

Babel

Version 3.44
2020/05/13

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	31
1.18	Accessing language info	33
1.19	Hyphenation and line breaking	34
1.20	Selection based on BCP 47 tags	36
1.21	Selecting scripts	37
1.22	Selecting directions	38
1.23	Language attributes	42
1.24	Hooks	42
1.25	Languages supported by babel with ldf files	44
1.26	Unicode character properties in luatex	45
1.27	Tweaking some features	45
1.28	Tips, workarounds, known issues and notes	46
1.29	Current and future work	47
1.30	Tentative and experimental code	47
2	Loading languages with language.dat	47
2.1	Format	48
3	The interface between the core of babel and the language definition files	49
3.1	Guidelines for contributed languages	50
3.2	Basic macros	50
3.3	Skeleton	51
3.4	Support for active characters	53
3.5	Support for saving macro definitions	53
3.6	Support for extending macros	53
3.7	Macros common to a number of languages	53
3.8	Encoding-dependent strings	54
4	Changes	57
4.1	Changes in babel version 3.9	57
II	Source code	58

5	Identification and loading of required files	58
6	locale directory	59
7	Tools	59
7.1	Multiple languages	63
7.2	The Package File (\LaTeX , babel.sty)	64
7.3	base	66
7.4	Conditional loading of shorthands	68
7.5	Cross referencing macros	69
7.6	Marks	72
7.7	Preventing clashes with other packages	73
7.7.1	ifthen	73
7.7.2	varioref	74
7.7.3	hhline	74
7.7.4	hyperref	74
7.7.5	fancyhdr	75
7.8	Encoding and fonts	75
7.9	Basic bidi support	77
7.10	Local Language Configuration	82
8	The kernel of Babel (babel.def, common)	86
8.1	Tools	86
9	Multiple languages	87
9.1	Selecting the language	89
9.2	Errors	98
9.3	Hooks	101
9.4	Setting up language files	103
9.5	Shorthands	105
9.6	Language attributes	114
9.7	Support for saving macro definitions	116
9.8	Short tags	117
9.9	Hyphens	117
9.10	Multiencoding strings	119
9.11	Macros common to a number of languages	125
9.12	Making glyphs available	125
9.12.1	Quotation marks	125
9.12.2	Letters	127
9.12.3	Shorthands for quotation marks	128
9.12.4	Umlauts and tremas	129
9.13	Layout	130
9.14	Load engine specific macros	131
9.15	Creating and modifying languages	131
10	Adjusting the Babel bahavior	146
11	Loading hyphenation patterns	148
12	Font handling with fontspec	153

13	Hooks for XeTeX and LuaTeX	157
13.1	XeTeX	157
13.2	Layout	159
13.3	LuaTeX	160
13.4	Southeast Asian scripts	166
13.5	CJK line breaking	170
13.6	Automatic fonts and ids switching	170
13.7	Layout	181
13.8	Auto bidi with basic and basic-r	183
14	Data for CJK	194
15	The ‘nil’ language	194
16	Support for Plain T_EX (plain.def)	195
16.1	Not renaming hyphen.tex	195
16.2	Emulating some L ^A T _E X features	196
16.3	General tools	196
16.4	Encoding related macros	200
17	Acknowledgements	203

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’	27
Package babel Info: The following fonts are not babel standard families	27

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 \LaTeX (ie, `\usepackage{<language>}`) is deprecated and you will get the error:²

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

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

```
! Package babel Error: Unknown language `#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` `{\language}`

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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 `bidir` 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 `bidir` 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` $\{\langle shorthands-list \rangle\}$

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

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

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

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

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

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>\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, L^AT_EX 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. 1.22.
- layout=** **New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.22.

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 `\ldf@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.

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 follows:

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’. It is advisable to set explicitly the script to either deva or dev2, eg:

```

\newfontscript{Devanagari}{deva}

```

Other Indic scripts are still under development in the default luatex renderer, but should work with the option `Renderer=Harfbuzz` in `FONTSPEC`. They also work with xetex, although fine tuning the font behaviour 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 lualatex also applies here. Some quick patterns could help, with something similar to:

```

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

```

East Asia scripts Settings for either Simplified or Traditional should work out of the box, with basic line breaking. Although for a few words and shorts 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}

```

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}	ebu	Embu
agq	Aghem	ee	Ewe
ak	Akan	el	Greek ^{ul}
am	Amharic ^{ul}	el-polyton	Polytonic Greek ^{ul}
ar	Arabic ^{ul}	en-AU	English ^{ul}
ar-DZ	Arabic ^{ul}	en-CA	English ^{ul}
ar-MA	Arabic ^{ul}	en-GB	English ^{ul}
ar-SY	Arabic ^{ul}	en-NZ	English ^{ul}
as	Assamese	en-US	English ^{ul}
asa	Asu	en	English ^{ul}
ast	Asturian ^{ul}	eo	Esperanto ^{ul}
az-Cyrl	Azerbaijani	es-MX	Spanish ^{ul}
az-Latn	Azerbaijani	es	Spanish ^{ul}
az	Azerbaijani ^{ul}	et	Estonian ^{ul}
bas	Basaa	eu	Basque ^{ul}
be	Belarusian ^{ul}	ewo	Ewondo
bem	Bemba	fa	Persian ^{ul}
bez	Bena	ff	Fulah
bg	Bulgarian ^{ul}	fi	Finnish ^{ul}
bm	Bambara	fil	Filipino
bn	Bangla ^{ul}	fo	Faroese
bo	Tibetan ^u	fr	French ^{ul}
brx	Bodo	fr-BE	French ^{ul}
bs-Cyrl	Bosnian	fr-CA	French ^{ul}
bs-Latn	Bosnian ^{ul}	fr-CH	French ^{ul}
bs	Bosnian ^{ul}	fr-LU	French ^{ul}
ca	Catalan ^{ul}	fur	Friulian ^{ul}
ce	Chechen	fy	Western Frisian
cgg	Chiga	ga	Irish ^{ul}
chr	Cherokee	gd	Scottish Gaelic ^{ul}
ckb	Central Kurdish	gl	Galician ^{ul}
cop	Coptic	grc	Ancient Greek ^{ul}
cs	Czech ^{ul}	gsw	Swiss German
cu	Church Slavic	gu	Gujarati
cu-Cyrs	Church Slavic	guz	Gusii
cu-Glag	Church Slavic	gv	Manx
cy	Welsh ^{ul}	ha-GH	Hausa
da	Danish ^{ul}	ha-NE	Hausa ^l
dav	Taita	ha	Hausa
de-AT	German ^{ul}	haw	Hawaiian
de-CH	German ^{ul}	he	Hebrew ^{ul}
de	German ^{ul}	hi	Hindi ^u
dje	Zarma	hr	Croatian ^{ul}
dsb	Lower Sorbian ^{ul}	hsb	Upper Sorbian ^{ul}
dua	Duala	hu	Hungarian ^{ul}
dyo	Jola-Fonyi	hy	Armenian ^u
dz	Dzongkha	ia	Interlingua ^{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
kln	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}
ksb	Shambala	pt-PT	Portuguese ^{ul}
ksf	Bafia	pt	Portuguese ^{ul}
ksh	Colognian	qu	Quechua
kw	Cornish	rm	Romansh ^{ul}
ky	Kyrgyz	rn	Rundi
lag	Langi	ro	Romanian ^{ul}
lb	Luxembourgish	rof	Rombo
lg	Ganda	ru	Russian ^{ul}
lkt	Lakota	rw	Kinyarwanda
ln	Lingala	rwk	Rwa
lo	Lao ^{ul}	sa-Beng	Sanskrit
lrc	Northern Luri	sa-Deva	Sanskrit
lt	Lithuanian ^{ul}	sa-Gujr	Sanskrit
lu	Luba-Katanga	sa-Knda	Sanskrit
luo	Luo	sa-Mlym	Sanskrit
luy	Luyia	sa-Telu	Sanskrit
lv	Latvian ^{ul}	sa	Sanskrit
mas	Masai	sah	Sakha
mer	Meru	saq	Samburu
mfe	Morisyen	sbp	Sangu
mg	Malagasy	se	Northern Sami ^{ul}
mgf	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian ^{ul}	sg	Sango
ml	Malayalam ^{ul}	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi ^{ul}	shi	Tachelhit
ms-BN	Malay ^l	si	Sinhala
ms-SG	Malay ^l	sk	Slovak ^{ul}
ms	Malay ^{ul}	sl	Slovenian ^{ul}

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

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

aghem	austrian
akan	azerbaijani-cyrillic
albanian	azerbaijani-cyrl
american	azerbaijani-latin
amharic	azerbaijani-latn
ancientgreek	azerbaijani
arabic	bafia
arabic-algeria	bambara
arabic-DZ	basaa
arabic-morocco	basque
arabic-MA	belarusian
arabic-syria	bemba
arabic-SY	bena
armenian	bengali
assamese	bodo
asturian	bosnian-cyrillic
asu	bosnian-cyrl
australian	bosnian-latin

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
chinese-simplified-hongkongsarchina	ganda
chinese-simplified-macausarchina	georgian
chinese-simplified-singapore	german-at
chinese-simplified	german-austria
chinese-traditional-hongkongsarchina	german-ch
chinese-traditional-macausarchina	german-switzerland
chinese-traditional	german
chinese	greek
churchslavic	gujarati
churchslavic-cyrs	gusii
churchslavic-oldcyrillic ¹³	hausa-gh
churchsslavic-glag	hausa-ghana
churchsslavic-glagolitic	hausa-ne
colognian	hausa-niger
cornish	hausa
croatian	hawaiian
czech	hebrew
danish	hindi
duala	hungarian
dutch	icelandic
dzongkha	igbo
embu	inarisami
english-au	indonesian
english-australia	interlingua
english-ca	irish
english-canada	italian
english-gb	japanese
english-newzealand	jolafonyi
english-nz	kabuverdianu
english-unitedkingdom	kabyle

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

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
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu

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

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}

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.

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

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

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

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

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

1.16 Creating a language

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

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

If the language *⟨language-name⟩* has not been loaded as class or package option and there are no *⟨options⟩*, 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`, *⟨language-name⟩* 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\maylangchaptername{..}
(babel)                Reported on input line 18.
```

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

EXAMPLE If you need a language named arhinish:

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

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 \localdate, 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 T_EX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with \babelpatterns, as for example:

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

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

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

script= $\langle\text{script-name}\rangle$

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= $\langle\text{language-name}\rangle$

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

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

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`.

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= $\langle\text{base}\rangle$ $\langle\text{shrink}\rangle$ $\langle\text{stretch}\rangle$

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= $\langle\text{penalty}\rangle$

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 `\defineshortand` 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 `\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):

- `\localnumeral{<style>}{<number>}`, like `\localnumeral{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`

Arabic `abjad`, `maghrebi.abjad`

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

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
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

1.18 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 macros 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 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 language 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).

\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 פרק.

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.

1.19 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

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.

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` $\{ \langle \text{hyphenrules-name} \rangle \} \{ \langle \text{lua-pattern} \rangle \} \{ \langle \text{replacement} \rangle \}$

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 `([îû])`, the replacement could be `{1|îû|íú}`, which maps *î* to *í*, and *û* 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:

```
\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.)

1.20 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: $\text{fr-Latn-FR} \rightarrow \text{fr-Latn} \rightarrow \text{fr-FR} \rightarrow \text{fr}$. Languages with the same resolved name are considered the same. Case is normalized before, so that $\text{fr-latn-fr} \rightarrow \text{fr-Latn-FR}$. 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).

`autload.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.

1.21 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

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.

encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.¹⁷

`\ensureascii` $\langle text \rangle$

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

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

The foregoing rules (which are applied “at begin document”) cover most of 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.

1.22 Selecting directions

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

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

¹⁷But still defined for backwards compatibility.

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 Αραβία)، استخدم الرومان ثلاث
    بادئات بـ "Arabia" على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
    حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[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{fuṣḥā l-ʿaṣr} (MSA) and
    \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{<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 `\parshape` in `luatex` (a \TeX primitive) which makes lists to be horizontally misplaced if they are inside a `\vbox` (like `minipage`) and the current direction is different from the main one. A workaround is to restore the main language before the box and then set the local one inside.

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.23 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).

1.24 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 T_EX parameters (#1, #2, #3), with the meaning given:

- addialect** (language name, dialect name) Used by `luababel.def` to load the patterns if not preloaded.
- patterns** (language name, language with encoding) Executed just after the `\language` has been set. The second argument has the patterns name actually selected (in the form of either `lang:ENC` or `lang`).
- hyphenation** (language name, language with encoding) Executed locally just before exceptions given in `\babelhyphenation` are actually set.
- defaultcommands** Used (locally) in `\StartBabelCommands`.
- encodedcommands** (input, font encodings) Used (locally) in `\StartBabelCommands`. Both xetex and luatex make sure the encoded text is read correctly.
- stopcommands** Used to reset the 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).

1.25 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 uppersorbian
Welsh welsh

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

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

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 .

1.26 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:

```
\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 $\backslash\text{babelprovide}$, or, if the last argument is empty, removes them. The last argument is the locale name:

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

1.27 Tweaking some features

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

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: bidi.text, bidi.mirroring, bidi.mapdigits, layout.lists, layout.tabular, linebreak.sea, linebreak.cjk. For example, you can set $\backslash\text{babeladjust}\{\text{bidi.text=off}\}$ if you are using an alternative algorithm or with large sections not requiring it. With lua $\text{h}\text{b}\text{t}\text{e}\text{x}$ you may need bidi.mirroring=off. Use with care, because these options do not deactivate other related options (like paragraph direction with bidi.text).

1.28 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 `ltxdoc` and `babel` use `\AtBeginDocument` to change some catcodes, and `babel` reloads `hline` to make sure `:` has the right one, so if you want to change the catcode of `|` it has to be done using the same method at the proper place, with

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

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

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

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.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 `\usesshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- `Babel` does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- 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.

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

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.

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

1.30 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 ,

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.

pdf_{La}TeX). 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
=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.

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.

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

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

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

The following assumptions are made:

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

Some recommendations:

- The preferred shorthand is `"`, which is not used in \LaTeX (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras\langle lang \rangle` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.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.
- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.
- Babel `ldf` files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

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

3.2 Basic macros

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

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

²⁶But not removed, for backward compatibility.

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

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

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

3.3 Skeleton

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

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

```

\lfdInit{<language>}{captions<language>}

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

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

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

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

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

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

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

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

\EndBabelCommands

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

\lfd@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.

<code>\initiate@active@char</code>	The internal macro <code>\initiate@active@char</code> is used in language definition files to instruct \TeX to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.
<code>\bbl@activate</code> <code>\bbl@deactivate</code>	The command <code>\bbl@activate</code> is used to change the way an active character expands. <code>\bbl@activate</code> ‘switches on’ the active behavior of the character. <code>\bbl@deactivate</code> lets the active character expand to its former (mostly) non-active self.
<code>\declare@shorthand</code>	The macro <code>\declare@shorthand</code> 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. <code>~</code> or <code>"a</code> ; and the code to be executed when the shorthand is encountered. (It does <i>not</i> raise an error if the shorthand character has not been “initiated”.)
<code>\bbl@add@special</code> <code>\bbl@remove@special</code>	The \TeX book states: “Plain \TeX includes a macro called <code>\dospecials</code> 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 <code>\dospecial</code> . \TeX adds another macro called <code>\@sanitize</code> representing the same character set, but without the curly braces. The macros <code>\bbl@add@special⟨char⟩</code> and <code>\bbl@remove@special⟨char⟩</code> add and remove the character <code>⟨char⟩</code> 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.

<code>\babel@save</code>	To save the current meaning of any control sequence, the macro <code>\babel@save</code> is provided. It takes one argument, <code>⟨csequence⟩</code> , the control sequence for which the meaning has to be saved.
<code>\babel@savevariable</code>	A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the <code>\</code> the primitive is considered to be a variable. The macro takes one argument, the <code>⟨variable⟩</code> . The effect of the preceding macros is to append a piece of code to the current definition of <code>\originalTeX</code> . When <code>\originalTeX</code> 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

<code>\addto</code>	The macro <code>\addto{⟨control sequence⟩}{⟨\TeX code⟩}</code> can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or <code>\relax</code>). This macro can, for instance, be used in adding instructions to a macro like <code>\extrasenglish</code> . 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 <code>etoolbox</code> , by Philipp Lehman, consider using the tools provided by this package instead of <code>\addto</code> .
---------------------	--

3.7 Macros common to a number of languages

<code>\bbl@allowhyphens</code>	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
--------------------------------	--

²⁷This mechanism was introduced by Bernd Raichle.

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

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

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

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

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with `strings=generic` (no block is taken into account except those). With `strings=encoded`, strings in those blocks are set as default (internally, ?). With `strings=encoded` strings are protected, but they are correctly expanded in `\MakeUppercase` and the like. If there is no key strings, string definitions are ignored, but `\SetCases` are still honored (in a encoded way). The `<category>` is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.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\monthiname{Jänner}

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiiname{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}
```

²⁸In future releases further categories may be added.


```

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

\EndBabelCommands

```

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

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

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

$\backslash EndBabelCommands$ Marks the end of the series of blocks.

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

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

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

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

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

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

A convenient way to define several ordered names at once. For example, to define $\backslash abmoniname$, $\backslash abmoniiname$, 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.

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

Sets globally code to be executed at $\backslash MakeUpper case$ and $\backslash MakeLower case$. The code would typically be things like $\backslash let \backslash BB \backslash bb$ and $\backslash uccode$ or $\backslash lccode$ (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 $\backslash @uc\lc list$ (eg, $\backslash bb \backslash BB \backslash cc \backslash CC$). The mandatory arguments take precedence over the optional one. This command, unlike $\backslash SetString$, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \LaTeX , we could set for Turkish:

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

```

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

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

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

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap *{<to-lower-macros>}*

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

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

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

```

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

```

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

4 Changes

4.1 Changes in babel version 3.9

Most of 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 other `language*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with `^` (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that lead to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with babel were not recognized when called as global options.

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

The 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 two parts: a generic one and a second one only for LaTeX.

babel.sty is the \TeX package, which sets options and loads language styles.

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.44>>
2 <<date=2020/05/13>>
```

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

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

This does not hurt, but should be fixed somehow.

```
3 <<(*Basic macros)>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8   {\def#1{#2}}%
9   {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@c1#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 Because the code that is used in the handling of active characters may need to look ahead,
\bbl@afterfi 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.

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}}

```

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

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

```

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\bb1@replace#1#2#3{% in #1 -> repl #2 by #3
92   \toks@{}}%
93 \def\bb1@replace@aux##1#2##2#2{%
94   \ifx\bb1@nil##2%
95     \toks@\expandafter{\the\toks@##1}%
96   \else
97     \toks@\expandafter{\the\toks@##1#3}%
98     \bb1@afterfi
99     \bb1@replace@aux##2#2%
100   \fi}%
101 \expandafter\bb1@replace@aux#1#2\bb1@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 \bb1@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bb1@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   \bb1@exp{\def\\bb1@parsedef##1\detokenize{macro:}}#2->#3\relax{%
105     \def\bb1@tempa{#1}%
106     \def\bb1@tempb{#2}%
107     \def\bb1@tempe{#3}}
108   \def\bb1@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_T_EX, 1 is luatex, and 2 is xetex. You may use the latter it 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\bbl@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>> ≡
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` To add languages to \TeX 's memory plain \TeX version 3.0 supplies `\newlanguage`, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`. For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefore `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain \TeX version 3.0.

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

```

165 <<*Define core switching macros>> ≡
166 \ifx\newlanguage\@undefined
167   \csname newcount\endcsname\last@language
168   \def\addlanguage#1{%
169     \global\advance\last@language\@ne

```



```

170 \ifnum\last@language<\@ccclvi
171 \else
172 \errmessage{No room for a new \string\language!}%
173 \fi
174 \global\chardef#1\last@language
175 \wlog{\string#1 = \string\language\the\last@language}}
176 \else
177 \countdef\last@language=19
178 \def\addlanguage{\alloc@9\language\chardef\@ccclvi}
179 \fi
180 <</Define core switching macros>>

```

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

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

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.

```

181 <{*package}>
182 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
183 \ProvidesPackage{babel}[<<date>> <<version>>] The Babel package]
184 \@ifpackagewith{babel}{debug}
185 {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
186 \let\bbl@debug\@firstofone}
187 {\providecommand\bbl@trace[1]{}%
188 \let\bbl@debug\@gobble}
189 <<Basic macros>>
190 % Temporarily repeat here the code for errors
191 \def\bbl@error#1#2{%
192 \begingroup
193 \def\{\MessageBreak}%
194 \PackageError{babel}{#1}{#2}%
195 \endgroup}
196 \def\bbl@warning#1{%
197 \begingroup
198 \def\{\MessageBreak}%
199 \PackageWarning{babel}{#1}%
200 \endgroup}
201 \def\bbl@infowarn#1{%
202 \begingroup
203 \def\{\MessageBreak}%
204 \GenericWarning
205 {(babel) \@spaces\@spaces\@spaces}%
206 {Package babel Info: #1}%

```

```

207 \endgroup}
208 \def\bbl@info#1{%
209 \begingroup
210 \def\{\MessageBreak}%
211 \PackageInfo{babel}{#1}%
212 \endgroup}
213 \def\bbl@nocaption{\protect\bbl@nocaption@i}
214 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
215 \global\@namedef{#2}{\textbf{?#1?}}}%
216 \@nameuse{#2}%
217 \bbl@warning{%
218 \@backslashchar#2 not set. Please, define\\%
219 it in the preamble with something like:\\%
220 \string\renewcommand\@backslashchar#2{..}\\%
221 Reported}}
222 \def\bbl@tentative{\protect\bbl@tentative@i}
223 \def\bbl@tentative@i#1{%
224 \bbl@warning{%
225 Some functions for '#1' are tentative.\\%
226 They might not work as expected and their behavior\\%
227 could change in the future.\\%
228 Reported}}
229 \def\@nolanerr#1{%
230 \bbl@error
231 {You haven't defined the language #1\space yet.\\%
232 Perhaps you misspelled it or your installation\\%
233 is not complete}%
234 {Your command will be ignored, type <return> to proceed}}
235 \def\@nopatterns#1{%
236 \bbl@warning
237 {No hyphenation patterns were preloaded for\\%
238 the language '#1' into the format.\\%
239 Please, configure your TeX system to add them and\\%
240 rebuild the format. Now I will use the patterns\\%
241 preloaded for \bbl@nulllanguage\space instead}}
242 % End of errors
243 \ifpackagewith{babel}{silent}
244 {\let\bbl@info\@gobble
245 \let\bbl@infowarn\@gobble
246 \let\bbl@warning\@gobble}
247 {}
248 %
249 \def\AfterBabelLanguage#1{%
250 \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.

```

251 \ifx\bbl@languages\undefined\else
252 \begingroup
253 \catcode\^^I=12
254 \@ifpackagewith{babel}{showlanguages}{%
255 \begingroup
256 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
257 \wlog{<*languages>}%
258 \bbl@languages
259 \wlog{</languages>}%
260 \endgroup}{%
261 \endgroup
262 \def\bbl@elt#1#2#3#4{%

```

```

263 \ifnum#2=\z@
264 \gdef\bbl@nulllanguage{#1}%
265 \def\bbl@elt##1##2##3##4{}%
266 \fi}%
267 \bbl@languages
268 \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 \LaTeX 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.

```

269 \bbl@trace{Defining option 'base'}
270 \@ifpackagewith{babel}{base}{%
271 \let\bbl@onlyswitch\empty
272 \let\bbl@provide@locale\relax
273 \input babel.def
274 \let\bbl@onlyswitch\undefined
275 \ifx\directlua\undefined
276 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
277 \else
278 \input luababel.def
279 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
280 \fi
281 \DeclareOption{base}{}%
282 \DeclareOption{showlanguages}{}%
283 \ProcessOptions
284 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
285 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
286 \global\let@ifl@ter@@\ifl@ter
287 \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\ifl@ter@@}%
288 \endinput}{}%
289 % \end{macrocode}
290 %
291 % \subsection{\texttt{key=value} options and other general option}
292 %
293 % The following macros extract language modifiers, and only real
294 % package options are kept in the option list. Modifiers are saved
295 % and assigned to |\BabelModifiers| at |\bbl@load@language|; when
296 % no modifiers have been given, the former is |\relax|. How
297 % modifiers are handled are left to language styles; they can use
298 % |\in@|, loop them with |\@for| or load |keyval|, for example.
299 %
300 % \begin{macrocode}
301 \bbl@trace{key=value and another general options}
302 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
303 \def\bbl@tempb#1.#2{%
304 #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
305 \def\bbl@tempd#1.#2\@nnil{%
306 \ifx\@empty#2%
307 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
308 \else
309 \in@{=}{#1}\ifin@
310 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
311 \else

```

```

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

```

319 \DeclareOption{KeepShorthandsActive}{}
320 \DeclareOption{activeacute}{}
321 \DeclareOption{activegrave}{}
322 \DeclareOption{debug}{}
323 \DeclareOption{noconfigs}{}
324 \DeclareOption{showlanguages}{}
325 \DeclareOption{silent}{}
326 \DeclareOption{mono}{}
327 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
328 % Don't use. Experimental. TODO.
329 \newif\ifbbl@single
330 \DeclareOption{selectors=off}{\bbl@singletrue}
331 <<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 have 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.

```

332 \let\bbl@opt@shorthands\@nnil
333 \let\bbl@opt@config\@nnil
334 \let\bbl@opt@main\@nnil
335 \let\bbl@opt@headfoot\@nnil
336 \let\bbl@opt@layout\@nnil

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

378 \edef\babel@opt@shorthands{%
379 \expandafter\babel@sh@string\babel@opt@shorthands\@empty}%

```

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

```

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

```

385 \ifx\babel@opt@headfoot\@nnil\else
386 \g@addto@macro\@resetactivechars{%

```

```

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

```

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

```

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

```

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

```

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

```

411 \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.

