or@long#1{%
6189 \@ifstar
6190 {\let\l@ngrel@x\relax#1}%
6191 {\let\l@ngrel@x\long#1}}
6192 \let\l@ngrel@x\relax
6193 \def\@car#1#2\@nil{#1}
6194 \def\@cdr#1#2\@nil{#2}
6195 \let\@typeset@protect\relax
6196 \let\protected@edef\edef
6197 \long\def\@gobble#1{}
6198 \edef\@backslashchar{\expandafter\@gobble\string\}
6199 \def\strip@prefix#1>{}
6200 \def\g@addto@macro#1#2{%
6201 \toks@\expandafter{#1#2}%
6202 \xdef#1{\the\toks@}}
6203 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6204 \def\@nameuse#1{\csname #1\endcsname}
6205 \def\@ifundefined#1{%
6206 \expandafter\ifx\csname#1\endcsname\relax
6207 \expandafter\@firstoftwo
6208 \else
6209 \expandafter\@secondoftwo
6210 \fi}
6211 \def\@expandtwoargs#1#2#3{%
6212 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6213 \def\zap@space#1 #2{%
6214 #1%
6215 \ifx#2\@empty\else\expandafter\zap@space\fi
6216 #2}
6217 \let\bbl@trace\@gobble

```

$\LaTeX 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

6218 \ifx\@preamblecmds\undefined
6219 \def\@preamblecmds{}
6220 \fi
6221 \def\@onlypreamble#1{%

```



```

6222 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6223   \@preamblecmds\do#1}}
6224 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

6225 \def\begindocument{%
6226   \@begindocumenthook
6227   \global\let\@begindocumenthook\@undefined
6228   \def\do##1{\global\let##1\@undefined}%
6229   \@preamblecmds
6230   \global\let\do\noexpand}

6231 \ifx\@begindocumenthook\@undefined
6232   \def\@begindocumenthook{}
6233 \fi
6234 \@onlypreamble\@begindocumenthook
6235 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

6236 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
6237 \@onlypreamble\AtEndOfPackage
6238 \def\@endoflfd{}
6239 \@onlypreamble\@endoflfd
6240 \let\bbl@afterlang\@empty
6241 \chardef\bbl@opt@hyphenmap\z@

```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

6242 \catcode`\&=\z@
6243 \ifx&\if@files\@undefined
6244   \expandafter\let\csname if@files\expandafter\endcsname
6245     \csname iffalse\endcsname
6246 \fi
6247 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

6248 \def\newcommand{\@star@or@long\new@command}
6249 \def\new@command#1{%
6250   \@testopt{\@newcommand#1}0}
6251 \def\@newcommand#1[#2]{%
6252   \@ifnextchar [{\@xargdef#1[#2]}%
6253     {\@argdef#1[#2]}}
6254 \long\def\@argdef#1[#2]#3{%
6255   \@yargdef#1\@ne{#2}{#3}}
6256 \long\def\@xargdef#1[#2][#3]#4{%
6257   \expandafter\def\expandafter#1\expandafter{%
6258     \expandafter\@protected@testopt\expandafter #1%
6259     \csname\string#1\expandafter\endcsname{#3}}%
6260   \expandafter\@yargdef \csname\string#1\endcsname
6261     \tw@{#2}{#4}}
6262 \long\def\@yargdef#1#2#3{%
6263   \@tempcnta#3\relax
6264   \advance \@tempcnta \@ne
6265   \let\@hash@\relax
6266   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6267   \@tempcntb #2%

```