```

412 <<*More package options>> ≡
413 \DeclareOption{safe=none}{\let\bbl@opt@safe\empty}
414 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
415 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
416 <</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.

```

417 \bbl@trace{Cross referencing macros}
418 \ifx\bbl@opt@safe\empty\else
419   \def\@newl@bel#1#2#3{%
420     {\@safe@activetrue
421       \bbl@ifunset{#1@#2}%
422       \relax
423       {\gdef\@multiplelabels{%
424         \latex@warning@no@line{There were multiply-defined labels}}}%
425       \latex@warning@no@line{Label `#2' multiply defined}}}%
426   \global\@namedef{#1@#2}{#3}}

```

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

```

427 \CheckCommand*\@testdef[3]{%
428   \def\reserved@a{#3}%
429   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
430   \else
431     \@tempwattrue
432   \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.

```

433 \def\@testdef#1#2#3{% TODO. With @samestring?
434   \@safe@activetrue
435   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
436   \def\bbl@tempb{#3}%
437   \@safe@activetrue
438   \ifx\bbl@tempa\relax
439   \else
440     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
441   \fi
442   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
443   \ifx\bbl@tempa\bbl@tempb
444   \else
445     \@tempwattrue
446   \fi}
447 \fi

```

`\ref` `\pageref` 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.

```

448 \bbl@xin@{R}\bbl@opt@safe
449 \ifin@
450   \bbl@redefineroobust\ref#1{%
451     \@safe@activetrue\org@ref{#1}\@safe@activetrue}
452   \bbl@redefineroobust\pageref#1{%
453     \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
454   \else
455     \let\org@ref\ref
456     \let\org@pageref\pageref
457   \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.

```
458 \bbl@xin@{B}\bbl@opt@safe
459 \ifin@
460 \bbl@redefine\@citex[#1]#2{%
461   \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
462   \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.

```
463 \AtBeginDocument{%
464   \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.)

```
465   \def\@citex[#1][#2]#3{%
466     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
467     \org@@citex[#1][#2]{\@tempa}}%
468   }{}}
```

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

```
469 \AtBeginDocument{%
470   \ifpackageloaded{cite}{%
471     \def\@citex[#1]#2{%
472       \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
473     }{}}
```

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

```
474 \bbl@redefine\nocite#1{%
475   \@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.

```
476 \bbl@redefine\bibcite{%
477   \bbl@cite@choice
478   \bibcite}
```

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

```
479 \def\bbl@bibcite#1#2{%
480   \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.

```

481 \def\bbl@cite@choice{%
482   \global\let\bibcite\bbl@bibcite
483   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
484   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
485   \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.

```

486 \AtBeginDocument{\bbl@cite@choice}

```

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

```

487 \bbl@redefine\@bibitem#1{%
488   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
489 \else
490   \let\org@nocite\nocite
491   \let\org@@citex\@citex
492   \let\org@bibcite\bibcite
493   \let\org@@bibitem\@bibitem
494 \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.

```

495 \bbl@trace{Marks}
496 \IfBabelLayout{sectioning}
497   {\ifx\bbl@opt@headfoot\@nnil
498     \g@addto@macro\@resetactivechars{%
499       \set@typeset@protect
500       \expandafter\select@language@x\expandafter{\bbl@main@language}%
501       \let\protect\noexpand
502       \edef\thepage{% TODO. Only with bidi. See also above
503         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}}}%
504   \fi}
505   {\ifbbl@single\else
506     \bbl@ifunset{markright}{\bbl@redefine\bbl@redefineroobust
507       \markright#1{%
508         \bbl@ifblank{#1}%
509         {\org@markright}}}%
510     {\toks@{#1}%
511       \bbl@exp{%
512         \org@markright{\protect\foreignlanguage{\language}%
513           {\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, \LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

514 \ifx\@mkboth\markboth
515   \def\bbl@tempc{\let\@mkboth\markboth}
516 \else
517   \def\bbl@tempc{}
518 \fi
519 \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
520 \markboth#1#2}%
521   \protected@edef\bbl@tempb##1{%
522     \protect\foreignlanguage
523     {\language}\protect\bbl@restore@actives##1}}%
524   \bbl@ifblank{#1}%
525     {\toks@{}}%
526     {\toks@\expandafter{\bbl@tempb{#1}}}%
527   \bbl@ifblank{#2}%
528     {\@temptokena{}}%
529     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
530   \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}
531   \bbl@tempc
532 \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.

```

533 \bbl@trace{Preventing clashes with other packages}
534 \bbl@xin@{R}\bbl@opt@safe
535 \ifin@
536   \AtBeginDocument{%
537     \@ifpackageloaded{ifthen}{%
538       \bbl@redefine@long\ifthenelse#1#2#3{%
539         \let\bbl@temp@pref\pageref
540         \let\pageref\org@pageref
541         \let\bbl@temp@ref\ref
542         \let\ref\org@ref
543         \@safe@activestrue
544         \org@ifthenelse{#1}%
545         {\let\pageref\bbl@temp@pref
546          \let\ref\bbl@temp@ref
547          \@safe@activesfalse

```

```

548         #2}%
549         {\let\pageref\bbl@temp@pref
550         \let\ref\bbl@temp@ref
551         \@safe@activesfalse
552         #3}%
553     }%
554 }{}%
555 }

```

7.7.2 varioref

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

```

556 \AtBeginDocument{%
557   \ifpackageloaded{varioref}{%
558     \bbl@redefine\@vpageref#1[#2]#3{%
559       \@safe@activetrue
560       \org@@vpageref{#1}[#2]{#3}%
561       \@safe@activesfalse}%
562     \bbl@redefine\vrefpagenum#1#2{%
563       \@safe@activetrue
564       \org\vrefpagenum{#1}{#2}%
565       \@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.

```

566   \expandafter\def\csname Ref \endcsname#1{%
567     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
568   }{}%
569 }
570 \fi

```

7.7.3 hhline

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

```

571 \AtEndOfPackage{%
572   \AtBeginDocument{%
573     \ifpackageloaded{hhline}%
574     {\expandafter\ifx\csname normal@char\string\endcsname\relax
575       \else
576         \makeatletter
577         \def\@currname{hhline}\input{hhline.sty}\makeatother
578         \fi}%
579     {}}

```

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?

```
580 \AtBeginDocument{%
581   \ifx\pdfstringdefDisableCommands\undefined\else
582     \pdfstringdefDisableCommands{\languageshorthands{system}}%
583   \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.

```
584 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
585   \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 .

```
586 \def\substitutefontfamily#1#2#3{%
587   \lowercase{\immediate\openout15=#1#2.fd\relax}%
588   \immediate\write15{%
589     \string\ProvidesFile{#1#2.fd}%
590     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
591     \space generated font description file]^{}
592     \string\DeclareFontFamily{#1}{#2}{^}{}
593     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^}{}
594     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^}{}
595     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^}{}
596     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^}{}
597     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^}{}
598     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^}{}
599     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^}{}
600     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^}{}
601   }%
602   \closeout15
603 }
604 \@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`

```
605 \bbl@trace{Encoding and fonts}
606 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
607 \newcommand\BabelNonText{TS1,T3,TS3}
608 \let\org@TeX\TeX
609 \let\org@LaTeX\LaTeX
610 \let\ensureascii\@firstofone
```

```

611 \AtBeginDocument{%
612   \in@false
613   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
614     \ifin@else
615       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
616     \fi}%
617   \ifin@ % if a text non-ascii has been loaded
618     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
619     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
620     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
621     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
622     \def\bbl@tempc#1ENC.DEF#2\@@{%
623       \ifx\@empty#2\else
624         \bbl@ifunset{T#1}%
625         {}%
626         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
627       \ifin@
628         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
629         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
630       \else
631         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
632       \fi}%
633     \fi}%
634   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
635   \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
636   \ifin@else
637     \edef\ensureascii#1{{%
638       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
639   \fi
640 \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.

```

641 \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.

```

642 \AtBeginDocument{%
643   \ifpackageloaded{fontspec}%
644     {\xdef\latinencoding{%
645       \ifx\UTFencname\@undefined
646         EU\ifcase\bbl@engine\or2\or1\fi
647       \else
648         \UTFencname
649       \fi}}%
650     {\gdef\latinencoding{OT1}%
651       \ifx\cf@encoding\bbl@t@one
652         \xdef\latinencoding{\bbl@t@one}%
653       \else
654         \ifx\@fontenc@load@list\@undefined
655           \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%

```

```

656     \else
657       \def\@elt#1{,#1,}%
658       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
659       \let\@elt\relax
660       \bbl@xin@{,T1,}\bbl@tempa
661       \ifin@
662         \xdef\latinencoding{\bbl@t@one}%
663       \fi
664     \fi
665   \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.

```

666 \DeclareRobustCommand{\latintext}{%
667   \fontencoding{\latinencoding}\selectfont
668   \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.

```

669 \ifx\@undefined\DeclareTextFontCommand
670   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
671 \else
672   \DeclareTextFontCommand{\textlatin}{\latintext}
673 \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 `Lua \TeX -ja` 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.

```

674 \ifodd\bbl@engine
675   \def\bbl@activate@preotf{%

```

```

676 \let\bbl@activate@preotf\relax % only once
677 \directlua{
678   Babel = Babel or {}
679   %
680   function Babel.pre_otfload_v(head)
681     if Babel.numbers and Babel.digits_mapped then
682       head = Babel.numbers(head)
683     end
684     if Babel.bidi_enabled then
685       head = Babel.bidi(head, false, dir)
686     end
687     return head
688   end
689   %
690   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
691     if Babel.numbers and Babel.digits_mapped then
692       head = Babel.numbers(head)
693     end
694     if Babel.bidi_enabled then
695       head = Babel.bidi(head, false, dir)
696     end
697     return head
698   end
699   %
700   luatexbase.add_to_callback('pre_linebreak_filter',
701     Babel.pre_otfload_v,
702     'Babel.pre_otfload_v',
703     luatexbase.priority_in_callback('pre_linebreak_filter',
704       'luaotfload.node_processor') or nil)
705   %
706   luatexbase.add_to_callback('hpack_filter',
707     Babel.pre_otfload_h,
708     'Babel.pre_otfload_h',
709     luatexbase.priority_in_callback('hpack_filter',
710       'luaotfload.node_processor') or nil)
711 }
712 \fi

```

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

```

713 \bbl@trace{Loading basic (internal) bidi support}
714 \ifodd\bbl@engine
715 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
716   \let\bbl@beforeforeign\leavevmode
717   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
718   \RequirePackage{luatexbase}
719   \bbl@activate@preotf
720   \directlua{
721     require('babel-data-bidi.lua')
722     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
723       require('babel-bidi-basic.lua')
724     \or
725       require('babel-bidi-basic-r.lua')
726     \fi}
727   % TODO - to locale_props, not as separate attribute
728   \newattribute\bbl@attr@dir
729   % TODO. I don't like it, hackish:
730   \bbl@exp{\output{\bodydir\pagedir\the\output}}
731   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}

```

```

732 \fi\fi
733 \else
734 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
735 \bbl@error
736 {The bidi method 'basic' is available only in\%
737 luatex. I'll continue with 'bidi=default', so\%
738 expect wrong results}%
739 {See the manual for further details.}%
740 \let\bbl@beforeforeign\leavevmode
741 \AtEndOfPackage{%
742 \EnableBabelHook{babel-bidi}%
743 \bbl@xebidipar}
744 \fi\fi
745 \def\bbl@loadxebidi#1{%
746 \ifx\RTLfootnotetext\@undefined
747 \AtEndOfPackage{%
748 \EnableBabelHook{babel-bidi}%
749 \ifx\fontspec\@undefined
750 \usepackage{fontspec}% bidi needs fontspec
751 \fi
752 \usepackage#1{bidi}}%
753 \fi}
754 \ifnum\bbl@bidimode>200
755 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
756 \bbl@tentative{bidi=bidi}
757 \bbl@loadxebidi{}
758 \or
759 \bbl@tentative{bidi=bidi-r}
760 \bbl@loadxebidi{[rldocument]}
761 \or
762 \bbl@tentative{bidi=bidi-l}
763 \bbl@loadxebidi{}
764 \fi
765 \fi
766 \fi
767 \ifnum\bbl@bidimode=\@ne
768 \let\bbl@beforeforeign\leavevmode
769 \ifodd\bbl@engine
770 \newattribute\bbl@attr@dir
771 \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
772 \fi
773 \AtEndOfPackage{%
774 \EnableBabelHook{babel-bidi}%
775 \ifodd\bbl@engine\else
776 \bbl@xebidipar
777 \fi}
778 \fi

```

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

```

779 \bbl@trace{Macros to switch the text direction}
780 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
781 \def\bbl@rscripts{% TODO. Base on codes ??
782 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
783 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
784 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
785 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
786 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
787 Old South Arabian,}%

```



```

788 \def\bbl@provide@dirs#1{%
789   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
790   \ifin@
791     \global\bbl@csarg\chardef{wdir@#1}\@ne
792     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
793     \ifin@
794       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
795       \fi
796     \else
797       \global\bbl@csarg\chardef{wdir@#1}\z@
798       \fi
799     \ifodd\bbl@engine
800       \bbl@csarg\ifcase{wdir@#1}%
801         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
802         \or
803         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
804         \or
805         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
806         \fi
807       \fi}
808 \def\bbl@switchdir{%
809   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
810   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
811   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
812 \def\bbl@setdirs#1{% TODO - math
813   \ifcase\bbl@select@type % TODO - strictly, not the right test
814     \bbl@bodydir{#1}%
815     \bbl@pardir{#1}%
816   \fi
817   \bbl@texmdir{#1}}
818 % TODO. Only if \bbl@bidimode > 0?:
819 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
820 \DisableBabelHook{babel-bidi}

```

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

```

821 \ifodd\bbl@engine % luatex=1
822   \chardef\bbl@thetexmdir\z@
823   \chardef\bbl@thepardir\z@
824   \def\bbl@getluadir#1{%
825     \directlua{
826       if tex.#1dir == 'TLT' then
827         tex.sprint('0')
828       elseif tex.#1dir == 'TRT' then
829         tex.sprint('1')
830       end}}
831   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
832     \ifcase#3\relax
833       \ifcase\bbl@getluadir{#1}\relax\else
834         #2 TLT\relax
835       \fi
836     \else
837       \ifcase\bbl@getluadir{#1}\relax
838         #2 TRT\relax
839       \fi
840     \fi}
841   \def\bbl@texmdir#1{%
842     \bbl@setluadir{text}\texmdir{#1}%
843     \chardef\bbl@thetexmdir#1\relax
844     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}

```

```

845 \def\bbl@pardir#1{%
846   \bbl@setluadir{par}\pardir{#1}%
847   \chardef\bbl@thepardir#1\relax}
848 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
849 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
850 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
851 % Sadly, we have to deal with boxes in math with basic.
852 % Activated every math with the package option bidi=:
853 \def\bbl@mathboxdir{%
854   \ifcase\bbl@thetextdir\relax
855     \everyhbox{\textdir TLT\relax}%
856   \else
857     \everyhbox{\textdir TRT\relax}%
858   \fi}
859 \frozen@everymath\expandafter{%
860   \expandafter\bbl@mathboxdir\the\frozen@everymath}
861 \frozen@everydisplay\expandafter{%
862   \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
863 \else % pdftex=0, xetex=2
864   \newcount\bbl@dirlevel
865   \chardef\bbl@thetextdir\z@
866   \chardef\bbl@thepardir\z@
867   \def\bbl@textdir#1{%
868     \ifcase#1\relax
869       \chardef\bbl@thetextdir\z@
870       \bbl@textdir@i\beginL\endL
871     \else
872       \chardef\bbl@thetextdir\@ne
873       \bbl@textdir@i\beginR\endR
874     \fi}
875   \def\bbl@textdir@i#1#2{%
876     \ifhmode
877       \ifnum\currentgrouplevel>\z@
878         \ifnum\currentgrouplevel=\bbl@dirlevel
879           \bbl@error{Multiple bidi settings inside a group}%
880           {I'll insert a new group, but expect wrong results.}%
881           \bgroup\aftergroup#2\aftergroup\egroup
882         \else
883           \ifcase\currentgrouptype\or % 0 bottom
884             \aftergroup#2% 1 simple {}
885           \or
886             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
887           \or
888             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
889           \or\or\or % vbox vtop align
890           \or
891             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
892           \or\or\or\or\or\or % output math disc insert vcent mathchoice
893           \or
894             \aftergroup#2% 14 \begingroup
895           \else
896             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
897           \fi
898         \fi
899         \bbl@dirlevel\currentgrouplevel
900       \fi
901       #1%
902     \fi}
903   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}

```

```

904 \let\bbl@bodydir@gobble
905 \let\bbl@pagedir@gobble
906 \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).

```

907 \def\bbl@xebidipar{%
908   \let\bbl@xebidipar\relax
909   \TeXeTstate\@ne
910   \def\bbl@xeverypar{%
911     \ifcase\bbl@thepardir
912       \ifcase\bbl@thetextdir\else\beginR\fi
913     \else
914       {\setbox\z@\lastbox\beginR\box\z@}%
915     \fi}%
916   \let\bbl@severypar\everypar
917   \newtoks\everypar
918   \everypar=\bbl@severypar
919   \bbl@severypar{\bbl@xeverypar\the\everypar}}
920 \ifnum\bbl@bidimode>200
921   \let\bbl@textdir@i@gobbletwo
922   \let\bbl@xebidipar@empty
923   \AddBabelHook{bidi}{foreign}{%
924     \def\bbl@tempa{\def\BabelText####1}%
925     \ifcase\bbl@thetextdir
926       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
927     \else
928       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
929     \fi}
930   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
931 \fi
932 \fi

```

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

```

933 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
934 \AtBeginDocument{%
935   \ifx\pdfstringdefDisableCommands\@undefined\else
936     \ifx\pdfstringdefDisableCommands\relax\else
937       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
938     \fi
939   \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`.

```

940 \bbl@trace{Local Language Configuration}
941 \ifx\loadlocalcfg\@undefined
942   \ifpackagewith{babel}{noconfigs}%
943     {\let\loadlocalcfg@gobble}%
944     {\def\loadlocalcfg#1{%
945       \InputIfFileExists{#1.cfg}%

```

```

946      {\typeout{*****^J%
947              * Local config file #1.cfg used^J%
948              *}}}%
949      \@empty}}
950 \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?

```

951 \ifx\@unexpandable@protect\@undefined
952   \def\@unexpandable@protect{\noexpand\protect\noexpand}
953   \long\def\protected@write#1#2#3{%
954     \begingroup
955       \let\thepage\relax
956       #2%
957       \let\protect\@unexpandable@protect
958       \edef\reserved@a{\write#1{#3}}%
959       \reserved@a
960     \endgroup
961     \if@nobreak\ifvmode\nobreak\fi\fi}
962 \fi
963 %
964 % \subsection{Language options}
965 %
966 % Languages are loaded when processing the corresponding option
967 % \textit{except} if a |main| language has been set. In such a
968 % case, it is not loaded until all options has been processed.
969 % The following macro inputs the ldf file and does some additional
970 % checks (|\input| works, too, but possible errors are not caught).
971 %
972 %   \begin{macrocode}
973 \bbl@trace{Language options}
974 \let\bbl@afterlang\relax
975 \let\BabelModifiers\relax
976 \let\bbl@loaded\@empty
977 \def\bbl@load@language#1{%
978   \InputIfFileExists{#1.ldf}%
979   {\edef\bbl@loaded{\CurrentOption
980     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
981     \expandafter\let\expandafter\bbl@afterlang
982       \csname\CurrentOption.ldf-h@@k\endcsname
983     \expandafter\let\expandafter\BabelModifiers
984       \csname bbl@mod@\CurrentOption\endcsname}%
985   {\bbl@error{%
986     Unknown option '\CurrentOption'. Either you misspelled it\\%
987     or the language definition file \CurrentOption.ldf was not found}{%
988     Valid options are: shorthands=, KeepShorthandsActive,\\%
989     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
990     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.

```

991 \def\bbl@try@load@lang#1#2#3{%
992   \IfFileExists{\CurrentOption.ldf}%
993   {\bbl@load@language{\CurrentOption}}%
994   {#1\bbl@load@language{#2}#3}}
995 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
996 \DeclareOption{hebrew}{%

```

```

997 \input{rlbabel.def}%
998 \bbl@load@language{hebrew}}
999 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1000 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1001 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1002 \DeclareOption{polutonikogreek}{%
1003 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1004 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1005 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1006 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

Another way to extend the list of ‘known’ options for babel was to create the file `bblopts.cfg` in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new `.ldf` file loading the actual one. You can also set the name of the file with the package option `config=<name>`, which will load `<name>.cfg` instead.

```

1007 \ifx\bbl@opt@config\@nnil
1008 \ifpackagewith{babel}{noconfigs}{}%
1009 {\InputIfFileExists{bblopts.cfg}%
1010 {\typeout{*****^J%
1011 * Local config file bblopts.cfg used^^J%
1012 *}}}%
1013 {}}%
1014 \else
1015 \InputIfFileExists{\bbl@opt@config.cfg}%
1016 {\typeout{*****^J%
1017 * Local config file \bbl@opt@config.cfg used^^J%
1018 *}}}%
1019 {\bbl@error{%
1020 Local config file '\bbl@opt@config.cfg' not found}{%
1021 Perhaps you misspelled it.}}%
1022 \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.

```

1023 \bbl@for\bbl@tempa\bbl@language@opts{%
1024 \bbl@ifunset{ds@\bbl@tempa}%
1025 {\edef\bbl@tempb{%
1026 \noexpand\DeclareOption
1027 {\bbl@tempa}%
1028 {\noexpand\bbl@load@language{\bbl@tempa}}}%
1029 \bbl@tempb}%
1030 \@empty}

```

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

```

1031 \bbl@foreach\@classoptionslist{%
1032 \bbl@ifunset{ds@#1}%
1033 {\IfFileExists{#1.ldf}%
1034 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1035 {}}%
1036 {}%

```

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

```

1037 \ifx\bbl@opt@main\@nnil\else
1038   \expandafter
1039   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1040   \DeclareOption{\bbl@opt@main}{}
1041 \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):

```

1042 \def\AfterBabelLanguage#1{%
1043   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1044 \DeclareOption*{}
1045 \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`.

```

1046 \bbl@trace{Option 'main'}
1047 \ifx\bbl@opt@main\@nnil
1048   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1049   \let\bbl@tempc\@empty
1050   \bbl@for\bbl@tempb\bbl@tempa{%
1051     \bbl@xin@{\,\bbl@tempb,}{\,\bbl@loaded,}%
1052     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1053   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1054   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1055   \ifx\bbl@tempb\bbl@tempc\else
1056     \bbl@warning{%
1057       Last declared language option is '\bbl@tempc',\%
1058       but the last processed one was '\bbl@tempb'.\%
1059       The main language cannot be set as both a global\%
1060       and a package option. Use 'main=\bbl@tempc' as\%
1061       option. Reported}%
1062   \fi
1063 \else
1064   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1065   \ExecuteOptions{\bbl@opt@main}
1066   \DeclareOption*{}
1067   \ProcessOptions*
1068 \fi
1069 \def\AfterBabelLanguage{%
1070   \bbl@error
1071   {Too late for \string\AfterBabelLanguage}%
1072   {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.

```

1073 \ifx\bbl@main@language\undefined
1074   \bbl@info{%
1075     You haven't specified a language. I'll use 'nil'\%
1076     as the main language. Reported}
1077   \bbl@load@language{nil}
1078 \fi
1079 \</package>
1080 \<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 T_EX users might want to use some of the features of the babel system too, care has to be taken that plain T_EX 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 T_EX and L^AT_EX, some of it is for the L^AT_EX 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

```
1081 \ifx\ldf@quit\@undefined\else
1082 \endinput\fi % Same line!
1083 <<Make sure ProvidesFile is defined>>
1084 \ProvidesFile{babel.def}[<<date>> <<version>> Babel common definitions]
```

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

```
1085 \ifx\AtBeginDocument\@undefined % TODO. change test.
1086 <<Emulate LaTeX>>
1087 \def\language#1{english}%
1088 \let\bbl@opt@shorthands\@nnil
1089 \def\bbl@ifshorthand#1#2#3{#2}%
1090 \let\bbl@language@opts\@empty
1091 \ifx\babeloptionstrings\@undefined
1092 \let\bbl@opt@strings\@nnil
1093 \else
1094 \let\bbl@opt@strings\babeloptionstrings
1095 \fi
1096 \def\BabelStringsDefault{generic}
1097 \def\bbl@tempa{normal}
1098 \ifx\babeloptionmath\bbl@tempa
1099 \def\bbl@mathnormal{\noexpand\textormath}
1100 \fi
1101 \def\AfterBabelLanguage#1#2{}
1102 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1103 \let\bbl@afterlang\relax
1104 \def\bbl@opt@safe{BR}
1105 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1106 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1107 \expandafter\newif\csname ifbbl@single\endcsname
1108 \chardef\bbl@bidimode\z@
1109 \fi
```

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

```
1110 \ifx\bbl@trace\@undefined
1111 \let\LdfInit\endinput
1112 \def\ProvidesLanguage#1{\endinput}
1113 \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.

```
1114 <<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.

```
1115 \def\bbl@version{<<version>>}%
1116 \def\bbl@date{<<date>>}%
1117 \def\adddialect#1#2{%
1118   \global\chardef#1#2\relax
1119   \bbl@usehooks{adddialect}{#1}{#2}}%
1120 \begingroup
1121   \count@#1\relax
1122   \def\bbl@elt##1##2##3##4{%
1123     \ifnum\count@=##2\relax
1124       \bbl@info{\string#1 = using hyphenrules for ##1\\%
1125         (\string\language\the\count@)}%
1126       \def\bbl@elt####1####2####3####4{%
1127         \fi}%
1128       \bbl@cs{languages}%
1129     \endgroup}
```

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

```
1130 \def\bbl@fixname#1{%
1131   \begingroup
1132   \def\bbl@tempe{l@}%
1133   \edef\bbl@tempd{\noexpand\ifundefined{\noexpand\bbl@tempe#1}}%
1134   \bbl@tempd
1135     {\lowercase\expandafter{\bbl@tempd}%
1136     {\uppercase\expandafter{\bbl@tempd}%
1137     \@empty
1138     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1139     {\uppercase\expandafter{\bbl@tempd}}}%
1140     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1141     {\lowercase\expandafter{\bbl@tempd}}}%
1142     \@empty
1143   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1144   \bbl@tempd
1145   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1146 \def\bbl@iflanguage#1{%
1147   \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@bcplookup either returns the found ini or it is \relax.

```

1148 \def\bbl@bcpcase#1#2#3#4\@#5{%
1149   \ifx\@empty#3%
1150     \uppercase{\def#5{#1#2}}%
1151   \else
1152     \uppercase{\def#5{#1}}%
1153     \lowercase{\edef#5{#5#2#3#4}}%
1154   \fi}
1155 \def\bbl@bcplookup#1-#2-#3-#4\@{%
1156   \let\bbl@bcp\relax
1157   \lowercase{\def\bbl@tempa{#1}}%
1158   \ifx\@empty#2%
1159     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1160   \else\ifx\@empty#3%
1161     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1162     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1163       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1164       {}}%
1165     \ifx\bbl@bcp\relax
1166       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1167     \fi
1168   \else
1169     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1170     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1171     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1172       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1173       {}}%
1174     \ifx\bbl@bcp\relax
1175       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1176       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1177       {}}%
1178     \fi
1179     \ifx\bbl@bcp\relax
1180       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1181       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1182       {}}%
1183     \fi
1184     \ifx\bbl@bcp\relax
1185       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1186     \fi
1187   \fi\fi}
1188 \let\bbl@autoload@options\@empty
1189 \let\bbl@initoload\relax
1190 \def\bbl@provide@locale{%
1191   \ifx\babelprovide\@undefined
1192     \bbl@error{For a language to be defined on the fly 'base'\%
1193               is not enough, and the whole package must be\%
1194               loaded. Either delete the 'base' option or\%
1195               request the languages explicitly}%
1196     {See the manual for further details.}%
1197   \fi
1198 % TODO. Option to search if loaded, with \LocaleForEach
1199 \let\bbl@auxname\language % Still necessary. TODO
1200 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1201 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%

```

```

1202 \ifbbl@bcpallowed
1203 \expandafter\ifx\csname date\language\endcsname\relax
1204 \expandafter
1205 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@
1206 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1207 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1208 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1209 \expandafter\ifx\csname date\language\endcsname\relax
1210 \let\bbl@initoload\bbl@bcp
1211 \bbl@exp{\\\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1212 \let\bbl@initoload\relax
1213 \fi
1214 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1215 \fi
1216 \fi
1217 \fi
1218 \expandafter\ifx\csname date\language\endcsname\relax
1219 \IfFileExists{babel-\language.tex}%
1220 {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\language}}}%
1221 {}%
1222 \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.

```

1223 \def\iflanguage#1{%
1224 \bbl@iflanguage{#1}{%
1225 \ifnum\csname l@#1\endcsname=\language
1226 \expandafter\@firstoftwo
1227 \else
1228 \expandafter\@secondoftwo
1229 \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.

```

1230 \let\bbl@select@type\z@
1231 \edef\selectlanguage{%
1232 \noexpand\protect
1233 \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`.

```

1234 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

1235 \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.

```
1236 \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`

```
1237 \def\bbl@push@language{%
1238   \ifx\language\undefined\else
1239     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1240   \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.

```
1241 \def\bbl@pop@lang#1+##3{%
1242   \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.

```
1243 \let\bbl@ifrestoring\@secondoftwo
1244 \def\bbl@pop@language{%
1245   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1246   \let\bbl@ifrestoring\@firstoftwo
1247   \expandafter\bbl@set@language\expandafter{\language}%
1248   \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).

```
1249 \chardef\localeid\z@
1250 \def\bbl@id@last{0} % No real need for a new counter
1251 \def\bbl@id@assign{%
1252   \bbl@ifunset{bbl@id@@\language}%
1253   {\count@bbl@id@last\relax
```

```

1254 \advance\count@ \@ne
1255 \bbl@csarg\chardef{id@@\language}\count@
1256 \edef\bbl@id@last{\the\count@}%
1257 \ifcase\bbl@engine\or
1258 \directlua{
1259     Babel = Babel or {}
1260     Babel.locale_props = Babel.locale_props or {}
1261     Babel.locale_props[\bbl@id@last] = {}
1262     Babel.locale_props[\bbl@id@last].name = '\language'
1263 }%
1264 \fi}%
1265 {}%
1266 \chardef\localeid\bbl@c1{id@}}

```

The unprotected part of \selectlanguage.

```

1267 \expandafter\def\csname selectlanguage \endcsname#1{%
1268 \ifnum\bbl@hymapsel=\@cc1v\let\bbl@hymapsel\tw@\fi
1269 \bbl@push@language
1270 \aftergroup\bbl@pop@language
1271 \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.

```

1272 \def\BabelContentsFiles{toc,lof,lot}
1273 \def\bbl@set@language#1{% from selectlanguage, pop@
1274 % The old buggy way. Preserved for compatibility.
1275 \edef\language{%
1276 \ifnum\escapechar=\expandafter`\string#1\@empty
1277 \else\string#1\@empty\fi}%
1278 \ifcat\relax\noexpand#1%
1279 \expandafter\ifx\csname date\language\endcsname\relax
1280 \edef\language{#1}%
1281 \let\localename\language
1282 \else
1283 \bbl@info{Using '\string\language' instead of 'language' is\\%
1284 deprecated. If what you want is to use a\\%
1285 macro containing the actual locale, make\\%
1286 sure it does not not match any language.\\%
1287 Reported}%
1288 % I'll\\%
1289 % try to fix '\string\localename', but I cannot promise\\%
1290 % anything. Reported}%
1291 \ifx\scantokens\undefined
1292 \def\localename{??}%
1293 \else
1294 \scantokens\expandafter{\expandafter
1295 \def\expandafter\localename\expandafter{\language}}%
1296 \fi
1297 \fi
1298 \else
1299 \def\localename{#1}% This one has the correct catcodes
1300 \fi

```

```

1301 \select@language{\language}%
1302 % write to auxs
1303 \expandafter\ifx\csname date\language\endcsname\relax\else
1304   \if@filesw
1305     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1306       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1307     \fi
1308     \bbl@usehooks{write}}}%
1309   \fi
1310 \fi}
1311 %
1312 \newif\ifbbl@bcpallowed
1313 \bbl@bcpallowedfalse
1314 \def\select@language#1{% from set@, babel@aux
1315   % set hmap
1316   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1317   % set name
1318   \edef\language{#1}%
1319   \bbl@fixname\language
1320   % TODO. name@map must be here?
1321   \bbl@provide@locale
1322   \bbl@iflanguage\language{%
1323     \expandafter\ifx\csname date\language\endcsname\relax
1324       \bbl@error
1325       {Unknown language '\language'. Either you have\\%
1326        misspelled its name, it has not been installed,\\%
1327        or you requested it in a previous run. Fix its name,\\%
1328        install it or just rerun the file, respectively. In\\%
1329        some cases, you may need to remove the aux file}%
1330       {You may proceed, but expect wrong results}%
1331     \else
1332       % set type
1333       \let\bbl@select@type\z@
1334       \expandafter\bbl@switch\expandafter{\language}%
1335     \fi}}
1336 \def\babel@aux#1#2{%
1337   \select@language{#1}%
1338   \bbl@foreach\BabelContentsFiles{%
1339     \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1340 \def\babel@toc#1#2{%
1341   \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.

```

1342 \newif\ifbbl@usedategroup

```

```

1343 \def\bbl@switch#1{% from select@, foreign@
1344 % make sure there is info for the language if so requested
1345 \bbl@ensureinfo{#1}%
1346 % restore
1347 \originalTeX
1348 \expandafter\def\expandafter\originalTeX\expandafter{%
1349   \csname noextras#1\endcsname
1350   \let\originalTeX\@empty
1351   \babel@beginsave}%
1352 \bbl@usehooks{afterreset}}}%
1353 \languageshorthands{none}%
1354 % set the locale id
1355 \bbl@id@assign
1356 % switch captions, date
1357 \ifcase\bbl@select@type
1358   \ifhmode
1359     \hskip\z@skip % trick to ignore spaces
1360     \csname captions#1\endcsname\relax
1361     \csname date#1\endcsname\relax
1362     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1363   \else
1364     \csname captions#1\endcsname\relax
1365     \csname date#1\endcsname\relax
1366   \fi
1367 \else
1368   \ifhmode
1369     \hskip\z@skip % trick to ignore spaces
1370     \bbl@xin@{,captions,},{, \bbl@select@opts,}%
1371     \ifin@
1372       \csname captions#1\endcsname\relax
1373     \fi
1374     \bbl@xin@{,date,},{, \bbl@select@opts,}%
1375     \ifin@ % if \foreign... within \<lang>date
1376       \csname date#1\endcsname\relax
1377     \fi
1378     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1379   \else
1380     \bbl@xin@{,captions,},{, \bbl@select@opts,}%
1381     \ifin@
1382       \csname captions#1\endcsname\relax
1383     \fi
1384     \bbl@xin@{,date,},{, \bbl@select@opts,}%
1385     \ifin@
1386       \csname date#1\endcsname\relax
1387     \fi
1388   \fi
1389 \fi
1390 % switch extras
1391 \bbl@usehooks{beforeextras}}}%
1392 \csname extras#1\endcsname\relax
1393 \bbl@usehooks{afterextras}}}%
1394 % > babel-ensure
1395 % > babel-sh-<short>
1396 % > babel-bidi
1397 % > babel-fontspec
1398 % hyphenation - case mapping
1399 \ifcase\bbl@opt@hyphenmap\or
1400   \def\BabelLower##1##2{\lccode##1=##2\relax}%
1401   \ifnum\bbl@hymapsel>4\else

```

```

1402 \csname\language @bbl@hyphenmap\endcsname
1403 \fi
1404 \chardef\bbl@opt@hyphenmap\z@
1405 \else
1406 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1407 \csname\language @bbl@hyphenmap\endcsname
1408 \fi
1409 \fi
1410 \global\let\bbl@hymapsel\@cclv
1411 % hyphenation - patterns
1412 \bbl@patterns{#1}%
1413 % hyphenation - mins
1414 \babel@savevariable\lefthyphenmin
1415 \babel@savevariable\righthyphenmin
1416 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1417 \set@hyphenmins\tw@\thr@\relax
1418 \else
1419 \expandafter\expandafter\expandafter\set@hyphenmins
1420 \csname #1hyphenmins\endcsname\relax
1421 \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.

```

1422 \long\def\otherlanguage#1{%
1423 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
1424 \csname selectlanguage \endcsname{#1}%
1425 \ignorespaces}

```

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

```

1426 \long\def\endotherlanguage{%
1427 \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`.

```

1428 \expandafter\def\csname otherlanguage*\endcsname{%
1429 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1430 \def\bbl@otherlanguage@s[#1]#2{%
1431 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1432 \def\bbl@select@opts{#1}%
1433 \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”.

```

1434 \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.

```

1435 \providecommand\bbl@beforeforeign{}
1436 \edef\foreignlanguage{%
1437   \noexpand\protect
1438   \expandafter\noexpand\csname foreignlanguage \endcsname}
1439 \expandafter\def\csname foreignlanguage \endcsname{%
1440   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1441 \newcommand\bbl@foreign@x[3][{}]{%
1442   \begingroup
1443     \def\bbl@select@opts{#1}%
1444     \let\BabelText\@firstofone
1445     \bbl@beforeforeign
1446     \foreign@language{#2}%
1447     \bbl@usehooks{foreign}{}%
1448     \BabelText{#3}% Now in horizontal mode!
1449   \endgroup}
1450 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1451   \begingroup
1452     {\par}%
1453     \let\BabelText\@firstofone
1454     \foreign@language{#1}%
1455     \bbl@usehooks{foreign*}{}%
1456     \bbl@dirparastext
1457     \BabelText{#2}% Still in vertical mode!
1458     {\par}%
1459   \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`.

```

1460 \def\foreign@language#1{%
1461   % set name
1462   \edef\language#1}%
1463   \ifbbl@usedategroup
1464     \bbl@add\bbl@select@opts{,date,}%
1465     \bbl@usedategroupfalse
1466   \fi
1467   \bbl@fixname\language
1468   % TODO. name@map here?
1469   \bbl@provide@locale
1470   \bbl@iflanguage\language{

```



```

1471 \expandafter\ifx\csname date\language\endcsname\relax
1472 \bbl@warning % TODO - why a warning, not an error?
1473 {Unknown language `#1'. Either you have\\%
1474 misspelled its name, it has not been installed,\\%
1475 or you requested it in a previous run. Fix its name,\\%
1476 install it or just rerun the file, respectively. In\\%
1477 some cases, you may need to remove the aux file.\\%
1478 I'll proceed, but expect wrong results.\\%
1479 Reported}%
1480 \fi
1481 % set type
1482 \let\bbl@select@type\@ne
1483 \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.

```

1484 \let\bbl@hyphlist\@empty
1485 \let\bbl@hyphenation@\relax
1486 \let\bbl@pttnlist\@empty
1487 \let\bbl@patterns@\relax
1488 \let\bbl@hymapsel=\@cclv
1489 \def\bbl@patterns#1{%
1490 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1491 \csname l@#1\endcsname
1492 \edef\bbl@tempa{#1}%
1493 \else
1494 \csname l@#1:\f@encoding\endcsname
1495 \edef\bbl@tempa{#1:\f@encoding}%
1496 \fi
1497 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1498 % > luatex
1499 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1500 \begingroup
1501 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1502 \ifin@else
1503 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
1504 \hyphenation{%
1505 \bbl@hyphenation@
1506 \@ifundefined{bbl@hyphenation@#1}%
1507 \@empty
1508 {\space\csname bbl@hyphenation@#1\endcsname}}%
1509 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1510 \fi
1511 \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*.

```

1512 \def\hyphenrules#1{%

```

```

1513 \edef\bbl@tempf{#1}%
1514 \bbl@fixname\bbl@tempf
1515 \bbl@iflanguage\bbl@tempf{%
1516   \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1517   \languageshortands{none}%
1518   \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1519     \set@hyphenmins\tw@\thr@@\relax
1520   \else
1521     \expandafter\expandafter\expandafter\set@hyphenmins
1522     \csname\bbl@tempf hyphenmins\endcsname\relax
1523   \fi}}
1524 \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 `\langhyphenmins` is already defined this command has no effect.

```

1525 \def\providehyphenmins#1#2{%
1526   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1527     \@namedef{#1hyphenmins}{#2}%
1528   \fi}

```

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

```

1529 \def\set@hyphenmins#1#2{%
1530   \lefthyphenmin#1\relax
1531   \righthyphenmin#2\relax}

```

`\ProvidesLanguage` The identification code for each file is something that was introduced in \LaTeX 2_ϵ . 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.

```

1532 \ifx\ProvidesFile\@undefined
1533   \def\ProvidesLanguage#1[#2 #3 #4]{%
1534     \wlog{Language: #1 #4 #3 <#2>}%
1535   }
1536 \else
1537   \def\ProvidesLanguage#1{%
1538     \begingroup
1539     \catcode`\ 10 %
1540     \@makeother\%
1541     \@ifnextchar[%]
1542       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1543   \def\@provideslanguage#1[#2]{%
1544     \wlog{Language: #1 #2}%
1545     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1546     \endgroup}
1547 \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`.

```

1548 \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.

```

1549 \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’:

```

1550 \providecommand\setlocale{%
1551   \bbl@error
1552   {Not yet available}%
1553   {Find an armchair, sit down and wait}}
1554 \let\uselocale\setlocale
1555 \let\locale\setlocale
1556 \let\selectlocale\setlocale
1557 \let\localename\setlocale
1558 \let\textlocale\setlocale
1559 \let\textlanguage\setlocale
1560 \let\languagetext\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 $\text{\LaTeX 2}_{\epsilon}$, 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.

```

1561 \edef\bbl@nulllanguage{\string\language=0}
1562 \ifx\PackageError\@undefined % TODO. Move to Plain
1563   \def\bbl@error#1#2{%
1564     \begingroup
1565       \newlinechar=`^^J
1566       \def\{^^J(babel) }%
1567       \errhelp{#2}\errmessage{\{#1}%
1568     \endgroup}
1569   \def\bbl@warning#1{%
1570     \begingroup
1571       \newlinechar=`^^J
1572       \def\{^^J(babel) }%
1573       \message{\{#1}%
1574     \endgroup}
1575   \let\bbl@infowarn\bbl@warning
1576   \def\bbl@info#1{%
1577     \begingroup
1578       \newlinechar=`^^J
1579       \def\{^^J}%
1580       \wlog{#1}%
1581     \endgroup}
1582 \fi
1583 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1584 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1585   \global\@namedef{#2}{\textbf{?#1?}}%
1586   \@nameuse{#2}%
1587   \bbl@warning{%
1588     \@backslashchar#2 not set. Please, define\\%
1589     it in the preamble with something like:\\%

```

```

1590 \string\renewcommand\@backslashchar#2{..}\%
1591 Reported}}
1592 \def\bbl@tentative{\protect\bbl@tentative@i}
1593 \def\bbl@tentative@i#1{%
1594 \bbl@warning{%
1595 Some functions for '#1' are tentative.\%
1596 They might not work as expected and their behavior\%
1597 could change in the future.\%
1598 Reported}}
1599 \def\@nolanerr#1{%
1600 \bbl@error
1601 {You haven't defined the language #1\space yet.\%
1602 Perhaps you misspelled it or your installation\%
1603 is not complete}%
1604 {Your command will be ignored, type <return> to proceed}}
1605 \def\@nopatterns#1{%
1606 \bbl@warning
1607 {No hyphenation patterns were preloaded for\%
1608 the language '#1' into the format.\%
1609 Please, configure your TeX system to add them and\%
1610 rebuild the format. Now I will use the patterns\%
1611 preloaded for \bbl@nulllanguage\space instead}}
1612 \let\bbl@usehooks\@gobbletwo
1613 \ifx\bbl@onlyswitch\@empty\endinput\fi
1614 % Here ended switch.def

Here ended switch.def.

1615 \ifx\directlua\@undefined\else
1616 \ifx\bbl@luapatterns\@undefined
1617 \input luababel.def
1618 \fi
1619 \fi
1620 <<Basic macros>>
1621 \bbl@trace{Compatibility with language.def}
1622 \ifx\bbl@languages\@undefined
1623 \ifx\directlua\@undefined
1624 \openin1 = language.def % TODO. Remove hardcoded number
1625 \ifeof1
1626 \closein1
1627 \message{I couldn't find the file language.def}
1628 \else
1629 \closein1
1630 \begingroup
1631 \def\addlanguage#1#2#3#4#5{%
1632 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1633 \global\expandafter\let\csname l@#1\expandafter\endcsname
1634 \csname lang@#1\endcsname
1635 \fi}%
1636 \def\uselanguage#1{%
1637 \input language.def
1638 \endgroup
1639 \fi
1640 \fi
1641 \chardef\l@english\z@
1642 \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.

```

1643 \def\addto#1#2{%
1644   \ifx#1\undefined
1645     \def#1{#2}%
1646   \else
1647     \ifx#1\relax
1648       \def#1{#2}%
1649     \else
1650       {\toks@\expandafter{#1#2}%
1651        \xdef#1{the\toks@}}%
1652     \fi
1653   \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?

```

1654 \def\bbl@withactive#1#2{%
1655   \begingroup
1656   \lccode`~=`#2\relax
1657   \lowercase{\endgroup#1~}}

```

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

```

1658 \def\bbl@redefine#1{%
1659   \edef\bbl@tempa{\bbl@stripslash#1}%
1660   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1661   \expandafter\def\csname\bbl@tempa\endcsname}
1662 \@onlypreamble\bbl@redefine

```

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

```

1663 \def\bbl@redefine@long#1{%
1664   \edef\bbl@tempa{\bbl@stripslash#1}%
1665   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1666   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1667 \@onlypreamble\bbl@redefine@long

```

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

```

1668 \def\bbl@redefineroobust#1{%
1669   \edef\bbl@tempa{\bbl@stripslash#1}%
1670   \bbl@ifunset{\bbl@tempa\space}%
1671   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1672    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1673   {\bbl@exp{\let\<org@\bbl@tempa\>\<\bbl@tempa\space>}}}%
1674   \@namedef{\bbl@tempa\space}}
1675 \@onlypreamble\bbl@redefineroobust

```

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.

```

1676 \bbl@trace{Hooks}
1677 \newcommand\AddBabelHook[3][{}%
1678   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1679   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1680   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1681   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1682     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1683     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1684   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1685 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1686 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1687 \def\bbl@usehooks#1#2{%
1688   \def\bbl@elt##1{%
1689     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1690     \bbl@cs{ev@#1@}%
1691     \ifx\language@undefined\else % Test required for Plain (?)
1692       \def\bbl@elt##1{%
1693         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1694         \bbl@cl{ev@#1}%
1695       \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).

```

1696 \def\bbl@evargs{,% <- don't delete this comma
1697   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1698   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1699   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1700   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1701   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.