```

6268 \@whilenum\@tempcntb <\@tempcnta
6269 \do{%
6270 \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
6271 \advance\@tempcntb \@ne}%
6272 \let\@hash@##%
6273 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6274 \def\providecommand{\@star@or@long\provide@command}
6275 \def\provide@command#1{%
6276 \begingroup
6277 \escapechar\m@ne\xdef\@gtempa{\string#1}%
6278 \endgroup
6279 \expandafter\ifundefined\@gtempa
6280 {\def\reserved@a{\new@command#1}}%
6281 {\let\reserved@a\relax
6282 \def\reserved@a{\new@command\reserved@a}}%
6283 \reserved@a}%
6284 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6285 \def\declare@robustcommand#1{%
6286 \edef\reserved@a{\string#1}%
6287 \def\reserved@b{#1}%
6288 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6289 \edef#1{%
6290 \ifx\reserved@a\reserved@b
6291 \noexpand\x@protect
6292 \noexpand#1%
6293 \fi
6294 \noexpand\protect
6295 \expandafter\noexpand\csname
6296 \expandafter\@gobble\string#1 \endcsname
6297 }%
6298 \expandafter\new@command\csname
6299 \expandafter\@gobble\string#1 \endcsname
6300 }
6301 \def\x@protect#1{%
6302 \ifx\protect\@typeset@protect\else
6303 \@x@protect#1%
6304 \fi
6305 }
6306 \catcode`\&=\z@ % Trick to hide conditionals
6307 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

6308 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6309 \catcode`\&=4
6310 \ifx\in@\@undefined
6311 \def\in@#1#2{%
6312 \def\in@@##1##2##3\in@{%
6313 \ifx\in@@##2\in@false\else\in@true\fi}%
6314 \in@@##1\in@\in@@}
6315 \else
6316 \let\bbl@tempa\@empty
6317 \fi
6318 \bbl@tempa

```

\LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or

false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain \TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```
6319 \def\@ifpackagewith#1#2#3#4{#3}
```

The \LaTeX macro \@ifl@aded checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```
6320 \def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands \newcommand and \providecommand exist with some sensible definition. They are not fully equivalent to their \LaTeX 2_ϵ versions; just enough to make things work in plain \TeX environments.

```
6321 \ifx\@tempcnta\@undefined
6322   \csname newcount\endcsname\@tempcnta\relax
6323 \fi
6324 \ifx\@tempcntb\@undefined
6325   \csname newcount\endcsname\@tempcntb\relax
6326 \fi
```