```

1702 \bbl@trace{Defining babelensure}
1703 \newcommand\babelensure[2][{}% TODO - revise test files
1704   \AddBabelHook{babel-ensure}{afterextras}{%
1705     \ifcase\bbl@select@type
1706       \bbl@cl{e}%
1707     \fi}%
1708   \beginngroup
1709     \let\bbl@ens@include\@empty
1710     \let\bbl@ens@exclude\@empty
1711     \def\bbl@ens@fontenc{\relax}%

```

```

1712 \def\bbl@tempb##1{%
1713   \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1714 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1715 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1716 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1717 \def\bbl@tempc{\bbl@ensure}%
1718 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1719   \expandafter{\bbl@ens@include}}%
1720 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1721   \expandafter{\bbl@ens@exclude}}%
1722 \toks@\expandafter{\bbl@tempc}%
1723 \bbl@exp{%
1724 \endgroup
1725 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1726 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1727 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1728   \ifx##1\undefined % 3.32 - Don't assume the macro exists
1729     \edef##1{\noexpand\bbl@nocaption
1730       {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}%
1731   \fi
1732   \ifx##1\@empty\else
1733     \in@{##1}{#2}%
1734     \ifin@ \else
1735       \bbl@ifunset{\bbl@ensure@\language\language}%
1736       {\bbl@exp{%
1737         \\DeclareRobustCommand\<bbl@ensure@\language\language>[1]{%
1738           \\foreignlanguage{\language\language}%
1739           {\ifx\relax#3\else
1740             \\fontencoding{#3}\\selectfont
1741             \fi
1742             #####1}}}%
1743         {}%
1744         \toks@\expandafter{##1}%
1745         \edef##1{%
1746           \bbl@csarg\noexpand{ensure@\language\language}%
1747           {\the\toks@}}%
1748       \fi
1749       \expandafter\bbl@tempb
1750     \fi}%
1751 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1752 \def\bbl@tempa##1{% elt for include list
1753   \ifx##1\@empty\else
1754     \bbl@csarg\in@{ensure@\language\language\expandafter}\expandafter{##1}%
1755     \ifin@ \else
1756       \bbl@tempb##1\@empty
1757     \fi
1758     \expandafter\bbl@tempa
1759   \fi}%
1760 \bbl@tempa#1\@empty}
1761 \def\bbl@captionslist{%
1762 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1763 \contentsname\listfigurename\listtablename\indexname\figurename
1764 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1765 \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`.

```
1766 \bbl@trace{Macros for setting language files up}
1767 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1768   \let\bbl@screset@empty
1769   \let\BabelStrings\bbl@opt@string
1770   \let\BabelOptions@empty
1771   \let\BabelLanguages\relax
1772   \ifx\originalTeX\@undefined
1773     \let\originalTeX@empty
1774   \else
1775     \originalTeX
1776   \fi}
1777 \def\LdfInit#1#2{%
1778   \chardef\atcatcode=\catcode`\@
1779   \catcode`\@=11\relax
1780   \chardef\eqcatcode=\catcode`\=
1781   \catcode`\==12\relax
1782   \expandafter\if\expandafter\@backslashchar
1783     \expandafter\@car\string#2\@nil
1784   \ifx#2\@undefined\else
1785     \ldf@quit{#1}%
1786   \fi
1787 \else
1788   \expandafter\ifx\csname#2\endcsname\relax\else
1789     \ldf@quit{#1}%
1790   \fi
1791 \fi
1792 \bbl@ldfinit}
```

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

```
1793 \def\ldf@quit#1{%
1794   \expandafter\main@language\expandafter{#1}%
1795   \catcode`\@=\atcatcode \let\atcatcode\relax
1796   \catcode`\==\eqcatcode \let\eqcatcode\relax
1797   \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.

```

1798 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1799   \bbl@afterlang
1800   \let\bbl@afterlang\relax
1801   \let\BabelModifiers\relax
1802   \let\bbl@screset\relax}%
1803 \def\ldf@finish#1{%
1804   \ifx\loadlocalcfg@undefined\else % For LaTeX 209
1805     \loadlocalcfg{#1}%
1806   \fi
1807   \bbl@afterldf{#1}%
1808   \expandafter\main@language\expandafter{#1}%
1809   \catcode`\@=\atcatcode \let\atcatcode\relax
1810   \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 .

```

1811 \@onlypreamble\LdfInit
1812 \@onlypreamble\ldf@quit
1813 \@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.

```

1814 \def\main@language#1{%
1815   \def\bbl@main@language{#1}%
1816   \let\language\bbl@main@language % TODO. Set localename
1817   \bbl@id@assign
1818   \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`.

```

1819 \def\bbl@beforestart{%
1820   \bbl@usehooks{beforestart}{}%
1821   \global\let\bbl@beforestart\relax}
1822 \AtBeginDocument{%
1823   \@nameuse{bbl@beforestart}%
1824   \if@filesw
1825     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1826   \fi
1827   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1828   \ifbbl@single % must go after the line above.
1829     \renewcommand\selectlanguage[1]{}%
1830     \renewcommand\foreignlanguage[2]{#2}%
1831     \global\let\babel@aux@gobbletwo % Also as flag
1832   \fi
1833   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1834 \def\select@language@x#1{%
1835   \ifcase\bbl@select@type

```

```

1836 \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1837 \else
1838 \select@language{#1}%
1839 \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.

```

1840 \bbl@trace{Shorhands}
1841 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1842 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1843 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1844 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1845 \begingroup
1846 \catcode`#1\active
1847 \nfss@catcodes
1848 \ifnum\catcode`#1=\active
1849 \endgroup
1850 \bbl@add\nfss@catcodes{\@makeother#1}%
1851 \else
1852 \endgroup
1853 \fi
1854 \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.

```

1855 \def\bbl@remove@special#1{%
1856 \begingroup
1857 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1858 \else\noexpand##1\noexpand##2\fi}%
1859 \def\do{\x\do}%
1860 \def\@makeother{\x\@makeother}%
1861 \edef\x{\endgroup
1862 \def\noexpand\dospecials{\dospecials}%
1863 \expandafter\ifx\cname @sanitize\endcname\relax\else
1864 \def\noexpand\@sanitize{\@sanitize}%
1865 \fi}%
1866 \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).

```

1867 \def\bbl@active@def#1#2#3#4{%
1868   \@namedef{#3#1}{%
1869     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1870       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1871     \else
1872       \bbl@afterfi\csname#2@sh@#1\endcsname
1873     \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.

```

1874 \long\@namedef{#3@arg#1}##1{%
1875   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1876     \bbl@afterelse\csname#4#1\endcsname##1%
1877   \else
1878     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1879   \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.

```

1880 \def\initiate@active@char#1{%
1881   \bbl@ifunset{active@char\string#1}%
1882   {\bbl@withactive
1883     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1884   {}

```

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`).

```

1885 \def\@initiate@active@char#1#2#3{%
1886   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1887   \ifx#1\@undefined
1888     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1889   \else
1890     \bbl@csarg\let{oridef@#2}#1%
1891     \bbl@csarg\edef{oridef@#2}{%
1892       \let\noexpand#1%
1893       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1894   \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*”).

```

1895 \ifx#1#3\relax
1896   \expandafter\let\csname normal@char#2\endcsname#3%

```

```

1897 \else
1898 \bbl@info{Making #2 an active character}%
1899 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1900 \@namedef{normal@char#2}{%
1901 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1902 \else
1903 \@namedef{normal@char#2}{#3}%
1904 \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).

```

1905 \bbl@restoreactive{#2}%
1906 \AtBeginDocument{%
1907 \catcode`#2\active
1908 \if@filesw
1909 \immediate\write\@mainaux{\catcode`\string#2\active}%
1910 \fi}%
1911 \expandafter\bbl@add@special\csname#2\endcsname
1912 \catcode`#2\active
1913 \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⟩`).

```

1914 \let\bbl@tempa\@firstoftwo
1915 \if\string^#2%
1916 \def\bbl@tempa{\noexpand\textormath}%
1917 \else
1918 \ifx\bbl@mathnormal\@undefined\else
1919 \let\bbl@tempa\bbl@mathnormal
1920 \fi
1921 \fi
1922 \expandafter\edef\csname active@char#2\endcsname{%
1923 \bbl@tempa
1924 {\noexpand\if@safe@actives
1925 \noexpand\expandafter
1926 \expandafter\noexpand\csname normal@char#2\endcsname
1927 \noexpand\else
1928 \noexpand\expandafter
1929 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1930 \noexpand\fi}%
1931 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1932 \bbl@csarg\edef{doactive#2}{%
1933 \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!).

```

1934 \bbl@csarg\edef{active@#2}{%

```

```

1935 \noexpand\active@prefix\noexpand#1%
1936 \expandafter\noexpand\csname active@char#2\endcsname}%
1937 \bbl@csarg\edef{normal@#2}{%
1938 \noexpand\active@prefix\noexpand#1%
1939 \expandafter\noexpand\csname normal@char#2\endcsname}%
1940 \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.

```

1941 \bbl@active@def#2\user@group{user@active}{language@active}%
1942 \bbl@active@def#2\language@group{language@active}{system@active}%
1943 \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.

```

1944 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1945 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1946 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1947 {\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.

```

1948 \if\string'#2%
1949 \let\prim@s\bbl@prim@s
1950 \let\active@math@prime#1%
1951 \fi
1952 \bbl@usehooks{initiateactive}{#{1}{#2}{#3}}

```

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

```

1953 <<More package options>> ≡
1954 \DeclareOption{math=active}{}
1955 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1956 <</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*.

```

1957 \ifpackagewith{babel}{KeepShorthandsActive}%
1958 {\let\bbl@restoreactive\@gobble}%
1959 {\def\bbl@restoreactive#1{%
1960 \bbl@exp{%
1961 \\\AfterBabelLanguage\\CurrentOption
1962 {\catcode`#1=\the\catcode`#1\relax}%
1963 \\\AtEndOfPackage
1964 {\catcode`#1=\the\catcode`#1\relax}}}%
1965 \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.

```

1966 \def\bbl@sh@select#1#2{%
1967   \expandafter\ifx\csname#1@sh#2@sel\endcsname\relax
1968     \bbl@afterelse\bbl@scndcs
1969   \else
1970     \bbl@afterfi\csname#1@sh#2@sel\endcsname
1971   \fi}

```

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

```

1972 \begingroup
1973 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1974 {\gdef\active@prefix#1{%
1975   \ifx\protect\@typeset@protect
1976     \else
1977       \ifx\protect\@unexpandable@protect
1978         \noexpand#1%
1979       \else
1980         \protect#1%
1981       \fi
1982       \expandafter\@gobble
1983     \fi}}
1984 {\gdef\active@prefix#1{%
1985   \ifincsname
1986     \string#1%
1987     \expandafter\@gobble
1988   \else
1989     \ifx\protect\@typeset@protect
1990     \else
1991       \ifx\protect\@unexpandable@protect
1992         \noexpand#1%
1993       \else
1994         \protect#1%
1995       \fi
1996       \expandafter\expandafter\expandafter\@gobble
1997     \fi
1998   \fi}}
1999 \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⟩`.

```

2000 \newif\if@safe@actives
2001 \@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.

```

2002 \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`

```

2003 \def\bbl@activate#1{%
2004   \bbl@withactive{\expandafter\let\expandafter}#1%
2005   \csname bbl@active@\string#1\endcsname}
2006 \def\bbl@deactivate#1{%
2007   \bbl@withactive{\expandafter\let\expandafter}#1%
2008   \csname bbl@normal@\string#1\endcsname}

```

`\bbl@firstcs` These macros are used only as a trick when declaring shorthands.

`\bbl@scndcs`

```

2009 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2010 \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.

```

2011 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2012 \def\@decl@short#1#2#3\@nil#4{%
2013   \def\bbl@tempa{#3}%
2014   \ifx\bbl@tempa\@empty
2015     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2016     \bbl@ifunset{#1@sh@\string#2@}{}%
2017     {\def\bbl@tempa{#4}%
2018      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2019      \else
2020        \bbl@info
2021        {Redefining #1 shorthand \string#2\\%
2022         in language \CurrentOption}%
2023        \fi}%
2024     \@namedef{#1@sh@\string#2@}{#4}%
2025   \else
2026     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2027     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2028     {\def\bbl@tempa{#4}%
2029      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2030      \else
2031        \bbl@info
2032        {Redefining #1 shorthand \string#2\string#3\\%
2033         in language \CurrentOption}%
2034        \fi}%
2035     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2036   \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.

```

2037 \def\textormath{%
2038   \ifmmode
2039     \expandafter\@secondoftwo
2040   \else
2041     \expandafter\@firstoftwo
2042   \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’.

`\language@group`

`\system@group`

```

2043 \def\user@group{user}
2044 \def\language@group{english} % TODO. I don't like defaults
2045 \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.

```

2046 \def\useshorthands{%
2047   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
2048 \def\bb1@usesh@s#1{%
2049   \bb1@usesh@x
2050   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
2051   {#1}}
2052 \def\bb1@usesh@x#1#2{%
2053   \bb1@ifshorthand{#2}%
2054   {\def\user@group{user}%
2055     \initiate@active@char{#2}%
2056     #1%
2057     \bb1@activate{#2}}%
2058   {\bb1@error
2059     {Cannot declare a shorthand turned off (\string#2)}
2060     {Sorry, but you cannot use shorthands which have been\\
2061       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.

```

2062 \def\user@language@group{user@\language@group}
2063 \def\bb1@set@user@generic#1#2{%
2064   \bb1@ifunset{user@generic@active#1}%
2065   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
2066     \bb1@active@def#1\user@group{user@generic@active}{language@active}%
2067     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2068       \expandafter\noexpand\csname normal@char#1\endcsname}%
2069     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
2070       \expandafter\noexpand\csname user@active#1\endcsname}}%
2071   \@empty}
2072 \newcommand\defineshorthand[3][user]{%
2073   \edef\bb1@tempa{\zap@space#1 \@empty}%
2074   \bb1@for\bb1@tempb\bb1@tempa{%
2075     \if*\expandafter\@car\bb1@tempb\@nil
2076       \edef\bb1@tempb{user@\expandafter\@gobble\bb1@tempb}%
2077       \@expandtwoargs
2078       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
2079     \fi
2080     \declare@shorthand{\bb1@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].

```

2081 \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`.

```

2082 \def\aliasshorthand#1#2{%
2083   \bbl@ifshorthand{#2}%
2084   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2085     \ifx\document\@notprerr
2086       \@notshorthand{#2}%
2087     \else
2088       \initiate@active@char{#2}%
2089       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2090         \csname active@char\string#1\endcsname
2091       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2092         \csname normal@char\string#1\endcsname
2093       \bbl@activate{#2}%
2094     \fi
2095   \fi}%
2096   {\bbl@error
2097     {Cannot declare a shorthand turned off (\string#2)}
2098     {Sorry, but you cannot use shorthands which have been\\%
2099       turned off in the package options}}}

```

`\@notshorthand`

```

2100 \def\@notshorthand#1{%
2101   \bbl@error{%
2102     The character '\string #1' should be made a shorthand character;\\%
2103     add the command \string\usesshorthands\string{#1\string} to
2104     the preamble.\\%
2105     I will ignore your instruction}%
2106   {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.

```

2107 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2108 \DeclareRobustCommand*\shorthandoff{%
2109   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2110 \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.

```

2111 \def\bbl@switch@sh#1#2{%
2112   \ifx#2\@nnil\else
2113     \bbl@ifunset{\bbl@active@\string#2}%
2114     {\bbl@error
2115       {I cannot switch '\string#2' on or off--not a shorthand}%
2116       {This character is not a shorthand. Maybe you made\\%
2117         a typing mistake? I will ignore your instruction}}%
2118     {\ifcase#1%
2119       \catcode`#212\relax
2120     \or

```

```

2121      \catcode`#2\active
2122      \or
2123      \csname bbl@oricat@\string#2\endcsname
2124      \csname bbl@oridef@\string#2\endcsname
2125      \fi}%
2126      \bbl@afterfi\bbl@switch@sh#1%
2127  \fi}

```

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

```

2128 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2129 \def\bbl@putsh#1{%
2130   \bbl@ifunset{\bbl@active@\string#1}%
2131   {\bbl@putsh@i#1\@empty\@nnil}%
2132   {\csname bbl@active@\string#1\endcsname}}
2133 \def\bbl@putsh@i#1#2\@nnil{%
2134   \csname\language\name @sh@\string#1@%
2135     \ifx\@empty#2\else\string#2@\fi\endcsname}
2136 \ifx\bbl@opt@shorthands\@nnil\else
2137   \let\bbl@s@initiate@active@char\initiate@active@char
2138   \def\initiate@active@char#1{%
2139     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2140   \let\bbl@s@switch@sh\bbl@switch@sh
2141   \def\bbl@switch@sh#1#2{%
2142     \ifx#2\@nnil\else
2143       \bbl@afterfi
2144       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2145     \fi}
2146   \let\bbl@s@activate\bbl@activate
2147   \def\bbl@activate#1{%
2148     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2149   \let\bbl@s@deactivate\bbl@deactivate
2150   \def\bbl@deactivate#1{%
2151     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2152 \fi

```

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

```

2153 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

\bbl@prim@s One of the internal macros that are involved in substituting \prime for each right quote in 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.

\bbl@pr@m@s

```

2154 \def\bbl@prim@s{%
2155   \prime\futurelet\@let@token\bbl@pr@m@s}
2156 \def\bbl@if@primes#1#2{%
2157   \ifx#1\@let@token
2158     \expandafter\@firstoftwo
2159   \else\ifx#2\@let@token
2160     \bbl@afterelse\expandafter\@firstoftwo
2161   \else
2162     \bbl@afterfi\expandafter\@secondoftwo
2163   \fi\fi}
2164 \begingroup
2165 \catcode`\^=7 \catcode`\*=\active \lccode`\*='^
2166 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"='\'
2167 \lowercase{%

```

```

2168 \gdef\bbl@pr@m@s{%
2169 \bbl@if@primes"%
2170 \pr@@s
2171 {\bbl@if@primes*^\pr@@@t\egroup}}
2172 \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).

```

2173 \initiate@active@char{~}
2174 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2175 \bbl@activate{~}

```

\OT1dpos The position of the double quote character is different for the OT1 and T1 encodings. It will
\T1dpos later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.

```

2176 \expandafter\def\csname OT1dpos\endcsname{127}
2177 \expandafter\def\csname T1dpos\endcsname{4}

```

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

```

2178 \ifx\f@encoding\@undefined
2179 \def\f@encoding{OT1}
2180 \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.

```

2181 \bbl@trace{Language attributes}
2182 \newcommand\languageattribute[2]{%
2183 \def\bbl@tempc{#1}%
2184 \bbl@fixname\bbl@tempc
2185 \bbl@iflanguage\bbl@tempc{%
2186 \bbl@vforeach{#2}{%

```

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

```

2187 \ifx\bbl@known@attribs\@undefined
2188 \in@false
2189 \else
2190 \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,}%
2191 \fi
2192 \ifin@
2193 \bbl@warning{%
2194 You have more than once selected the attribute '##1'\%
2195 for language #1. Reported}%
2196 \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.

```

2197 \bbl@exp{%
2198   \bbl@add@list\bbl@known@attrs{\bbl@tempc-##1}}%
2199 \edef\bbl@tempa{\bbl@tempc-##1}%
2200 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2201 {\csname\bbl@tempc @attr##1\endcsname}%
2202 {\@attrerr{\bbl@tempc}{##1}}%
2203 \fi}}
2204 \@onlypreamble\languageattribute

```

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

```

2205 \newcommand*{\@attrerr}[2]{%
2206   \bbl@error
2207   {The attribute #2 is unknown for language #1.}%
2208   {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}`.

```

2209 \def\bbl@declare@ttribute#1#2#3{%
2210   \bbl@xin{,#2,},{,\BabelModifiers,}%
2211   \ifin@
2212     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2213   \fi
2214   \bbl@add@list\bbl@attributes{#1-#2}%
2215   \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`'.

```

2216 \def\bbl@ifattributeset#1#2#3#4{%
2217   \ifx\bbl@known@attrs\@undefined
2218     \in@false
2219   \else
2220     \bbl@xin{,#1-#2,},{,\bbl@known@attrs,}%
2221   \fi
2222   \ifin@
2223     \bbl@afterelse#3%
2224   \else
2225     \bbl@afterfi#4%
2226   \fi
2227 }

```

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

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 `\bbl@tempa` is changed. Finally we execute `\bbl@tempa`.

```

2228 \def\bbl@ifknown@ttrib#1#2{%
2229   \let\bbl@tempa\@secondoftwo
2230   \bbl@loopx\bbl@tempb{#2}{%
2231     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
2232     \ifin@
2233       \let\bbl@tempa\@firstoftwo
2234     \else
2235       \fi}%
2236   \bbl@tempa
2237 }
```

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

```

2238 \def\bbl@clear@ttribs{%
2239   \ifx\bbl@attributes\undefined\else
2240     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2241       \expandafter\bbl@clear@ttrib\bbl@tempa.
2242     }%
2243     \let\bbl@attributes\undefined
2244   \fi}
2245 \def\bbl@clear@ttrib#1-#2.{%
2246   \expandafter\let\csname#1@attr#2\endcsname\undefined}
2247 \AtBeginDocument{\bbl@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`

```

2248 \bbl@trace{Macros for saving definitions}
2249 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```

2250 \newcount\babel@savecnt
2251 \babel@beginsave
```

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

```

2252 \def\babel@save#1{%
2253   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2254   \toks@\expandafter{\originalTeX\let#1=}
2255   \bbl@exp{%
```

³¹`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```

2256 \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}%
2257 \advance\babel@savecnt\@ne}
2258 \def\babel@savevariable#1{%
2259 \toks@\expandafter{\originalTeX #1}%
2260 \bbl@exp{\def\originalTeX{\the\toks@the#1\relax}}}

```

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

```

2261 \def\bbl@frenchspacing{%
2262 \ifnum\the\sfcode`\.=\@m
2263 \let\bbl@nonfrenchspacing\relax
2264 \else
2265 \frenchspacing
2266 \let\bbl@nonfrenchspacing\nonfrenchspacing
2267 \fi}
2268 \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.

```

2269 \bbl@trace{Short tags}
2270 \def\babeltags#1{%
2271 \edef\bbl@tempa{\zap@space#1 \@empty}%
2272 \def\bbl@tempb##1=##2\@{ }%
2273 \edef\bbl@tempc{%
2274 \noexpand\newcommand
2275 \expandafter\noexpand\csname ##1\endcsname{%
2276 \noexpand\protect
2277 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2278 \noexpand\newcommand
2279 \expandafter\noexpand\csname text##1\endcsname{%
2280 \noexpand\foreignlanguage{##2}}
2281 \bbl@tempc}%
2282 \bbl@for\bbl@tempa\bbl@tempa{%
2283 \expandafter\bbl@tempb\bbl@tempa\@{ }

```

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.

```

2284 \bbl@trace{Hyphens}
2285 \@onlypreamble\babelhyphenation
2286 \AtEndOfPackage{%
2287 \newcommand\babelhyphenation[2][\@empty]{%
2288 \ifx\bbl@hyphenation@\relax
2289 \let\bbl@hyphenation@\@empty
2290 \fi
2291 \ifx\bbl@hyphlist\@empty\else
2292 \bbl@warning{%
2293 You must not intermingle \string\selectlanguage\space and\%
2294 \string\babelhyphenation\space or some exceptions will not\%

```

```

2295         be taken into account. Reported}%
2296     \fi
2297     \ifx\@empty#1%
2298         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2299     \else
2300         \bbl@vforeach{#1}{%
2301             \def\bbl@tempa{##1}%
2302             \bbl@fixname\bbl@tempa
2303             \bbl@iflanguage\bbl@tempa{%
2304                 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2305                     \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2306                         \@empty
2307                     {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2308                     #2}}}%
2309     \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`³².

```

2310 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2311 \def\bbl@t@one{T1}
2312 \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.

```

2313 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2314 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2315 \def\bbl@hyphen{%
2316     \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2317 \def\bbl@hyphen@i#1#2{%
2318     \bbl@ifunset{bbl@hy#1#2\@empty}%
2319     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2320     {\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.

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.

```

2321 \def\bbl@usehyphen#1{%
2322     \leavevmode
2323     \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2324     \nobreak\hskip\z@skip}
2325 \def\bbl@usehyphen#1{%
2326     \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2327 \def\bbl@hyphenchar{%
2328     \ifnum\hyphenchar\font=\m@ne
2329         \babelnullhyphen
2330     \else
2331         \char\hyphenchar\font
2332     \fi}

```

³² $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

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.

```

2333 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2334 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2335 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2336 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2337 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2338 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2339 \def\bbl@hy@repeat{%
2340   \bbl@usehyphen{%
2341     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2342 \def\bbl@hy@@repeat{%
2343   \bbl@usehyphen{%
2344     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2345 \def\bbl@hy@empty{\hskip\z@skip}
2346 \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.

```

2347 \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.

```

2348 \bbl@trace{Multiencoding strings}
2349 \def\bbl@tglobal#1{\global\let#1#1}
2350 \def\bbl@recode#1{% TODO. Used only once?
2351   \@tempcnta="7F
2352   \def\bbl@tempa{%
2353     \ifnum\@tempcnta>"FF\else
2354       \catcode\@tempcnta=#1\relax
2355       \advance\@tempcnta\@ne
2356       \expandafter\bbl@tempa
2357     \fi}%
2358   \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).

```

2359 \@ifpackagewith{babel}{nocase}%
2360 {\let\bbl@patchuclc\relax}%
2361 {\def\bbl@patchuclc{%

```



```

2362 \global\let\bbl@patchuclc\relax
2363 \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2364 \gdef\bbl@uclc##1{%
2365   \let\bbl@encoded\bbl@encoded@uclc
2366   \bbl@ifunset{\language @bbl@uclc}% and resumes it
2367   {##1}%
2368   {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2369     \csname\language @bbl@uclc\endcsname}%
2370     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2371   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2372   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
2373 <<(*More package options)>> ≡
2374 \DeclareOption{nocase}{}
2375 <</More package options>>

```

The following package options control the behavior of \SetString.

```

2376 <<(*More package options)>> ≡
2377 \let\bbl@opt@strings\@nnil % accept strings=value
2378 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2379 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2380 \def\BabelStringsDefault{generic}
2381 <</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.

```

2382 \@onlypreamble\StartBabelCommands
2383 \def\StartBabelCommands{%
2384   \begingroup
2385   \bbl@recatcode{11}%
2386   <<Macros local to BabelCommands>>
2387   \def\bbl@provstring##1##2{%
2388     \providecommand##1{##2}%
2389     \bbl@tglobal##1}%
2390   \global\let\bbl@scafter\@empty
2391   \let\StartBabelCommands\bbl@startcmds
2392   \ifx\BabelLanguages\relax
2393     \let\BabelLanguages\CurrentOption
2394   \fi
2395   \begingroup
2396   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2397   \StartBabelCommands}
2398 \def\bbl@startcmds{%
2399   \ifx\bbl@screset\@nnil\else
2400     \bbl@usehooks{stopcommands}{}%
2401   \fi
2402   \endgroup
2403   \begingroup
2404   \@ifstar
2405     {\ifx\bbl@opt@strings\@nnil
2406       \let\bbl@opt@strings\BabelStringsDefault
2407     \fi
2408     \bbl@startcmds@i}%
2409   \bbl@startcmds@i}
2410 \def\bbl@startcmds@i#1#2{%
2411   \edef\bbl@L{\zap@space#1 \@empty}%
2412   \edef\bbl@G{\zap@space#2 \@empty}%

```

```

2413 \bbl@startcmds@ii}
2414 \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.

```

2415 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2416 \let\SetString\@gobbletwo
2417 \let\bbl@stringdef\@gobbletwo
2418 \let\AfterBabelCommands\@gobble
2419 \ifx\@empty#1%
2420 \def\bbl@sc@label{generic}%
2421 \def\bbl@encstring##1##2{%
2422 \ProvideTextCommandDefault##1{##2}%
2423 \bbl@tglobal##1%
2424 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2425 \let\bbl@sctest\in@true
2426 \else
2427 \let\bbl@sc@charset\space % <- zapped below
2428 \let\bbl@sc@fontenc\space % <- " "
2429 \def\bbl@tempa##1=##2\@nil{%
2430 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2431 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2432 \def\bbl@tempa##1 ##2{% space -> comma
2433 ##1%
2434 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2435 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2436 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2437 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2438 \def\bbl@encstring##1##2{%
2439 \bbl@foreach\bbl@sc@fontenc{%
2440 \bbl@ifunset{T####1}%
2441 }%
2442 {\ProvideTextCommand##1{####1}{##2}%
2443 \bbl@tglobal##1%
2444 \expandafter
2445 \bbl@tglobal\csname####1\string##1\endcsname}}}%
2446 \def\bbl@sctest{%
2447 \bbl@xin@{,\bbl@opt@strings,},{,\bbl@sc@label,\bbl@sc@fontenc,}}%
2448 \fi
2449 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2450 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2451 \let\AfterBabelCommands\bbl@aftercmds
2452 \let\SetString\bbl@setstring
2453 \let\bbl@stringdef\bbl@encstring
2454 \else % ie, strings=value
2455 \bbl@sctest
2456 \ifin@
2457 \let\AfterBabelCommands\bbl@aftercmds
2458 \let\SetString\bbl@setstring
2459 \let\bbl@stringdef\bbl@provstring

```

```

2460 \fi\fi\fi
2461 \bbl@scswitch
2462 \ifx\bbl@G\@empty
2463   \def\SetString##1##2{%
2464     \bbl@error{Missing group for string \string##1}%
2465     {You must assign strings to some category, typically\\%
2466       captions or extras, but you set none}}%
2467 \fi
2468 \ifx\@empty#1%
2469   \bbl@usehooks{defaultcommands}{}%
2470 \else
2471   \@expandtwoargs
2472   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2473 \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).

```

2474 \def\bbl@forlang#1#2{%
2475   \bbl@for#1\bbl@L{%
2476     \bbl@xin@{, #1, }{, \BabelLanguages,}%
2477     \ifin@#2\relax\fi}}
2478 \def\bbl@scswitch{%
2479   \bbl@forlang\bbl@tempa{%
2480     \ifx\bbl@G\@empty\else
2481       \ifx\SetString\@gobbletwo\else
2482         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2483         \bbl@xin@{, \bbl@GL, }{, \bbl@screset,}%
2484         \ifin@\else
2485           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2486           \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
2487         \fi
2488       \fi
2489     \fi}}
2490 \AtEndOfPackage{%
2491   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2492   \let\bbl@scswitch\relax}
2493 \@onlypreamble\EndBabelCommands
2494 \def\EndBabelCommands{%
2495   \bbl@usehooks{stopcommands}{}%
2496   \endgroup
2497   \endgroup
2498   \bbl@scafter}
2499 \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.

```

2500 \def\bbl@setstring#1#2{%
2501   \bbl@forlang\bbl@tempa{%
2502     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2503     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2504     {\global\expandafter % TODO - con \bbl@exp ?
2505       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2506       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
2507     }%
2508   \def\BabelString{#2}%
2509   \bbl@usehooks{stringprocess}{}%
2510   \expandafter\bbl@stringdef
2511   \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`.

```

2512 \ifx\bbl@opt@strings\relax
2513   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2514   \bbl@patchuclc
2515   \let\bbl@encoded\relax
2516   \def\bbl@encoded@uclc#1{%
2517     \@inmathwarn#1%
2518     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2519       \expandafter\ifx\csname ?\string#1\endcsname\relax
2520         \TextSymbolUnavailable#1%
2521       \else
2522         \csname ?\string#1\endcsname
2523       \fi
2524     \else
2525       \csname\cf@encoding\string#1\endcsname
2526     \fi}
2527 \else
2528   \def\bbl@scset#1#2{\def#1{#2}}
2529 \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.

```

2530 <<(*Macros local to BabelCommands)>> ≡
2531 \def\SetStringLoop##1##2{%
2532   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2533   \count@\z@
2534   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2535     \advance\count@\@ne
2536     \toks@\expandafter{\bbl@tempa}%
2537     \bbl@exp{%
2538       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2539       \count@=\the\count@\relax}}}%
2540 <</Macros local to BabelCommands>>

```

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

```

2541 \def\bbl@aftercmds#1{%
2542   \toks@\expandafter{\bbl@scafter#1}%
2543   \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.

```

2544 <<*Macros local to BabelCommands>> ≡
2545 \newcommand\SetCase[3][\%
2546   \bbl@patchuclc
2547   \bbl@forlang\bbl@tempa\%
2548   \expandafter\bbl@encstring
2549   \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2550   \expandafter\bbl@encstring
2551   \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2552   \expandafter\bbl@encstring
2553   \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2554 <</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.

```

2555 <<*Macros local to BabelCommands>> ≡
2556 \newcommand\SetHyphenMap[1]{\%
2557   \bbl@forlang\bbl@tempa\%
2558   \expandafter\bbl@stringdef
2559   \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2560 <</Macros local to BabelCommands>>

```

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

```

2561 \newcommand\BabelLower[2]{\% one to one.
2562   \ifnum\lccode#1=#2\else
2563     \babel@savevariable{\lccode#1}%
2564     \lccode#1=#2\relax
2565   \fi}
2566 \newcommand\BabelLowerMM[4]{\% many-to-many
2567   \@tempcnta=#1\relax
2568   \@tempcntb=#4\relax
2569   \def\bbl@tempa{\%
2570     \ifnum\@tempcnta>#2\else
2571       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2572       \advance\@tempcnta#3\relax
2573       \advance\@tempcntb#3\relax
2574       \expandafter\bbl@tempa
2575     \fi}%
2576   \bbl@tempa}
2577 \newcommand\BabelLowerMO[4]{\% many-to-one
2578   \@tempcnta=#1\relax
2579   \def\bbl@tempa{\%
2580     \ifnum\@tempcnta>#2\else
2581       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2582       \advance\@tempcnta#3
2583       \expandafter\bbl@tempa
2584     \fi}%
2585   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2586 <<*More package options>> ≡
2587 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2588 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2589 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2590 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}

```

```

2591 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2592 <</More package options>>

```

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

```

2593 \AtEndOfPackage{%
2594   \ifx\bb1@opt@hyphenmap\undefined
2595     \bb1@xin{,}{\bb1@language@opts}%
2596     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
2597   \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.

```

2598 \bb1@trace{Macros related to glyphs}
2599 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z\hbox{#1}%
2600   \dimen\z\ht\z@ \advance\dimen\z@ -\ht\tw@%
2601   \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.

```

2602 \def\save@sf@q#1{\leavevmode
2603   \begingroup
2604   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2605   \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.

```

2606 \ProvideTextCommand{\quotedblbase}{OT1}{%
2607   \save@sf@q{\set@low@box{\textquotedblright\}}%
2608   \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.

```

2609 \ProvideTextCommandDefault{\quotedblbase}{%
2610   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2611 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2612   \save@sf@q{\set@low@box{\textquoteright\}}%
2613   \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.

```

2614 \ProvideTextCommandDefault{\quotesinglbase}{%
2615   \UseTextSymbol{OT1}{\quotesinglbase}}

```

`\guillemetleft` The guillemet characters are not available in OT1 encoding. They are faked. (Wrong names
`\guillemetright` with o preserved for compatibility.)

```

2616 \ProvideTextCommand{\guillemetleft}{OT1}{%
2617   \ifmmode
2618     \ll
2619   \else
2620     \save@sf@q{\nobreak
2621       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2622   \fi}
2623 \ProvideTextCommand{\guillemetright}{OT1}{%
2624   \ifmmode
2625     \gg
2626   \else
2627     \save@sf@q{\nobreak
2628       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2629   \fi}
2630 \ProvideTextCommand{\guillemotleft}{OT1}{%
2631   \ifmmode
2632     \ll
2633   \else
2634     \save@sf@q{\nobreak
2635       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2636   \fi}
2637 \ProvideTextCommand{\guillemotright}{OT1}{%
2638   \ifmmode
2639     \gg
2640   \else
2641     \save@sf@q{\nobreak
2642       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2643   \fi}

```

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

```

2644 \ProvideTextCommandDefault{\guillemetleft}{%
2645   \UseTextSymbol{OT1}{\guillemetleft}}
2646 \ProvideTextCommandDefault{\guillemetright}{%
2647   \UseTextSymbol{OT1}{\guillemetright}}
2648 \ProvideTextCommandDefault{\guillemotleft}{%
2649   \UseTextSymbol{OT1}{\guillemotleft}}
2650 \ProvideTextCommandDefault{\guillemotright}{%
2651   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

\guilsinglright 2652 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2653   \ifmmode
2654     <%
2655   \else
2656     \save@sf@q{\nobreak
2657       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2658   \fi}
2659 \ProvideTextCommand{\guilsinglright}{OT1}{%
2660   \ifmmode
2661     >%
2662   \else
2663     \save@sf@q{\nobreak
2664       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2665   \fi}

```

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

```
2666 \ProvideTextCommandDefault{\guilsinglleft}{%
2667   \UseTextSymbol{OT1}{\guilsinglleft}}
2668 \ProvideTextCommandDefault{\guilsinglright}{%
2669   \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 encoded fonts. Therefore we fake it for the OT1 encoding.

```
2670 \DeclareTextCommand{\ij}{OT1}{%
2671   i\kern-0.02em\bbl@allowhyphens j}
2672 \DeclareTextCommand{\IJ}{OT1}{%
2673   I\kern-0.02em\bbl@allowhyphens J}
2674 \DeclareTextCommand{\ij}{T1}{\char188}
2675 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2676 \ProvideTextCommandDefault{\ij}{%
2677   \UseTextSymbol{OT1}{\ij}}
2678 \ProvideTextCommandDefault{\IJ}{%
2679   \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).

```
2680 \def\crrtic@{\hrule height0.1ex width0.3em}
2681 \def\crttic@{\hrule height0.1ex width0.33em}
2682 \def\ddj@{%
2683   \setbox0\hbox{d}\dimen@=\ht0
2684   \advance\dimen@1ex
2685   \dimen@.45\dimen@
2686   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2687   \advance\dimen@ii.5ex
2688   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2689 \def\DDJ@{%
2690   \setbox0\hbox{D}\dimen@=.55\ht0
2691   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2692   \advance\dimen@ii.15ex % correction for the dash position
2693   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2694   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2695   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2696 %
2697 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2698 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2699 \ProvideTextCommandDefault{\dj}{%
2700   \UseTextSymbol{OT1}{\dj}}
2701 \ProvideTextCommandDefault{\DJ}{%
2702   \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.

```
2703 \DeclareTextCommand{\SS}{OT1}{SS}
2704 \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 2705 \ProvideTextCommandDefault{\glq}{%
2706 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2707 \ProvideTextCommand{\grq}{T1}{%
2708 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2709 \ProvideTextCommand{\grq}{TU}{%
2710 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2711 \ProvideTextCommand{\grq}{OT1}{%
2712 \save@sf@q{\kern-.0125em
2713 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2714 \kern.07em\relax}}
2715 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}
```

\glqq The ‘german’ double quotes.

```
\grqq 2716 \ProvideTextCommandDefault{\glqq}{%
2717 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2718 \ProvideTextCommand{\grqq}{T1}{%
2719 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2720 \ProvideTextCommand{\grqq}{TU}{%
2721 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2722 \ProvideTextCommand{\grqq}{OT1}{%
2723 \save@sf@q{\kern-.07em
2724 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2725 \kern.07em\relax}}
2726 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}
```

\flq The ‘french’ single guillemets.

```
\frq 2727 \ProvideTextCommandDefault{\flq}{%
2728 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2729 \ProvideTextCommandDefault{\frq}{%
2730 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq The ‘french’ double guillemets.

```
\frqq 2731 \ProvideTextCommandDefault{\flqq}{%
2732 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2733 \ProvideTextCommandDefault{\frqq}{%
2734 \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 positioning, the default will be `\umlauthigh` (the normal positioning).

`\umlautlow`

```
2735 \def\umlauthigh{%
2736   \def\bbl@umlauta##1{\leavevmode\bgroup%
2737     \expandafter\accent\csname\fontencoding dqpos\endcsname
2738     ##1\bbl@allowhyphens\egroup}%
2739   \let\bbl@umlaute\bbl@umlauta}
2740 \def\umlautlow{%
2741   \def\bbl@umlauta{\protect\lower@umlaut}}
2742 \def\umlautelower{%
2743   \def\bbl@umlaute{\protect\lower@umlaut}}
2744 \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.

```
2745 \expandafter\ifx\csname U@D\endcsname\relax
2746   \csname newdimen\endcsname\U@D
2747 \fi
```

The following code fools $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ ’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.

```
2748 \def\lower@umlaut#1{%
2749   \leavevmode\bgroup
2750   \U@D 1ex%
2751   {\setbox\z@\hbox{%
2752     \expandafter\char\csname\fontencoding dqpos\endcsname}%
2753     \dimen@ -.45ex\advance\dimen@\ht\z@
2754     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2755   \expandafter\accent\csname\fontencoding dqpos\endcsname
2756   \fontdimen5\font\U@D #1%
2757   \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).

```
2758 \AtBeginDocument{%
2759   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
```

```

2760 \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaut{e}}%
2761 \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaut{i}}%
2762 \DeclareTextCompositeCommand{"}{OT1}{j}{\bbl@umlaut{j}}%
2763 \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlaut{o}}%
2764 \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlaut{u}}%
2765 \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlaut{A}}%
2766 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaut{E}}%
2767 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaut{I}}%
2768 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlaut{O}}%
2769 \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlaut{U}}

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty \language is defined. Currently used in Amharic.

```

2770 \ifx\l@english\@undefined
2771 \chardef\l@english\z@
2772 \fi
2773 % The following is used to cancel rules in ini files (see Amharic).
2774 \ifx\l@babelnohyphens\@undefined
2775 \newlanguage\l@babelnohyphens
2776 \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.

```

2777 \bbl@trace{Bidi layout}
2778 \providecommand\IfBabelLayout[3]{#3}%
2779 \newcommand\BabelPatchSection[1]{%
2780 \@ifundefined{#1}{%
2781 \bbl@exp{\let\bbl@ss@#1>\<#1>}%
2782 \@namedef{#1}{%
2783 \@ifstar{\bbl@presec@#1}%
2784 {\@dblarg{\bbl@presec@#1}}}%
2785 \def\bbl@presec@#1[#2]#3{%
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 [\\\foreignlanguage{\language}{\unexpanded{#3}}}%
2792 \\\select@language@x{\language}}%
2793 \def\bbl@presec@#1#2{%
2794 \bbl@exp{%
2795 \\\select@language@x{\bbl@main@language}%
2796 \\\bbl@cs{sspre@#1}%
2797 \\\bbl@cs{ss@#1}*%
2798 [\\\foreignlanguage{\language}{\unexpanded{#2}}]%
2799 \\\select@language@x{\language}}%
2800 \IfBabelLayout{sectioning}%
2801 {\BabelPatchSection{part}%
2802 \BabelPatchSection{chapter}%
2803 \BabelPatchSection{section}%
2804 \BabelPatchSection{subsection}%
2805 \BabelPatchSection{subsubsection}%
2806 \BabelPatchSection{paragraph}%
2807 \BabelPatchSection{subparagraph}%
2808 \def\babel@toc#1{%

```

```

2809 \select@language@x{\bbl@main@language}}{}
2810 \IfBabelLayout{captions}%
2811 {\BabelPatchSection{caption}}{}

```

9.14 Load engine specific macros

```

2812 \bbl@trace{Input engine specific macros}
2813 \ifcase\bbl@engine
2814 \input txtbabel.def
2815 \or
2816 \input luababel.def
2817 \or
2818 \input xebabel.def
2819 \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.

```

2820 \bbl@trace{Creating languages and reading ini files}
2821 \newcommand\babelprovide[2][{}%
2822 \let\bbl@savelangname\language
2823 \edef\bbl@savelocaleid{\the\localeid}%
2824 % Set name and locale id
2825 \edef\language{#2}%
2826 % \global\@namedef{\bbl@lcname@#2}{#2}%
2827 \bbl@id@assign
2828 \let\bbl@KVP@captions\@nil
2829 \let\bbl@KVP@import\@nil
2830 \let\bbl@KVP@main\@nil
2831 \let\bbl@KVP@script\@nil
2832 \let\bbl@KVP@language\@nil
2833 \let\bbl@KVP@hyphenrules\@nil % only for provide@new
2834 \let\bbl@KVP@mapfont\@nil
2835 \let\bbl@KVP@maparabic\@nil
2836 \let\bbl@KVP@mapdigits\@nil
2837 \let\bbl@KVP@intraspace\@nil
2838 \let\bbl@KVP@intrapenalty\@nil
2839 \let\bbl@KVP@onchar\@nil
2840 \let\bbl@KVP@alph\@nil
2841 \let\bbl@KVP@Alph\@nil
2842 \let\bbl@KVP@info\@nil % Ignored with import? Or error/warning?
2843 \bbl@forkv{#1}{% TODO - error handling
2844 \in@{/}{##1}%
2845 \ifin@
2846 \bbl@renewinikey##1\@{##2}%
2847 \else
2848 \bbl@csarg\def{KVP@##1}{##2}%
2849 \fi}%
2850 % == import, captions ==
2851 \ifx\bbl@KVP@import\@nil\else
2852 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2853 {\ifx\bbl@initoload\relax
2854 \begingroup
2855 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2856 \InputIfFileExists{babel-#2.tex}{}}%
2857 \endgroup

```

```

2858     \else
2859     \xdef\bb1@KVP@import{\bb1@initoload}%
2860     \fi}%
2861     {}%
2862 \fi
2863 \ifx\bb1@KVP@captions\@nil
2864     \let\bb1@KVP@captions\bb1@KVP@import
2865 \fi
2866 % Load ini
2867 \bb1@ifunset{date#2}%
2868     {\bb1@provide@new{#2}}%
2869     {\bb1@ifblank{#1}%
2870         {\bb1@error
2871             {If you want to modify `#2' you must tell how in\\%
2872             the optional argument. See the manual for the\\%
2873             available options.}%
2874             {Use this macro as documented}}%
2875         {\bb1@provide@renew{#2}}}%
2876 % Post tasks
2877 \bb1@exp{\\\babelensure[exclude=\\today]{#2}}%
2878 \bb1@ifunset{\bb1@ensure@\\languagename}%
2879     {\bb1@exp{%
2880         \\DeclareRobustCommand\<\bb1@ensure@\\languagename>[1]{%
2881             \\foreignlanguage{\\languagename}%
2882             {###1}}}%
2883     }%
2884 \bb1@exp{%
2885     \\bb1@toglobal\<\bb1@ensure@\\languagename>%
2886     \\bb1@toglobal\<\bb1@ensure@\\languagename\space>%
2887 % At this point all parameters are defined if 'import'. Now we
2888 % execute some code depending on them. But what about if nothing was
2889 % imported? We just load the very basic parameters: ids and a few
2890 % more.
2891 \bb1@ifunset{\bb1@lname#2}% TODO. Duplicated
2892     {\def\BabelBeforeIni###1##2{%
2893         \begingroup
2894         \catcode`\[=12 \catcode`\]=12 \catcode`\==12
2895         \catcode`\;=12 \catcode`\|=12 %
2896         \let\bb1@ini@captions@aux\@gobbletwo
2897         \def\bb1@inidate ####1.####2.####3.####4\relax ####5####6}%
2898         \bb1@read@ini{##1}{basic data}%
2899         \bb1@exportkey{chrng}{characters.ranges}{}%
2900         \bb1@exportkey{dgnat}{numbers.digits.native}{}%
2901         \bb1@exportkey{prehc}{typography.prehyphenchar}{}%
2902         \bb1@exportkey{lnbrk}{typography.linebreaking}{h}%
2903         \bb1@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2904         \bb1@exportkey{rgthm}{typography.righthyphenmin}{3}%
2905         \bb1@exportkey{hyphr}{typography.hyphenrules}{}%
2906         \bb1@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2907         \bb1@exportkey{hyots}{typography.hyphenate.other.script}{}%
2908         \bb1@exportkey{intsp}{typography.intraspaces}{}%
2909         \ifx\bb1@initoload\relax\endinput\fi
2910     \endgroup}%
2911 \begingroup % boxed, to avoid extra spaces:
2912 \ifx\bb1@initoload\relax
2913     \setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}%
2914 \else
2915     \setbox\z@\hbox{\BabelBeforeIni{\bb1@initoload}}%
2916 \fi