To prevent wasting two counters in \LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

```
6327 \ifx\bye\@undefined
6328   \advance\count10 by -2\relax
6329 \fi
6330 \ifx\@ifnextchar\@undefined
6331   \def\@ifnextchar#1#2#3{%
6332     \let\reserved@d=#1%
6333     \def\reserved@a{#2}\def\reserved@b{#3}%
6334     \futurelet\@let@token\@ifnch}
6335   \def\@ifnch{%
6336     \ifx\@let@token\@sptoken
6337       \let\reserved@c\@xifnch
6338     \else
6339       \ifx\@let@token\reserved@d
6340         \let\reserved@c\reserved@a
6341       \else
6342         \let\reserved@c\reserved@b
6343       \fi
6344     \fi
6345     \reserved@c}
6346   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6347   \def\{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6348 \fi
6349 \def\@testopt#1#2{%
6350   \@ifnextchar[#{#1}{#1[#2]}}
6351 \def\@protected@testopt#1{%
6352   \ifx\protect\@typeset@protect
6353     \expandafter\@testopt
6354   \else
6355     \@x@protect#1%
6356   \fi}
6357 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6358   #2\relax}\fi}
6359 \long\def\@iwhilenum#1{\ifnum #1\relax\expandafter\@iwhilenum
6360   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```
6361 \def\DeclareTextCommand{%
6362   \@dec@text@cmd\providecommand
6363 }
6364 \def\ProvideTextCommand{%
6365   \@dec@text@cmd\providecommand
6366 }
6367 \def\DeclareTextSymbol#1#2#3{%
6368   \@dec@text@cmd\chardef#1{#2}#3\relax
6369 }
6370 \def\@dec@text@cmd#1#2#3{%
6371   \expandafter\def\expandafter#2%
6372     \expandafter{%
6373       \csname#3-cmd\expandafter\endcsname
6374       \expandafter#2%
6375       \csname#3\string#2\endcsname
6376     }%
6377 %   \let\@ifdefinable\@rc@ifdefinable
6378   \expandafter#1\csname#3\string#2\endcsname
6379 }
6380 \def\@current@cmd#1{%
6381   \ifx\protect\@typeset@protect\else
6382     \noexpand#1\expandafter\@gobble
6383   \fi
6384 }
6385 \def\@changed@cmd#1#2{%
6386   \ifx\protect\@typeset@protect
6387     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6388       \expandafter\ifx\csname ?\string#1\endcsname\relax
6389         \expandafter\def\csname ?\string#1\endcsname{%
6390           \@changed@x@err{#1}%
6391         }%
6392       \fi
6393       \global\expandafter\let
6394         \csname\cf@encoding\string#1\expandafter\endcsname
6395         \csname ?\string#1\endcsname
6396     \fi
6397     \csname\cf@encoding\string#1%
6398       \expandafter\endcsname
6399   \else
6400     \noexpand#1%
6401   \fi
6402 }
6403 \def\@changed@x@err#1{%
6404   \errhelp{Your command will be ignored, type <return> to proceed}%
6405   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6406 \def\DeclareTextCommandDefault#1{%
6407   \DeclareTextCommand#1?%
6408 }
6409 \def\ProvideTextCommandDefault#1{%
6410   \ProvideTextCommand#1?%
6411 }
6412 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6413 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6414 \def\DeclareTextAccent#1#2#3{%
6415   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6416 }
```

```

6417 \def\DeclareTextCompositeCommand#1#2#3#4{%
6418   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6419   \edef\reserved@b{\string##1}%
6420   \edef\reserved@c{%
6421     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6422   \ifx\reserved@b\reserved@c
6423     \expandafter\expandafter\expandafter\ifx
6424       \expandafter\@car\reserved@a\relax\relax\@nil
6425       \@text@composite
6426   \else
6427     \edef\reserved@b##1{%
6428       \def\expandafter\noexpand
6429         \csname#2\string#1\endcsname####1{%
6430         \noexpand\@text@composite
6431         \expandafter\noexpand\csname#2\string#1\endcsname
6432         ####1\noexpand\@empty\noexpand\@text@composite
6433         {##1}%
6434       }%
6435     }%
6436     \expandafter\reserved@b\expandafter{\reserved@a{##1}}}%
6437   \fi
6438   \expandafter\def\csname\expandafter\string\csname
6439     #2\endcsname\string#1-\string#3\endcsname{##4}
6440 \else
6441   \errhelp{Your command will be ignored, type <return> to proceed}%
6442   \errmessage{\string\DeclareTextCompositeCommand\space used on
6443     inappropriate command \protect#1}
6444 \fi
6445 }
6446 \def\@text@composite#1#2#3\@text@composite{%
6447   \expandafter\@text@composite@x
6448     \csname\string#1-\string#2\endcsname
6449 }
6450 \def\@text@composite@x#1#2{%
6451   \ifx#1\relax
6452     #2%
6453   \else
6454     #1%
6455   \fi
6456 }
6457 %
6458 \def\@strip@args#1:#2-#3\@strip@args{#2}
6459 \def\DeclareTextComposite#1#2#3#4{%
6460   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6461   \bgroup
6462     \lcode`\@=#4%
6463     \lowercase{%
6464       \egroup
6465       \reserved@a \@
6466     }%
6467 }
6468 %
6469 \def\UseTextSymbol#1#2{#2}
6470 \def\UseTextAccent#1#2#3{}
6471 \def\@use@text@encoding#1{}
6472 \def\DeclareTextSymbolDefault#1#2{%
6473   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6474 }
6475 \def\DeclareTextAccentDefault#1#2{%

```

```

6476 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6477 }
6478 \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.

```

6479 \DeclareTextAccent{"}{OT1}{127}
6480 \DeclareTextAccent{'}{OT1}{19}
6481 \DeclareTextAccent{^}{OT1}{94}
6482 \DeclareTextAccent`{OT1}{18}
6483 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```

6484 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6485 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
6486 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
6487 \DeclareTextSymbol{\textquoteright}{OT1}{''}
6488 \DeclareTextSymbol{\i}{OT1}{16}
6489 \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`.

```

6490 \ifx\scriptsize\@undefined
6491 \let\scriptsize\sevenrm
6492 \fi
6493 % End of code for plain
6494 <</Emulate LaTeX>>

```

A proxy file:

```

6495 <*plain>
6496 \input babel.def
6497 </plain>

```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitschuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.

- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International $\mathcal{E}\TeX$ is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using $\mathcal{E}\TeX$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).