```

```

2917 \endgroup}%
2918 {}%
2919 % == script, language ==
2920 % Override the values from ini or defines them
2921 \ifx\bb1@KVP@script\@nil\else
2922 \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
2923 \fi
2924 \ifx\bb1@KVP@language\@nil\else
2925 \bb1@csarg\edef{lname@#2}{\bb1@KVP@language}%
2926 \fi
2927 % == onchar ==
2928 \ifx\bb1@KVP@onchar\@nil\else
2929 \bb1@luahyphenate
2930 \directlua{
2931   if Babel.locale_mapped == nil then
2932     Babel.locale_mapped = true
2933     Babel.linebreaking.add_before(Babel.locale_map)
2934     Babel.loc_to_scr = {}
2935     Babel.chr_to_loc = Babel.chr_to_loc or {}
2936   end}%
2937 \bb1@xin@{ ids }{ \bb1@KVP@onchar\space}%
2938 \ifin@
2939 \ifx\bb1@starthyphens\@undefined % Needed if no explicit selection
2940 \AddBabelHook{babel-onchar}{beforestart}{\bb1@starthyphens}%
2941 \fi
2942 \bb1@exp{\bb1@add\bb1@starthyphens
2943 { \bb1@patterns@lua{languagename} }}%
2944 % TODO - error/warning if no script
2945 \directlua{
2946   if Babel.script_blocks['\bb1@cl{sbc}'] then
2947     Babel.loc_to_scr[\the\localeid] =
2948       Babel.script_blocks['\bb1@cl{sbc}']
2949     Babel.locale_props[\the\localeid].lc = \the\localeid\space
2950     Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@languagename}\space
2951   end
2952 }%
2953 \fi
2954 \bb1@xin@{ fonts }{ \bb1@KVP@onchar\space}%
2955 \ifin@
2956 \bb1@ifunset{bb1@lsys@languagename}{\bb1@provide@lsys{languagename}}{%
2957 \bb1@ifunset{bb1@wdir@languagename}{\bb1@provide@dirs{languagename}}{%
2958 \directlua{
2959   if Babel.script_blocks['\bb1@cl{sbc}'] then
2960     Babel.loc_to_scr[\the\localeid] =
2961       Babel.script_blocks['\bb1@cl{sbc}']
2962   end}%
2963 \ifx\bb1@mapselect\@undefined
2964 \AtBeginDocument{%
2965 \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
2966 {\selectfont}}%
2967 \def\bb1@mapselect{%
2968 \let\bb1@mapselect\relax
2969 \edef\bb1@prefontid{\fontid\font}}%
2970 \def\bb1@mapdir##1{%
2971 {\def\languagename{##1}%
2972 \let\bb1@ifrestoring\@firstoftwo % To avoid font warning
2973 \bb1@switchfont
2974 \directlua{
2975   Babel.locale_props[\the\csname bb1@id@##1\endcsname]%

```

```

2976             ['/bbl@prefontid'] = \fontid\font\space}}}%
2977     \fi
2978     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2979     \fi
2980     % TODO - catch non-valid values
2981 \fi
2982 % == mapfont ==
2983 % For bidi texts, to switch the font based on direction
2984 \ifx\bbl@KVP@mapfont\@nil\else
2985     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
2986     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2987         mapfont. Use 'direction'.%
2988         {See the manual for details.}}}%
2989     \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys@\language}}}%
2990     \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}}%
2991 \ifx\bbl@mapselect\@undefined
2992     \AtBeginDocument{%
2993         \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2994         {\selectfont}}%
2995     \def\bbl@mapselect{%
2996         \let\bbl@mapselect\relax
2997         \edef\bbl@prefontid{\fontid\font}}%
2998     \def\bbl@mapdir##1{%
2999         {\def\language{##1}%
3000         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3001         \bbl@switchfont
3002         \directlua{Babel.fontmap
3003             [\the\csname bbl@wdir@##1\endcsname]%
3004             [\bbl@prefontid]=\fontid\font}}}%
3005     \fi
3006     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3007 \fi
3008 % == intraspace, intrapenalty ==
3009 % For CJK, East Asian, Southeast Asian, if interspace in ini
3010 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3011     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3012 \fi
3013 \bbl@provide@intraspace
3014 % == hyphenate.other.locale ==
3015 \bbl@ifunset{\bbl@hyotl@\language}}}%
3016 {\bbl@csarg\bbl@replace{hyotl@\language}{ }{,}}%
3017 \bbl@startcommands*\language}}}%
3018 \bbl@csarg\bbl@foreach{hyotl@\language}{%
3019     \ifcase\bbl@engine
3020     \ifnum##1<257
3021         \SetHyphenMap{\BabelLower{##1}{##1}}%
3022     \fi
3023     \else
3024         \SetHyphenMap{\BabelLower{##1}{##1}}%
3025     \fi}%
3026 \bbl@endcommands}%
3027 % == hyphenate.other.script ==
3028 \bbl@ifunset{\bbl@hyots@\language}}}%
3029 {\bbl@csarg\bbl@replace{hyots@\language}{ }{,}}%
3030 \bbl@csarg\bbl@foreach{hyots@\language}{%
3031     \ifcase\bbl@engine
3032     \ifnum##1<257
3033         \global\lcode##1=##1\relax
3034     \fi

```

```

3035     \else
3036     \global\lccode##1=##1\relax
3037     \fi}}%
3038 % == maparabic ==
3039 % Native digits, if provided in ini (TeX level, xe and lua)
3040 \ifcase\bbbl@engine\else
3041     \bbbl@ifunset{\bbbl@dgnat@\languagename}{}%
3042     {\expandafter\ifx\csname \bbbl@dgnat@\languagename\endcsname\@empty\else
3043     \expandafter\expandafter\expandafter
3044     \bbbl@setdigits\csname \bbbl@dgnat@\languagename\endcsname
3045     \ifx\bbbl@KVP@maparabic\@nil\else
3046     \ifx\bbbl@latinarabic\@undefined
3047     \expandafter\let\expandafter\@arabic
3048     \csname \bbbl@counter@\languagename\endcsname
3049     \else % ie, if layout=counters, which redefines \@arabic
3050     \expandafter\let\expandafter\bbbl@latinarabic
3051     \csname \bbbl@counter@\languagename\endcsname
3052     \fi
3053     \fi
3054     \fi}%
3055 \fi
3056 % == mapdigits ==
3057 % Native digits (lua level).
3058 \ifodd\bbbl@engine
3059     \ifx\bbbl@KVP@mapdigits\@nil\else
3060     \bbbl@ifunset{\bbbl@dgnat@\languagename}{}%
3061     {\RequirePackage{luatexbase}%
3062     \bbbl@activate@preotf
3063     \directlua{
3064         Babel = Babel or {} %% -> presets in luababel
3065         Babel.digits_mapped = true
3066         Babel.digits = Babel.digits or {}
3067         Babel.digits[\the\localeid] =
3068             table.pack(string.utfvalue('\bbbl@cl{dgnat}'))
3069         if not Babel.numbers then
3070             function Babel.numbers(head)
3071                 local LOCALE = luatexbase.registernumber'\bbbl@attr@locale'
3072                 local GLYPH = node.id'glyph'
3073                 local inmath = false
3074                 for item in node.traverse(head) do
3075                     if not inmath and item.id == GLYPH then
3076                         local temp = node.get_attribute(item, LOCALE)
3077                         if Babel.digits[temp] then
3078                             local chr = item.char
3079                             if chr > 47 and chr < 58 then
3080                                 item.char = Babel.digits[temp][chr-47]
3081                             end
3082                         end
3083                     elseif item.id == node.id'math' then
3084                         inmath = (item.subtype == 0)
3085                     end
3086                 end
3087                 return head
3088             end
3089         end
3090     } }%
3091     \fi
3092 \fi
3093 % == alph, Alph ==

```



```

3094 % What if extras<lang> contains a \babel@save\@alph? It won't be
3095 % restored correctly when exiting the language, so we ignore
3096 % this change with the \bbl@alph@saved trick.
3097 \ifx\bbl@KVP@alph\@nil\else
3098   \toks@\expandafter\expandafter\expandafter{%
3099     \csname extras\language\endcsname}%
3100   \bbl@exp{%
3101     \def\<extras\language>{%
3102       \let\\bbl@alph@saved\\@alph
3103       \the\toks@
3104       \let\\@alph\\bbl@alph@saved
3105       \\babel@save\\@alph
3106       \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
3107   \fi
3108 \ifx\bbl@KVP@Alph\@nil\else
3109   \toks@\expandafter\expandafter\expandafter{%
3110     \csname extras\language\endcsname}%
3111   \bbl@exp{%
3112     \def\<extras\language>{%
3113       \let\\bbl@Alph@saved\\@Alph
3114       \the\toks@
3115       \let\\@Alph\\bbl@Alph@saved
3116       \\babel@save\\@Alph
3117       \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3118   \fi
3119 % == require.babel in ini ==
3120 % To load or reload the babel-*.tex, if require.babel in ini
3121 \bbl@ifunset{bbl@rqtex@\language}{}%
3122 {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3123   \let\BabelBeforeIni\@gobbletwo
3124   \chardef\atcatcode=\catcode`\@
3125   \catcode`\@=11\relax
3126   \InputIfFileExists{babel-\bbl@cs{rqtex@\language}.tex}{}%
3127   \catcode`\@=\atcatcode
3128   \let\atcatcode\relax
3129 \fi}%
3130 % == main ==
3131 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3132   \let\language\bbl@savelangname
3133   \chardef\localeid\bbl@savelocaleid\relax
3134 \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.

```

3135 % TODO. Merge with \localenumeral:
3136 % \newcommand\localedigits{\@nameuse{\language digits}}
3137 \def\bbl@setdigits#1#2#3#4#5{%
3138   \bbl@exp{%
3139     \def\<\language digits>####1{% ie, \langdigits
3140       \<bbl@digits@\language>####1\\@nil}%
3141     \def\<\language counter>####1{% ie, \langcounter
3142       \\expandafter\<bbl@counter@\language>%
3143       \\csname c#####1\endcsname}%
3144     \def\<bbl@counter@\language>####1{% ie, \bbl@counter@lang
3145       \\expandafter\<bbl@digits@\language>%
3146       \\number####1\\@nil}}%
3147   \def\bbl@tempa##1##2##3##4##5{%
3148     \bbl@exp{% Wow, quite a lot of hashes! :-(

```

[illegible]

```

3167 \def\bbl@provide@new#1{%
3168   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3169   \@namedef{extras#1}{}%
3170   \@namedef{noextras#1}{}%
3171   \bbl@startcommands*{#1}{captions}%
3172     \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3173       \def\bbl@tempb##1{% elt for \bbl@captionslist
3174         \ifx##1\@empty\else
3175           \bbl@exp{%
3176             \\SetString\\##1{%
3177               \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3178             \expandafter\bbl@tempb
3179             \fi}%
3180             \expandafter\bbl@tempb\bbl@captionslist\@empty
3181           \else
3182             \ifx\bbl@initoload\relax
3183               \bbl@read@ini{\bbl@KVP@captions}{data}% Here letters cat = 11
3184             \else
3185               \bbl@read@ini{\bbl@initoload}{data}% Here all letters cat = 11
3186             \fi
3187             \bbl@after@ini
3188             \bbl@savestrings
3189           \fi
3190   \StartBabelCommands*{#1}{date}%
3191   \ifx\bbl@KVP@import\@nil
3192     \bbl@exp{%
3193       \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
3194   \else
3195     \bbl@savetoday
3196     \bbl@savedate
3197   \fi
3198   \bbl@endcommands
3199   \bbl@ifunset{bbl@lname@#1}% TODO. Duplicated
3200   {\def\BabelBeforeIni##1##2{%
3201     \begingroup
3202       \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3203       \catcode`\;=12 \catcode`\|=12 %
3204       \let\bbl@ini@captions@aux\@gobbletwo
3205       \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6{}}%

```

```

3206 \bbl@read@ini{##1}{basic data}%
3207 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3208 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3209 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3210 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3211 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3212 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3213 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3214 \bbl@exportkey{chrng}{characters.ranges}{}%
3215 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3216 \ifx\bbl@initoload\relax\endinput\fi
3217 \endgroup}%
3218 \begingroup % boxed, to avoid extra spaces:
3219 \ifx\bbl@initoload\relax
3220 \setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%
3221 \else
3222 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3223 \fi
3224 \endgroup}%
3225 {}%
3226 \bbl@exp{%
3227 \gdef\<#1hyphenmins>{%
3228 {\bbl@ifunset{\bbl@lftm@#1}{2}{\bbl@cs{lftm@#1}}}%
3229 {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3230 \bbl@provide@hyphens{#1}%
3231 \ifx\bbl@KVP@main\@nil\else
3232 \expandafter\main@language\expandafter{#1}%
3233 \fi}
3234 \def\bbl@provide@renew#1{%
3235 \ifx\bbl@KVP@captions\@nil\else
3236 \StartBabelCommands*{#1}{captions}%
3237 \bbl@read@ini{\bbl@KVP@captions}{data}% Here all letters cat = 11
3238 \bbl@after@ini
3239 \bbl@savestrings
3240 \EndBabelCommands
3241 \fi
3242 \ifx\bbl@KVP@import\@nil\else
3243 \StartBabelCommands*{#1}{date}%
3244 \bbl@savetoday
3245 \bbl@savedate
3246 \EndBabelCommands
3247 \fi
3248 % == hyphenrules ==
3249 \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3250 \def\bbl@provide@hyphens#1{%
3251 \let\bbl@tempa\relax
3252 \ifx\bbl@KVP@hyphenrules\@nil\else
3253 \bbl@replace\bbl@KVP@hyphenrules{ },}%
3254 \bbl@foreach\bbl@KVP@hyphenrules{%
3255 \ifx\bbl@tempa\relax % if not yet found
3256 \bbl@ifsamestring{##1}{+}%
3257 {\bbl@exp{\addlanguage\<l@##1>}}}%
3258 {}%
3259 \bbl@ifunset{l@##1}%
3260 {}%
3261 {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
3262 \fi}%

```

```

3263 \fi
3264 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
3265 \ifx\bbbl@KVP@import\@nil
3266 \ifx\bbbl@initoload\relax\else
3267 \bbbl@exp{% and hyphenrules is not empty
3268 \\\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3269 }%
3270 {\let\\bbbl@tempa<l@\bbbl@cl{hyphr}>}}%
3271 \fi
3272 \else % if importing
3273 \bbbl@exp{% and hyphenrules is not empty
3274 \\\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3275 }%
3276 {\let\\bbbl@tempa<l@\bbbl@cl{hyphr}>}}%
3277 \fi
3278 \fi
3279 \bbbl@ifunset{\bbbl@tempa}% ie, relax or undefined
3280 {\bbbl@ifunset{l@#1}% no hyphenrules found - fallback
3281 {\bbbl@exp{\\adddialect<l@#1>\language}}%
3282 }% so, l@<lang> is ok - nothing to do
3283 {\bbbl@exp{\\adddialect<l@#1>\bbbl@tempa}}% found in opt list or ini
3284

```

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.

```

3285 \ifx\bbbl@readstream\@undefined
3286 \csname newread\endcsname\bbbl@readstream
3287 \fi
3288 \def\bbbl@inipreread#1=#2\@{%
3289 \bbbl@trim\def\bbbl@tempa{#1}% Redundant below !!
3290 \bbbl@trim\toks@{#2}%
3291 % Move trims here ??
3292 \bbbl@ifunset{\bbbl@KVP@\bbbl@section/\bbbl@tempa}%
3293 {\bbbl@exp{%
3294 \\\g@addto@macro\\bbbl@inidata{%
3295 \\\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3296 \expandafter\bbbl@inireader\bbbl@tempa=#2\@}%
3297 }%
3298 \def\bbbl@read@ini#1#2{%
3299 \bbbl@csarg\edef{lini@language}{#1}%
3300 \openin\bbbl@readstream=babel-#1.ini
3301 \ifeof\bbbl@readstream
3302 \bbbl@error
3303 {There is no ini file for the requested language\\%
3304 (#1). Perhaps you misspelled it or your installation\\%
3305 is not complete.}%
3306 {Fix the name or reinstall babel.}%
3307 \else
3308 \bbbl@exp{\def\\bbbl@inidata{\\bbbl@elt{identificacion}{tag.ini}{#1}}}%
3309 \let\bbbl@section\@empty
3310 \let\bbbl@savestrings\@empty
3311 \let\bbbl@savetoday\@empty
3312 \let\bbbl@savestate\@empty
3313 \let\bbbl@inireader\bbbl@iniskip
3314 \bbbl@info{Importing #2 for language\\%
3315 from babel-#1.ini. Reported}%
3316 \loop
3317 \if T\ifeof\bbbl@readstream F\fi T\relax % Trick, because inside \loop
3318 \endlinechar\m@ne

```

```

3319 \read\bbbl@readstream to \bbbl@line
3320 \endlinechar`\^^M
3321 \ifx\bbbl@line\@empty\else
3322 \expandafter\bbbl@inline\bbbl@line\bbbl@inline
3323 \fi
3324 \repeat
3325 \bbbl@foreach\bbbl@renewlist{%
3326 \bbbl@ifunset{bbbl@renew###1}{\bbbl@inisec[##1]\@}%
3327 \global\let\bbbl@renewlist\@empty
3328 % Ends last section. See \bbbl@inisec
3329 \def\bbbl@elt##1##2{\bbbl@inireader##1=##2\@}%
3330 \bbbl@cs{renew@\bbbl@section}%
3331 \global\bbbl@csarg\let{renew@\bbbl@section}\relax
3332 \bbbl@cs{secpost@\bbbl@section}%
3333 \bbbl@csarg{\global\expandafter\let}{inidata@\language}\bbbl@inidata
3334 \bbbl@exp{\bbbl@add@list\bbbl@ini@loaded{\language}}%
3335 \bbbl@tglobal\bbbl@ini@loaded
3336 \fi}
3337 \def\bbbl@inline#1\bbbl@inline{%
3338 \@ifnextchar[\bbbl@inisec{\@ifnextchar;\bbbl@iniskip\bbbl@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.

```

3339 \def\bbbl@iniskip#1\@{% if starts with ;
3340 \def\bbbl@inisec[##1]##2\@{% if starts with opening bracket
3341 \def\bbbl@elt##1##2{%
3342 \expandafter\toks\expandafter{%
3343 \expandafter{\bbbl@section}{##1}{##2}}%
3344 \bbbl@exp{%
3345 \g@addto@macro\bbbl@inidata{\bbbl@elt\the\toks}%
3346 \bbbl@inireader##1=##2\@}%
3347 \bbbl@cs{renew@\bbbl@section}%
3348 \global\bbbl@csarg\let{renew@\bbbl@section}\relax
3349 \bbbl@cs{secpost@\bbbl@section}%
3350 % The previous code belongs to the previous section.
3351 % Now start the current one.
3352 \def\bbbl@section{##1}%
3353 \def\bbbl@elt##1##2{%
3354 \@namedef{bbbl@KVP@##1/##1}{}}%
3355 \bbbl@cs{renew@##1}%
3356 \bbbl@cs{secpre@##1}% pre-section `hook'
3357 \bbbl@ifunset{bbbl@inikv@##1}%
3358 {\let\bbbl@inireader\bbbl@iniskip}%
3359 {\bbbl@exp{\let\bbbl@inireader\bbbl@inikv@##1>}}}
3360 \let\bbbl@renewlist\@empty
3361 \def\bbbl@renewinikey#1/#2\@#3{%
3362 \bbbl@ifunset{bbbl@renew@##1}%
3363 {\bbbl@add@list\bbbl@renewlist{##1}}%
3364 {}%
3365 \bbbl@csarg\bbbl@add{renew@##1}{\bbbl@elt{##2}{##3}}

```

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

```

3366 \def\bbbl@inikv#1=##2\@{% key=value
3367 \bbbl@trim\def\bbbl@tempa{##1}%
3368 \bbbl@trim\toks{##2}%
3369 \bbbl@csarg\edef{kv@\bbbl@section.\bbbl@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.

```

3370 \def\bbl@exportkey#1#2#3{%
3371   \bbl@ifunset{bbl@kv@#2}%
3372     {\bbl@csarg\gdef{#1@\language}\{#3}}%
3373     {\expandafter\ifx\csname bbl@kv@#2\endcsname\empty
3374       \bbl@csarg\gdef{#1@\language}\{#3}}%
3375     \else
3376       \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3377       \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@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

```

3378 \def\bbl@iniwarning#1{%
3379   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3380     {\bbl@warning{%
3381       From babel-\bbl@cs{lini@\language}.ini:\%
3382       \bbl@cs{@kv@identification.warning#1}\%
3383       Reported }}%
3384   \let\bbl@inikv@identification\bbl@inikv
3385   \def\bbl@secpost@identification{%
3386     \bbl@iniwarning}%
3387   \ifcase\bbl@engine
3388     \bbl@iniwarning{.pdflatex}%
3389   \or
3390     \bbl@iniwarning{.lualatex}%
3391   \or
3392     \bbl@iniwarning{.xelatex}%
3393   \fi%
3394   \bbl@exportkey{elname}{identification.name.english}{}%
3395   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3396     {\csname bbl@elname@\language\endcsname}}%
3397   \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
3398   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3399   \bbl@exportkey{esname}{identification.script.name}{}%
3400   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3401     {\csname bbl@esname@\language\endcsname}}%
3402   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3403   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
3404   \let\bbl@inikv@typography\bbl@inikv
3405   \let\bbl@inikv@characters\bbl@inikv
3406   \let\bbl@inikv@numbers\bbl@inikv
3407   \def\bbl@inikv@counters#1=#2\@@{%
3408     \bbl@ifsamestring{#1}{digits}%
3409       {\bbl@error{The counter name 'digits' is reserved for mapping\%
3410         decimal digits}%
3411         {Use another name.}}%
3412     {}}%
3413   \def\bbl@tempc{#1}%
3414   \bbl@trim@def{\bbl@tempb*}{#2}%
3415   \in@{.1$}{#1$}%
3416   \ifin@
3417     \bbl@replace\bbl@tempc{.1}{}%
3418     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
3419       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3420   \fi

```

```

3421 \in@{.F.}{#1}%
3422 \ifin@else\in@{.S.}{#1}\fi
3423 \ifin@
3424 \bbl@csarg\protected@xdef{cntr@#1@\language name}{\bbl@tempb*}%
3425 \else
3426 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3427 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3428 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language name}\bbl@tempa
3429 \fi}
3430 \def\bbl@after@ini{%
3431 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3432 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3433 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3434 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3435 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3436 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3437 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3438 \bbl@exportkey{intsp}{typography.intraspace}{}%
3439 \bbl@exportkey{jstfy}{typography.justify}{w}%
3440 \bbl@exportkey{chrng}{characters.ranges}{}%
3441 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3442 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3443 \bbl@toglobal\bbl@savetoday
3444 \bbl@toglobal\bbl@savestate}

```

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

```

3445 \ifcase\bbl@engine
3446 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3447 \bbl@ini@captions@aux{#1}{#2}}
3448 \else
3449 \def\bbl@inikv@captions#1=#2\@@{%
3450 \bbl@ini@captions@aux{#1}{#2}}
3451 \fi

```

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

```

3452 \def\bbl@ini@captions@aux#1#2{%
3453 \bbl@trim\def\bbl@tempa{#1}%
3454 \bbl@ifblank{#2}%
3455 {\bbl@exp{%
3456 \toks@{\bbl@nocaption{\bbl@tempa}{\language name\bbl@tempa name}}}%
3457 {\bbl@trim\toks@{#2}}}%
3458 \bbl@exp{%
3459 \bbl@add\bbl@savestrings{%
3460 \SetString\<\bbl@tempa name>{\the\toks@}}}}

```

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

TODO. Remove cypypaste pattern.

```

3461 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{% for defaults
3462 \bbl@inidate#1...\relax{#2}}
3463 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
3464 \bbl@inidate#1...\relax{#2}{islamic}}
3465 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
3466 \bbl@inidate#1...\relax{#2}{hebrew}}
3467 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
3468 \bbl@inidate#1...\relax{#2}{persian}}

```

```

3469 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
3470 \bbl@inidate#1...\relax{#2}{indian}}
3471 \ifcase\bbl@engine
3472 \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{% override
3473 \bbl@inidate#1...\relax{#2}{}}
3474 \bbl@csarg\def{secpre@date.gregorian.licr}{% discard uni
3475 \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
3476 \fi
3477 % TODO. With the following there is no need to ensure if \select...
3478 \newcommand\localedate{\@nameuse{\bbl@date@\language}}
3479 % eg: 1=months, 2=wide, 3=1, 4=dummy
3480 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3481 \bbl@trim@def\bbl@tempa{#1.#2}%
3482 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3483 {\bbl@trim@def\bbl@tempa{#3}%
3484 \bbl@trim\toks@{#5}%
3485 \bbl@exp{%
3486 \\\bbl@add\\\bbl@savestate{%
3487 \\\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}}}%
3488 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3489 {\bbl@trim@def\bbl@toreplace{#5}%
3490 \bbl@TG@date
3491 \global\bbl@csarg\let{date@\language}\bbl@toreplace
3492 \bbl@exp{%
3493 \gdef\<\language date>{\\\protect\<\language date >}%
3494 \gdef\<\language date >####1####2####3{%
3495 \\\bbl@usedategroupttrue
3496 \<bbl@ensure@\language>{%
3497 \<bbl@date@\language>{####1}{####2}{####3}}}%
3498 \\\bbl@add\\\bbl@savetoday{%
3499 \\\SetString\\today{%
3500 \<\language date>{\\\the\year}{\\the\month}{\\the\day}}}}}%
3501 {}

```

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.

```

3502 \let\bbl@calendar\@empty
3503 \newcommand\BabelDateSpace{\nobreakspace}
3504 \newcommand\BabelDateDot{.\@}
3505 \newcommand\BabelDated[1]{\number#1}
3506 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3507 \newcommand\BabelDateM[1]{\number#1}
3508 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3509 \newcommand\BabelDateMMMM[1]{%
3510 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3511 \newcommand\BabelDatey[1]{\number#1}%
3512 \newcommand\BabelDateyy[1]{%
3513 \ifnum#1<10 0\number#1 %
3514 \else\ifnum#1<100 \number#1 %
3515 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3516 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3517 \else
3518 \bbl@error
3519 {Currently two-digit years are restricted to the\
3520 range 0-9999.}%
3521 {There is little you can do. Sorry.}%
3522 \fi\fi\fi\fi}
3523 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0

```



```

3524 \def\bb1@replace@finish@iii#1{%
3525   \bb1@exp{\def\#1###1###2###3{\the\toks@}}
3526 \def\bb1@TG@date{%
3527   \bb1@replace\bb1@toreplace{[ ]}{\BabelDateSpace{}}%
3528   \bb1@replace\bb1@toreplace{[. ]}{\BabelDateDot{}}%
3529   \bb1@replace\bb1@toreplace{[d]}{\BabelDated{###3}}%
3530   \bb1@replace\bb1@toreplace{[dd]}{\BabelDatedd{###3}}%
3531   \bb1@replace\bb1@toreplace{[M]}{\BabelDateM{###2}}%
3532   \bb1@replace\bb1@toreplace{[MM]}{\BabelDateMM{###2}}%
3533   \bb1@replace\bb1@toreplace{[MMMM]}{\BabelDateMMMM{###2}}%
3534   \bb1@replace\bb1@toreplace{[y]}{\BabelDatey{###1}}%
3535   \bb1@replace\bb1@toreplace{[yy]}{\BabelDateyy{###1}}%
3536   \bb1@replace\bb1@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3537   \bb1@replace\bb1@toreplace{[y|]}{\bb1@datecctr[###1|]}%
3538   \bb1@replace\bb1@toreplace{[m|]}{\bb1@datecctr[###2|]}%
3539   \bb1@replace\bb1@toreplace{[d|]}{\bb1@datecctr[###3|]}%
3540 % Note after \bb1@replace \toks@ contains the resulting string.
3541 % TODO - Using this implicit behavior doesn't seem a good idea.
3542   \bb1@replace@finish@iii\bb1@toreplace}
3543 \def\bb1@datecctr[#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.

```

3544 \def\bb1@provide@lsys#1{%
3545   \bb1@ifunset{bb1@lname@#1}%
3546     {\bb1@ini@basic{#1}}%
3547     {}%
3548   \bb1@csarg\let{lsys@#1}\@empty
3549   \bb1@ifunset{bb1@sname@#1}{\bb1@csarg\gdef{sname@#1}{Default}}{}%
3550   \bb1@ifunset{bb1@sotf@#1}{\bb1@csarg\gdef{sotf@#1}{DFLT}}{}%
3551   \bb1@csarg\bb1@add@list{lsys@#1}{Script=\bb1@cs{sname@#1}}%
3552   \bb1@ifunset{bb1@lname@#1}%
3553     {\bb1@csarg\bb1@add@list{lsys@#1}{Language=\bb1@cs{lname@#1}}}%
3554   \ifcase\bb1@engine\or\or
3555     \bb1@ifunset{bb1@prehc@#1}%
3556       {\bb1@exp{\bb1@ifblank{\bb1@cs{prehc@#1}}}%
3557       {}%
3558       {\bb1@csarg\bb1@add@list{lsys@#1}{HyphenChar="200B}}}%
3559   \fi
3560   \bb1@csarg\bb1@toGLOBAL{lsys@#1}}

```

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

```

3561 \def\bb1@ini@basic#1{%
3562   \def\BabelBeforeIni##1##2{%
3563     \begingroup
3564       \bb1@add\bb1@secpost@identification{\closein\bb1@readstream}%
3565       \catcode`\[=12 \catcode`\]=12 \catcode`\=12
3566       \catcode`\;=12 \catcode`\|=12 %
3567       \bb1@read@ini{##1}{font and identification data}%
3568       \endinput % babel- .tex may contain onlypreamble's
3569       \endgroup}% boxed, to avoid extra spaces:
3570   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3571 \def\bb1@buildifcase#1 {% Returns \bb1@tempa, requires \toks@={ }

```

```

3572 \ifx\\#1% % \\ before, in case #1 is multiletter
3573 \bbl@exp{%
3574 \def\\bbl@tempa####1{%
3575 \<ifcase>####1\space\the\toks@\\<else>\\@ctrerr\<fi>}}%
3576 \else
3577 \toks@\\expandafter{\the\toks@\\or #1}%
3578 \expandafter\bbl@buildifcase
3579 \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 an special case, for a fixed form (see babel-he.ini, for example).

```

3580 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1\language}{#2}}
3581 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3582 % TODO. \localecounter{digits}{..} What a mistake on my part!!
3583 % But the solution seems even logical ;- )
3584 \newcommand\localecounter[2]{%
3585 \expandafter\bbl@localecntr\csname c@#2\endcsname{#1}}
3586 \def\bbl@alphanumeric#1#2{%
3587 \expandafter\bbl@alphanumeric@i\number#2 76543210\\@{#1}}
3588 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\\@{#9}{%
3589 \ifcase\car#8\\nil\or % Currenty <10000, but prepared for bigger
3590 \bbl@alphanumeric@ii{#9}000000#1\or
3591 \bbl@alphanumeric@ii{#9}00000#1#2\or
3592 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3593 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3594 \bbl@alphnum@invalid{>9999}%
3595 \fi}
3596 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3597 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8\language}%
3598 {\bbl@cs{cntr@#1.4\language}{#5}
3599 \bbl@cs{cntr@#1.3\language}{#6}
3600 \bbl@cs{cntr@#1.2\language}{#7}
3601 \bbl@cs{cntr@#1.1\language}{#8}
3602 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3603 \bbl@ifunset{bbl@cntr@#1.S.321\language}{}%
3604 {\bbl@cs{cntr@#1.S.321\language}{}%
3605 \fi}%
3606 {\bbl@cs{cntr@#1.F.\number#5#6#7#8\language}}}
3607 \def\bbl@alphnum@invalid#1{%
3608 \bbl@error{Alphabetic numeral too large (#1)}%
3609 {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.

```

3610 \newcommand\localeinfo[1]{%
3611 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
3612 {\bbl@error{I've found no info for the current locale.\\%
3613 The corresponding ini file has not been loaded\\%
3614 Perhaps it doesn't exist}%
3615 {See the manual for details.}}%
3616 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}}
3617 % \namedef{bbl@info@name.locale}{lcname}
3618 \namedef{bbl@info@tag.ini}{lini}
3619 \namedef{bbl@info@name.english}{elname}

```

```

3620 \@namedef{bbl@info@name.opentype}{lname}
3621 \@namedef{bbl@info@tag.bcp47}{lbcpl}
3622 \@namedef{bbl@info@tag.opentype}{lotf}
3623 \@namedef{bbl@info@script.name}{esname}
3624 \@namedef{bbl@info@script.name.opentype}{sname}
3625 \@namedef{bbl@info@script.tag.bcp47}{sbcpl}
3626 \@namedef{bbl@info@script.tag.opentype}{sotf}
3627 \let\bbl@ensureinfo\@gobble
3628 \newcommand\BabelEnsureInfo{%
3629   \def\bbl@ensureinfo##1{%
3630     \ifx\InputIfFileExists\undefined\else % not in plain
3631       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}%
3632     \fi}}

```

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`.

```

3633 \newcommand\getlocaleproperty[3]{%
3634   \let#1\relax
3635   \def\bbl@elt##1##2##3{%
3636     \bbl@ifsamestring{##1/##2}{##3}%
3637     {\providecommand#1{##3}%
3638     \def\bbl@elt####1####2####3{}}}%
3639   {}}%
3640   \bbl@cs{inidata@#2}%
3641   \ifx#1\relax
3642     \bbl@error
3643     {Unknown key for locale '#2':\%
3644     #3}%
3645     \string#1 will be set to \relax}%
3646     {Perhaps you misspelled it.}%
3647   \fi}
3648 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel bahavior

A generic high level interface is provided to adjust some global and general settings.

```

3649 \newcommand\babeladjust[1]{% TODO. Error handling.
3650   \bbl@forkv{#1}{%
3651     \bbl@ifunset{bbl@ADJ@##1@##2}%
3652     {\bbl@cs{ADJ@##1}{##2}}%
3653     {\bbl@cs{ADJ@##1@##2}}}
3654 %
3655 \def\bbl@adjust@lua#1#2{%
3656   \ifvmode
3657     \ifnum\currentgrouplevel=\z@
3658       \directlua{ Babel.#2 }%
3659       \expandafter\expandafter\expandafter\@gobble
3660     \fi
3661   \fi
3662   {\bbl@error % The error is gobbled if everything went ok.
3663     {Currently, #1 related features can be adjusted only\%
3664     in the main vertical list.}%
3665     {Maybe things change in the future, but this is what it is.}}}
3666 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3667   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3668 \@namedef{bbl@ADJ@bidi.mirroring@off}{%

```

```

3669 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3670 \@namedef{bbl@ADJ@bidi.text@on}{%
3671 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3672 \@namedef{bbl@ADJ@bidi.text@off}{%
3673 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3674 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3675 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3676 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3677 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3678 %
3679 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3680 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3681 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3682 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3683 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3684 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3685 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3686 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3687 %
3688 \def\bbl@adjust@layout#1{%
3689 \ifvmode
3690 #1%
3691 \expandafter\@gobble
3692 \fi
3693 {\bbl@error % The error is gobbled if everything went ok.
3694 {Currently, layout related features can be adjusted only\\%
3695 in vertical mode.}%
3696 {Maybe things change in the future, but this is what it is.}}
3697 \@namedef{bbl@ADJ@layout.tabular@on}{%
3698 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3699 \@namedef{bbl@ADJ@layout.tabular@off}{%
3700 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3701 \@namedef{bbl@ADJ@layout.lists@on}{%
3702 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3703 \@namedef{bbl@ADJ@layout.lists@off}{%
3704 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3705 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3706 \bbl@activateposthyphen}
3707 %
3708 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3709 \bbl@bcpallowedtrue}
3710 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3711 \bbl@bcpallowedfalse}
3712 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3713 \def\bbl@bcp@prefix{#1}}
3714 \def\bbl@bcp@prefix{bcp47-}
3715 \@namedef{bbl@ADJ@autoload.options}#1{%
3716 \def\bbl@autoload@options{#1}}
3717 \let\bbl@autoload@bcptoptions\@empty
3718 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3719 \def\bbl@autoload@bcptoptions{#1}}
3720 % TODO: use babel name, override
3721 %
3722 % As the final task, load the code for lua.
3723 %
3724 \ifx\directlua\@undefined\else
3725 \ifx\bbl@luapatterns\@undefined
3726 \input luababel.def
3727 \fi

```

```

3728 \fi
3729 </core>

A proxy file for switch.def

3730 <*kernel>
3731 \let\bbl@onlyswitch\@empty
3732 \input babel.def
3733 \let\bbl@onlyswitch\@undefined
3734 </kernel>
3735 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by $\text{iniT}_\text{E}\text{X}$ because it should instruct $\text{T}_\text{E}\text{X}$ 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 $\text{L}_\text{A}\text{T}_\text{E}_\text{X}$ 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.

```

3736 <<Make sure ProvidesFile is defined>>
3737 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
3738 \xdef\bbl@format{\jobname}
3739 \def\bbl@version{\<version>}
3740 \def\bbl@date{\<date>}
3741 \ifx\AtBeginDocument\@undefined
3742   \def\@empty{}
3743   \let\orig@dump\dump
3744   \def\dump{%
3745     \ifx\@ztryfc\@undefined
3746     \else
3747       \toks0=\expandafter{\@preamblecmds}%
3748       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3749       \def\@begindocumenthook{}%
3750     \fi
3751     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3752 \fi
3753 <<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.

```

3754 \def\process@line#1#2 #3 #4 {%
3755   \ifx=#1%
3756     \process@synonym{#2}%
3757   \else
3758     \process@language{#1#2}{#3}{#4}%
3759   \fi
3760   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bbl@languages` is also set to empty.

```

3761 \toks@{}
3762 \def\bbl@languages{}

```

When no languages have been loaded yet, the name following the = will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.)

Otherwise the name will be a synonym for the language loaded last.

We also need to copy the `hyphenmin` parameters for the synonym.

```

3763 \def\process@synonym#1{%
3764   \ifnum\last@language=\m@ne
3765     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3766   \else
3767     \expandafter\chardef\csname l@#1\endcsname\last@language
3768     \wlog{\string\l@#1=\string\language\the\last@language}%
3769     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3770       \csname\language\hyphenmins\endcsname
3771     \let\bbl@elt\relax
3772   \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
3773   \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.

```

3774 \def\process@language#1#2#3{%
3775   \expandafter\addlanguage\csname l@#1\endcsname
3776   \expandafter\language\csname l@#1\endcsname
3777   \edef\language{#1}%

```

```

3778 \bbl@hook@everylanguage{#1}%
3779 % > luatex
3780 \bbl@get@enc#1::\@@@
3781 \begingroup
3782 \leftthyphenmin\m@ne
3783 \bbl@hook@loadpatterns{#2}%
3784 % > luatex
3785 \ifnum\leftthyphenmin=\m@ne
3786 \else
3787 \expandafter\xdef\csname #1hyphenmins\endcsname{%
3788 \the\leftthyphenmin\the\rightthyphenmin}%
3789 \fi
3790 \endgroup
3791 \def\bbl@tempa{#3}%
3792 \ifx\bbl@tempa\@empty\else
3793 \bbl@hook@loadexceptions{#3}%
3794 % > luatex
3795 \fi
3796 \let\bbl@elt\relax
3797 \edef\bbl@languages{%
3798 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3799 \ifnum\the\language=\z@
3800 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3801 \set@hyphenmins\tw@\thr@@\relax
3802 \else
3803 \expandafter\expandafter\expandafter\set@hyphenmins
3804 \csname #1hyphenmins\endcsname
3805 \fi
3806 \the\toks@
3807 \toks@{}}%
3808 \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.

```

3809 \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.

```

3810 \def\bbl@hook@everylanguage#1{}
3811 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3812 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3813 \def\bbl@hook@loadkernel#1{%
3814 \def\addlanguage{\alloc@9\language\chardef\@cclvi}%
3815 \def\adddialect##1##2{%
3816 \global\chardef##1##2\relax
3817 \wlog{\string##1 = a dialect from \string\language##2}}%
3818 \def\iflanguage##1{%
3819 \expandafter\ifx\csname l@##1\endcsname\relax
3820 \@nolanerr{##1}%
3821 \else
3822 \ifnum\csname l@##1\endcsname=\language
3823 \expandafter\expandafter\expandafter\@firstoftwo
3824 \else
3825 \expandafter\expandafter\expandafter\@secondoftwo
3826 \fi
3827 \fi}%
3828 \def\providehyphenmins##1##2{%

```

```

3829 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
3830 \namedef{##1hyphenmins}{##2}%
3831 \fi}%
3832 \def\set@hyphenmins##1##2{%
3833 \leftthyphenmin##1\relax
3834 \righthyphenmin##2\relax}%
3835 \def\selectlanguage{%
3836 \errhelp{Selecting a language requires a package supporting it}%
3837 \errmessage{Not loaded}}%
3838 \let\foreignlanguage\selectlanguage
3839 \let\otherlanguage\selectlanguage
3840 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
3841 \def\bbl@usehooks##1##2{% TODO. Temporary!!
3842 \def\setlocale{%
3843 \errhelp{Find an armchair, sit down and wait}%
3844 \errmessage{Not yet available}}%
3845 \let\uselocale\setlocale
3846 \let\locale\setlocale
3847 \let\selectlocale\setlocale
3848 \let\localename\setlocale
3849 \let\textlocale\setlocale
3850 \let\textlanguage\setlocale
3851 \let\languagegettext\setlocale}
3852 \begingroup
3853 \def\AddBabelHook#1#2{%
3854 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3855 \def\next{\toks1}%
3856 \else
3857 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3858 \fi
3859 \next}
3860 \ifx\directlua\@undefined
3861 \ifx\XeTeXinputencoding\@undefined\else
3862 \input xebabel.def
3863 \fi
3864 \else
3865 \input luababel.def
3866 \fi
3867 \openin1 = babel-\bbl@format.cfg
3868 \ifeof1
3869 \else
3870 \input babel-\bbl@format.cfg\relax
3871 \fi
3872 \closein1
3873 \endgroup
3874 \bbl@hook@loadkernel{switch.def}

```

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

```

3875 \openin1 = language.dat

```

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

```

3876 \def\language{english}%
3877 \ifeof1
3878 \message{I couldn't find the file language.dat,\space
3879 I will try the file hyphen.tex}
3880 \input hyphen.tex\relax
3881 \chardef\l@english\z@
3882 \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`.

```
3883 \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.

```
3884 \loop
3885   \endlinechar\m@ne
3886   \read1 to \bbl@line
3887   \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.

```
3888   \if T\ifeof1F\fi T\relax
3889   \ifx\bbl@line\@empty\else
3890     \edef\bbl@line{\bbl@line\space\space\space}%
3891     \expandafter\process@line\bbl@line\relax
3892   \fi
3893 \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.

```
3894 \begingroup
3895   \def\bbl@elt#1#2#3#4{%
3896     \global\language=#2\relax
3897     \gdef\language#1}%
3898   \def\bbl@elt##1##2##3##4{}}%
3899   \bbl@languages
3900 \endgroup
3901 \fi
3902 \closein1
```

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

```
3903 \if/\the\toks@/\else
3904   \errhelp{language.dat loads no language, only synonyms}
3905   \errmessage{Orphan language synonym}
3906 \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.

```
3907 \let\bbl@line\@undefined
3908 \let\process@line\@undefined
3909 \let\process@synonym\@undefined
3910 \let\process@language\@undefined
3911 \let\bbl@get@enc\@undefined
3912 \let\bbl@hyph@enc\@undefined
3913 \let\bbl@tempa\@undefined
3914 \let\bbl@hook@loadkernel\@undefined
3915 \let\bbl@hook@everylanguage\@undefined
3916 \let\bbl@hook@loadpatterns\@undefined
3917 \let\bbl@hook@loadexceptions\@undefined
3918 \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].

```
3919 <<(*More package options)>> ≡
3920 \chardef\bbl@bidimode\z@
3921 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
3922 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
3923 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
3924 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
3925 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
3926 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
3927 <</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`.

```
3928 <<(*Font selection)>> ≡
3929 \bbl@trace{Font handling with fontspec}
3930 \@onlypreamble\babelfont
3931 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
3932   \bbl@foreach{#1}{\%
3933     \expandafter\ifx\csname date##1\endcsname\relax
3934       \IfFileExists{babel-##1.tex}%
3935         {\babelprovide{##1}}%
3936         {}%
3937     \fi}%
3938 \edef\bbl@tempa{#1}%
3939 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3940 \ifx\fontspec\undefined
3941   \usepackage{fontspec}%
3942 \fi
3943 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3944 \bbl@bblfont}
3945 \newcommand\bbl@bblfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
3946   \bbl@ifunset{\bbl@tempb family}%
3947   {\bbl@providedefam{\bbl@tempb}}%
3948   {\bbl@exp{\%
3949     \\\bbl@sreplace\<\bbl@tempb family >%
3950     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3951 % For the default font, just in case:
3952 \bbl@ifunset{\bbl@lsys\@languagename}{\bbl@provide@lsys{\@languagename}}}%
3953 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3954   {\bbl@csarg\edef{\bbl@tempb dflt@}{\<{#1}{#2}}}% save bbl@rmdflt@
3955   \bbl@exp{\%
3956     \let\<\bbl@\bbl@tempb dflt@\@languagename>\<\bbl@\bbl@tempb dflt@>%
3957     \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\@languagename>%
3958     \<\bbl@tempb default>\<\bbl@tempb family>}}}%
3959   {\bbl@foreach\bbl@tempa{\% ie bbl@rmdflt@lang / *scrt
3960     \bbl@csarg\def{\bbl@tempb dflt@##1}{\<{#1}{#2}}}}}%
3961 \def\bbl@providedefam#1{\%
3962   \bbl@exp{\%
3963     \\\newcommand\<#1default>{}% Just define it
3964     \\\bbl@add@list\<\bbl@font@fams{#1}%
3965     \\\DeclareRobustCommand\<#1family>{\%
```

If the family in the previous command does not exist, it must be defined. Here is how:

```
3961 \def\bbl@providedefam#1{\%
3962   \bbl@exp{\%
3963     \\\newcommand\<#1default>{}% Just define it
3964     \\\bbl@add@list\<\bbl@font@fams{#1}%
3965     \\\DeclareRobustCommand\<#1family>{\%
```

```

3966      \\\not@math@alphabet\<#1family>\relax
3967      \\\fontfamily\<#1default>\\selectfont}%
3968      \\\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.

```

3969 \def\bbl@nostdfont#1{%
3970   \bbl@ifunset{\bbl@WFF@\f@family}%
3971   {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
3972   \bbl@infowarn{The current font is not a babel standard family:\\%
3973   #1%
3974   \fontname\font\\%
3975   There is nothing intrinsically wrong with this warning, and\\%
3976   you can ignore it altogether if you do not need these\\%
3977   families. But if they are used in the document, you should be\\%
3978   aware 'babel' will no set Script and Language for them, so\\%
3979   you may consider defining a new family with \string\babelfont.\\%
3980   See the manual for further details about \string\babelfont.\\%
3981   Reported}}
3982   {}}%
3983 \gdef\bbl@switchfont{%
3984   \bbl@ifunset{\bbl@lsys@\language\name}{\bbl@provide@lsys{\language\name}}}%
3985   \bbl@exp{% eg Arabic -> arabic
3986   \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
3987   \bbl@foreach\bbl@font@fams{%
3988     \bbl@ifunset{\bbl@##1dflt@\language\name}% (1) language?
3989     {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
3990     {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
3991     {}}% 123=F - nothing!
3992     {\bbl@exp{% 3=T - from generic
3993       \global\let\<bbl@##1dflt@\language\name>%
3994       \<bbl@##1dflt@>}}}%
3995     {\bbl@exp{% 2=T - from script
3996       \global\let\<bbl@##1dflt@\language\name>%
3997       \<bbl@##1dflt@*\bbl@tempa>}}}%
3998     {}}% 1=T - language, already defined
3999   \def\bbl@tempa{\bbl@nostdfont}}}%
4000   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4001     \bbl@ifunset{\bbl@##1dflt@\language\name}%
4002     {\bbl@cs{famrst@##1}%
4003     \global\bbl@csarg\let{famrst@##1}\relax}%
4004     {\bbl@exp{% order is relevant
4005       \\\bbl@add\\originalTeX{%
4006       \\\bbl@font@rst{\bbl@cl{##1dflt}}}%
4007       \<##1default>\<##1family>{##1}}}%
4008       \\\bbl@font@set\<bbl@##1dflt@\language\name>% the main part!
4009       \<##1default>\<##1family>}}}%
4010   \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.

```

4011 \ifx\f@family\undefined\else % if latex
4012   \ifcase\bbl@engine % if pdftex
4013     \let\bbl@ckeckstdfonts\relax
4014   \else
4015     \def\bbl@ckeckstdfonts{%
4016       \begingroup
4017       \global\let\bbl@ckeckstdfonts\relax
4018       \let\bbl@tempa\@empty

```

```

4019 \bbl@foreach\bbl@font@fams{%
4020 \bbl@ifunset\bbl@##1dflt@}%
4021 {\@nameuse{##1family}}%
4022 \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4023 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \f@family\\}%
4024 \space\space\fontname\font\\}%
4025 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4026 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4027 }%
4028 \ifx\bbl@tempa\@empty\else
4029 \bbl@infowarn{The following font families will use the default\\%
4030 settings for all or some languages:\\%
4031 \bbl@tempa
4032 There is nothing intrinsically wrong with it, but\\%
4033 'babel' will no set Script and Language, which could\\%
4034 be relevant in some languages. If your document uses\\%
4035 these families, consider redefining them with \string\babelfont.\\%
4036 Reported}%
4037 \fi
4038 \endgroup}
4039 \fi
4040 \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.

```

4041 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4042 \bbl@xin@{<>}{#1}%
4043 \ifin@
4044 \bbl@exp{\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4045 \fi
4046 \bbl@exp{%
4047 \def\\#2#1% eg, \rmdefault{\bbl@rmdflt@lang}
4048 \\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\bbl@tempa\relax}}}%
4049 % TODO - next should be global?, but even local does its job. I'm
4050 % still not sure -- must investigate:
4051 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4052 \let\bbl@tempe\bbl@mapselect
4053 \let\bbl@mapselect\relax
4054 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4055 \let#4\@empty % Make sure \renewfontfamily is valid
4056 \bbl@exp{%
4057 \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4058 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4059 {\newfontscript\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4060 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4061 {\newfontlanguage\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4062 \\renewfontfamily\\#4%
4063 [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4064 \begingroup
4065 #4%
4066 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4067 \endgroup
4068 \let#4\bbl@temp@fam
4069 \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4070 \let\bbl@mapselect\bbl@tempe}%

```

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

```
4071 \def\bbbl@font@rst#1#2#3#4{%
4072   \bbbl@csarg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}
```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```
4073 \def\bbbl@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 :-).

```
4074 \newcommand\babelFSstore[2][]{%
4075   \bbbl@ifblank{#1}%
4076   {\bbbl@csarg\def{sname@#2}{Latin}}%
4077   {\bbbl@csarg\def{sname@#2}{#1}}%
4078   \bbbl@provide@dirs{#2}%
4079   \bbbl@csarg\ifnum{wdir@#2}>\z@
4080     \let\bbbl@beforeforeign\leavevmode
4081     \EnableBabelHook{babel-bidi}%
4082   \fi
4083   \bbbl@foreach{#2}{%
4084     \bbbl@FSstore{##1}{rm}\rmdefault\bbbl@save@rmdefault
4085     \bbbl@FSstore{##1}{sf}\sfdefault\bbbl@save@sfdefault
4086     \bbbl@FSstore{##1}{tt}\ttdefault\bbbl@save@ttdefault}}
4087 \def\bbbl@FSstore#1#2#3#4{%
4088   \bbbl@csarg\edef{#2default#1}{#3}%
4089   \expandafter\addto\csname extras#1\endcsname{%
4090     \let#4#3%
4091     \ifx#3\f@family
4092       \edef#3{\csname bbl@#2default#1\endcsname}%
4093       \fontfamily{#3}\selectfont
4094     \else
4095       \edef#3{\csname bbl@#2default#1\endcsname}%
4096       \fi}%
4097   \expandafter\addto\csname noextras#1\endcsname{%
4098     \ifx#3\f@family
4099       \fontfamily{#4}\selectfont
4100     \fi
4101     \let#3#4}}
4102 \let\bbbl@langfeatures\@empty
4103 \def\babelFSfeatures{% make sure \fontspec is redefined once
4104   \let\bbbl@ori@fontspec\fontspec
4105   \renewcommand\fontspec[1][]{%
4106     \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
4107   \let\babelFSfeatures\bbbl@FSfeatures
4108   \babelFSfeatures}
4109 \def\bbbl@FSfeatures#1#2{%
4110   \expandafter\addto\csname extras#1\endcsname{%
4111     \babel@save\bbbl@langfeatures
4112     \edef\bbbl@langfeatures{#2,}}
4113 <</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.

```
4114 <<{*Footnote changes}>> ≡
4115 \bbl@trace{Bidi footnotes}
4116 \ifnum\bbl@bidimode>\z@
4117   \def\bbl@footnote#1#2#3{%
4118     \ifnextchar[%
4119       {\bbl@footnote@o{#1}{#2}{#3}}%
4120       {\bbl@footnote@x{#1}{#2}{#3}}}
4121   \def\bbl@footnote@x#1#2#3#4{%
4122     \bgroup
4123     \select@language@x{\bbl@main@language}%
4124     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4125     \egroup}
4126   \def\bbl@footnote@o#1#2#3[#4]#5{%
4127     \bgroup
4128     \select@language@x{\bbl@main@language}%
4129     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4130     \egroup}
4131   \def\bbl@footnotetext#1#2#3{%
4132     \ifnextchar[%
4133       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4134       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4135   \def\bbl@footnotetext@x#1#2#3#4{%
4136     \bgroup
4137     \select@language@x{\bbl@main@language}%
4138     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4139     \egroup}
4140   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4141     \bgroup
4142     \select@language@x{\bbl@main@language}%
4143     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4144     \egroup}
4145   \def\BabelFootnote#1#2#3#4{%
4146     \ifx\bbl@fn@footnote\@undefined
4147       \let\bbl@fn@footnote\footnote
4148     \fi
4149     \ifx\bbl@fn@footnotetext\@undefined
4150       \let\bbl@fn@footnotetext\footnotetext
4151     \fi
4152     \bbl@ifblank{#2}%
4153     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4154      \@namedef{\bbl@stripslash#1text}%
4155      {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4156     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4157      \@namedef{\bbl@stripslash#1text}%
4158      {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4159   \fi
4160 <</Footnote changes>>
```

Now, the code.

```
4161 <{*xetex}>
4162 \def\BabelStringsDefault{unicode}
4163 \let\xebbl@stop\relax
```

```

4164 \AddBabelHook{xetex}{encodedcommands}{%
4165   \def\bbl@tempa{#1}%
4166   \ifx\bbl@tempa@empty
4167     \XeTeXinputencoding"bytes"%
4168   \else
4169     \XeTeXinputencoding"#1"%
4170   \fi
4171   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4172 \AddBabelHook{xetex}{stopcommands}{%
4173   \xebbl@stop
4174   \let\xebbl@stop\relax}
4175 \def\bbl@intraspace#1 #2 #3@@{%
4176   \bbl@csarg\gdef{xeisp@\language}%
4177   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4178 \def\bbl@intrapenalty#1@@{%
4179   \bbl@csarg\gdef{xeipn@\language}%
4180   {\XeTeXlinebreakpenalty #1\relax}}
4181 \def\bbl@provide@intraspace{%
4182   \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4183   \ifin@else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4184   \ifin@
4185     \bbl@ifunset{\bbl@intsp@\language}{}%
4186     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4187       \ifx\bbl@KVP@intraspace\@nil
4188         \bbl@exp{%
4189           \\bbl@intraspace\bbl@cl{intsp}\\\\@}%
4190         \fi
4191         \ifx\bbl@KVP@intrapenalty\@nil
4192           \bbl@intrapenalty0@@
4193         \fi
4194       \fi
4195       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4196         \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4197       \fi
4198       \ifx\bbl@KVP@intrapenalty\@nil\else
4199         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4200       \fi
4201       \bbl@exp{%
4202         \\bbl@add\<extras\language>{%
4203           \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
4204           \<bbl@xeisp@\language>%
4205           \<bbl@xeipn@\language>%
4206           \\bbl@toglobal\<extras\language>%
4207           \\bbl@add\<noextras\language>{%
4208             \XeTeXlinebreaklocale "en"%
4209             \\bbl@toglobal\<noextras\language>}}%
4210       \ifx\bbl@ispace\undefined
4211         \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4212       \ifx\AtBeginDocument\@notprerr
4213         \expandafter\@secondoftwo % to execute right now
4214       \fi
4215       \AtBeginDocument{%
4216         \expandafter\bbl@add
4217         \csname selectfont \endcsname{\bbl@ispace}%
4218         \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4219       \fi}%
4220   \fi}
4221 \ifx\DisableBabelHook\undefined\endinput\fi
4222 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}

```

```

4223 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
4224 \DisableBabelHook{babel-fontspec}
4225 <<Font selection>>
4226 \input txtbabel.def
4227 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titlesp, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T_EX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdfTeX and xetex.

```

4228 (*texxet)
4229 \providecommand\bbl@provide@intraspace{}
4230 \bbl@trace{Redefinitions for bidi layout}
4231 \def\bbl@sspre@caption{%
4232   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4233 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4234 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4235 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4236 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4237   \def\hangfrom#1{%
4238     \setbox\@tempboxa\hbox{#1}%
4239     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4240     \noindent\box\@tempboxa}
4241 \def\raggedright{%
4242   \let\@centercr
4243   \bbl@startskip\z@skip
4244   \@rightskip\@flushglue
4245   \bbl@endskip\@rightskip
4246   \parindent\z@
4247   \parfillskip\bbl@startskip}
4248 \def\raggedleft{%
4249   \let\@centercr
4250   \bbl@startskip\@flushglue
4251   \bbl@endskip\z@skip
4252   \parindent\z@
4253   \parfillskip\bbl@endskip}
4254 \fi
4255 \IfBabelLayout{lists}
4256   {\bbl@sreplace\list
4257     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4258     \def\bbl@listleftmargin{%
4259       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4260     \ifcase\bbl@engine
4261       \def\labelenumii{}\theenumii{}\% pdfTeX doesn't reverse ()
4262       \def\p@enumiii{\p@enumii}\theenumii{}\%
4263     \fi
4264     \bbl@sreplace\@verbatim
4265       {\leftskip\@totalleftmargin}%
4266       {\bbl@startskip\textwidth
4267         \advance\bbl@startskip-\linewidth}%
4268     \bbl@sreplace\@verbatim

```



```

4269     {\rightskip\z@skip}%
4270     {\bbl@endskip\z@skip}}%
4271 {}
4272 \IfBabelLayout{contents}
4273   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4274    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4275 {}
4276 \IfBabelLayout{columns}
4277   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4278    \def\bbl@outputbox#1{%
4279      \hb@xt@\textwidth{%
4280        \hskip\columnwidth
4281        \hfil
4282        {\normalcolor\vrule \@width\columnseprule}%
4283        \hfil
4284        \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4285        \hskip-\textwidth
4286        \hb@xt@\columnwidth{\box\@outputbox \hss}%
4287        \hskip\columnsep
4288        \hskip\columnwidth}}}%
4289 {}
4290 <<Footnote changes>>
4291 \IfBabelLayout{footnotes}%
4292   {\BabelFootnote\footnote\language{}{}}%
4293   \BabelFootnote\localfootnote\language{}{}}%
4294   \BabelFootnote\mainfootnote{}{}}{}
4295 {}

```

Implicitly reverses sectioning labels in `bidibasic`, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

4296 \IfBabelLayout{counters}%
4297   {\let\bbl@latinarabic=\@arabic
4298    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4299    \let\bbl@asciroman=\@roman
4300    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4301    \let\bbl@asciiRoman=\@Roman
4302    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4303 </texxet>

```

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, they are added, but note if the language patterns have not been preloaded they won't at run time.

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

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

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

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). 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).

```

4304 (*luatex)
4305 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4306 \bbl@trace{Read language.dat}
4307 \ifx\bbl@readstream\undefined
4308   \csname newread\endcsname\bbl@readstream
4309 \fi
4310 \begingroup
4311   \toks@{}
4312   \count@ \z@ % 0=start, 1=0th, 2=normal
4313   \def\bbl@process@line#1#2 #3 #4 {%
4314     \ifx=#1%
4315       \bbl@process@synonym{#2}%
4316     \else
4317       \bbl@process@language{#1#2}{#3}{#4}%
4318     \fi
4319     \ignorespaces}
4320   \def\bbl@manylang{%
4321     \ifnum\bbl@last>\@ne
4322       \bbl@info{Non-standard hyphenation setup}%
4323     \fi
4324     \let\bbl@manylang\relax}
4325   \def\bbl@process@language#1#2#3{%
4326     \ifcase\count@
4327       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4328     \or
4329       \count@\tw@
4330     \fi
4331     \ifnum\count@=\tw@
4332       \expandafter\addlanguage\csname l@#1\endcsname
4333       \language\allocationnumber
4334       \chardef\bbl@last\allocationnumber
4335       \bbl@manylang
4336       \let\bbl@elt\relax
4337       \xdef\bbl@languages{%
4338         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4339     \fi
4340     \the\toks@
4341     \toks@{}}

```

```

4342 \def\bbl@process@synonym@aux#1#2{%
4343   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4344   \let\bbl@elt\relax
4345   \xdef\bbl@languages{%
4346     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4347 \def\bbl@process@synonym#1{%
4348   \ifcase\count@
4349     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4350   \or
4351     \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4352   \else
4353     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4354   \fi}
4355 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4356   \chardef\l@english\z@
4357   \chardef\l@USenglish\z@
4358   \chardef\bbl@last\z@
4359   \global\@namedef{bbl@hyphendata@0}{\hyphen.tex{}}
4360   \gdef\bbl@languages{%
4361     \bbl@elt{english}{0}{\hyphen.tex{}}%
4362     \bbl@elt{USenglish}{0}{}}%
4363 \else
4364   \global\let\bbl@languages@format\bbl@languages
4365   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4366     \ifnum#2>\z@\else
4367       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4368     \fi}%
4369   \xdef\bbl@languages{\bbl@languages}%
4370 \fi
4371 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4372 \bbl@languages
4373 \openin\bbl@readstream=language.dat
4374 \ifeof\bbl@readstream
4375   \bbl@warning{I couldn't find language.dat. No additional\\%
4376     patterns loaded. Reported}%
4377 \else
4378   \loop
4379     \endlinechar\m@ne
4380     \read\bbl@readstream to \bbl@line
4381     \endlinechar\^^M
4382     \if T\ifeof\bbl@readstream F\fi T\relax
4383     \ifx\bbl@line\@empty\else
4384       \edef\bbl@line{\bbl@line\space\space\space}%
4385       \expandafter\bbl@process@line\bbl@line\relax
4386     \fi
4387   \repeat
4388 \fi
4389 \endgroup
4390 \bbl@trace{Macros for reading patterns files}
4391 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4392 \ifx\babelcatcodetablenum\@undefined
4393   \ifx\newcatcodetable\@undefined
4394     \def\babelcatcodetablenum{5211}
4395     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4396   \else
4397     \newcatcodetable\babelcatcodetablenum
4398     \newcatcodetable\bbl@pattcodes
4399   \fi
4400 \else

```

```

4401 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4402 \fi
4403 \def\bbl@luapatterns#1#2{%
4404   \bbl@get@enc#1::\@@@
4405   \setbox\z@\hbox\bgroup
4406     \begingroup
4407       \savecatcodetable\babelcatcodetablenum\relax
4408       \initcatcodetable\bbl@pattcodes\relax
4409       \catcodetable\bbl@pattcodes\relax
4410       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4411       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4412       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4413       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4414       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4415       \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4416       \input #1\relax
4417       \catcodetable\babelcatcodetablenum\relax
4418     \endgroup
4419   \def\bbl@tempa{#2}%
4420   \ifx\bbl@tempa@empty\else
4421     \input #2\relax
4422   \fi
4423 \egroup}%
4424 \def\bbl@patterns@lua#1{%
4425   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4426     \csname l@#1\endcsname
4427     \edef\bbl@tempa{#1}%
4428   \else
4429     \csname l@#1:\f@encoding\endcsname
4430     \edef\bbl@tempa{#1:\f@encoding}%
4431   \fi\relax
4432   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4433   \@ifundefined{bbl@hyphendata@the\language}%
4434   {\def\bbl@elt##1##2##3##4{%
4435     \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4436     \def\bbl@tempb{##3}%
4437     \ifx\bbl@tempb@empty\else % if not a synonymous
4438       \def\bbl@tempc{##3}{##4}}%
4439     \fi
4440     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4441     \fi}%
4442   \bbl@languages
4443   \@ifundefined{bbl@hyphendata@the\language}%
4444   {\bbl@info{No hyphenation patterns were set for\%
4445     language '\bbl@tempa'. Reported}}%
4446   {\expandafter\expandafter\expandafter\bbl@luapatterns
4447     \csname bbl@hyphendata@the\language\endcsname}}}%
4448 \endinput\fi
4449 % Here ends \ifx\AddBabelHook@undefined
4450 % A few lines are only read by hyphen.cfg
4451 \ifx\DisableBabelHook@undefined
4452   \AddBabelHook{luatex}{everylanguage}{%
4453     \def\process@language##1##2##3{%
4454       \def\process@line####1####2 ####3 ####4 {}}%
4455   \AddBabelHook{luatex}{loadpatterns}{%
4456     \input #1\relax
4457     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4458       {##1}{}}%
4459   \AddBabelHook{luatex}{loadexceptions}{%

```

```

4460 \input #1\relax
4461 \def\bbl@tempb##1##2{{##1}{##2}}%
4462 \expandafter\def\csname bbl@hyphendata@the\language\endcsname
4463 {\expandafter\expandafter\expandafter\bbl@tempb
4464 \csname bbl@hyphendata@the\language\endcsname}}
4465 \endinput\fi
4466 % Here stops reading code for hyphen.cfg
4467 % The following is read the 2nd time it's loaded
4468 \begingroup
4469 \catcode`\%=12
4470 \catcode`\'=12
4471 \catcode`\%=12
4472 \catcode`\:=12
4473 \directlua{
4474   Babel = Babel or {}
4475   function Babel.bytes(line)
4476     return line:gsub(".",
4477       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4478   end
4479   function Babel.begin_process_input()
4480     if luatexbase and luatexbase.add_to_callback then
4481       luatexbase.add_to_callback('process_input_buffer',
4482         Babel.bytes, 'Babel.bytes')
4483     else
4484       Babel.callback = callback.find('process_input_buffer')
4485       callback.register('process_input_buffer', Babel.bytes)
4486     end
4487   end
4488   function Babel.end_process_input ()
4489     if luatexbase and luatexbase.remove_from_callback then
4490       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4491     else
4492       callback.register('process_input_buffer', Babel.callback)
4493     end
4494   end
4495   function Babel.addpatterns(pp, lg)
4496     local lg = lang.new(lg)
4497     local pats = lang.patterns(lg) or ''
4498     lang.clear_patterns(lg)
4499     for p in pp:gmatch('[^%s]+') do
4500       ss = ''
4501       for i in string.utfcharacters(p:gsub('%d', '')) do
4502         ss = ss .. '%d?' .. i
4503       end
4504       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4505       ss = ss:gsub('%.%%d%?$', '%%.')
4506       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4507       if n == 0 then
4508         tex.sprint(
4509           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4510           .. p .. [[]])
4511         pats = pats .. ' ' .. p
4512       else
4513         tex.sprint(
4514           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4515           .. p .. [[]])
4516       end
4517     end
4518     lang.patterns(lg, pats)

```

```

4519 end
4520 }
4521 \endgroup
4522 \ifx\newattribute\undefined\else
4523   \newattribute\bbbl@attr@locale
4524   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbbl@attr@locale'}
4525   \AddBabelHook{luatex}{beforeextras}{%
4526     \setattribute\bbbl@attr@locale\localeid}
4527 \fi
4528 \def\BabelStringsDefault{unicode}
4529 \let\luabbl@stop\relax
4530 \AddBabelHook{luatex}{encodedcommands}{%
4531   \def\bbbl@tempa{utf8}\def\bbbl@tempb{#1}%
4532   \ifx\bbbl@tempa\bbbl@tempb\else
4533     \directlua{Babel.begin_process_input()}%
4534     \def\luabbl@stop{%
4535       \directlua{Babel.end_process_input()}}%
4536   \fi}%
4537 \AddBabelHook{luatex}{stopcommands}{%
4538   \luabbl@stop
4539   \let\luabbl@stop\relax}
4540 \AddBabelHook{luatex}{patterns}{%
4541   \@ifundefined{bbbl@hyphendata@the\language}%
4542   {\def\bbbl@elt##1##2##3##4{%
4543     \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4544     \def\bbbl@tempb{##3}%
4545     \ifx\bbbl@tempb\empty\else % if not a synonymous
4546       \def\bbbl@tempc{##3}{##4}%
4547     \fi
4548     \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
4549     \fi}%
4550   \bbbl@languages
4551   \@ifundefined{bbbl@hyphendata@the\language}%
4552   {\bbbl@info{No hyphenation patterns were set for\%
4553     language '#2'. Reported}}%
4554   {\expandafter\expandafter\expandafter\bbbl@luapatterns
4555     \csname bbbl@hyphendata@the\language\endcsname}}}%
4556 \@ifundefined{bbbl@patterns@}{}%
4557 \begingroup
4558   \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
4559   \ifin\else
4560     \ifx\bbbl@patterns@\empty\else
4561       \directlua{ Babel.addpatterns(
4562         [[\bbbl@patterns@]], \number\language) }%
4563     \fi
4564     \@ifundefined{bbbl@patterns@#1}%
4565     {\empty
4566       {\directlua{ Babel.addpatterns(
4567         [[\space\csname bbbl@patterns@#1\endcsname]],
4568         \number\language) }}%
4569       \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
4570     \fi
4571   \endgroup}%
4572 \bbbl@exp{%
4573   \bbbl@ifunset{bbbl@prehc\languagename}{}%
4574   {\bbbl@ifblank{\bbbl@cs{prehc\languagename}}}%
4575   {\prehyphenchar=\bbbl@c1{prehc}\relax}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbbl@patterns@` for the

global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4576 \@onlypreamble\babelpatterns
4577 \AtEndOfPackage{%
4578   \newcommand\babelpatterns[2][\@empty]{%
4579     \ifx\bbl@patterns@relax
4580       \let\bbl@patterns@\@empty
4581     \fi
4582     \ifx\bbl@pttnlist\@empty\else
4583       \bbl@warning{%
4584         You must not intermingle \string\selectlanguage\space and\%
4585         \string\babelpatterns\space or some patterns will not\%
4586         be taken into account. Reported}%
4587     \fi
4588     \ifx\@empty#1%
4589       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4590     \else
4591       \edef\bbl@tempb{\zap@space#1 \@empty}%
4592       \bbl@for\bbl@tempa\bbl@tempb{%
4593         \bbl@fixname\bbl@tempa
4594         \bbl@iflanguage\bbl@tempa{%
4595           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4596             \ifundefined\bbl@patterns@\bbl@tempa}%
4597             \@empty
4598             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4599             #2}}}%
4600     \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).

```

4601 \directlua{
4602   Babel = Babel or {}
4603   Babel.linebreaking = Babel.linebreaking or {}
4604   Babel.linebreaking.before = {}
4605   Babel.linebreaking.after = {}
4606   Babel.locale = {} % Free to use, indexed with \localeid
4607   function Babel.linebreaking.add_before(func)
4608     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4609     table.insert(Babel.linebreaking.before , func)
4610   end
4611   function Babel.linebreaking.add_after(func)
4612     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4613     table.insert(Babel.linebreaking.after, func)
4614   end
4615 }
4616 \def\bbl@intraspace#1 #2 #3\@@{%
4617   \directlua{
4618     Babel = Babel or {}
4619     Babel.intraspaces = Babel.intraspaces or {}
4620     Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
4621       {b = #1, p = #2, m = #3}
4622     Babel.locale_props[\the\localeid].intraspace = %

```

```

4623      {b = #1, p = #2, m = #3}
4624  }}
4625  \def\bbl@intrapenalty#1\@@{%
4626    \directlua{
4627      Babel = Babel or {}
4628      Babel.intrapenalties = Babel.intrapenalties or {}
4629      Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4630      Babel.locale_props[\the\localeid].intrapenalty = #1
4631    }}
4632  \begingroup
4633  \catcode`\%=12
4634  \catcode`\^=14
4635  \catcode`\'=12
4636  \catcode`\~=12
4637  \gdef\bbl@seaintraspace{^
4638    \let\bbl@seaintraspace\relax
4639    \directlua{
4640      Babel = Babel or {}
4641      Babel.sea_enabled = true
4642      Babel.sea_ranges = Babel.sea_ranges or {}
4643      function Babel.set_chranges (script, chrng)
4644        local c = 0
4645        for s, e in string.gmatch(chrng..' ', '(-)%%.(-)%s') do
4646          Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4647          c = c + 1
4648        end
4649      end
4650      function Babel.sea_disc_to_space (head)
4651        local sea_ranges = Babel.sea_ranges
4652        local last_char = nil
4653        local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4654        for item in node.traverse(head) do
4655          local i = item.id
4656          if i == node.id'glyph' then
4657            last_char = item
4658          elseif i == 7 and item.subtype == 3 and last_char
4659            and last_char.char > 0x0C99 then
4660            quad = font.getfont(last_char.font).size
4661            for lg, rg in pairs(sea_ranges) do
4662              if last_char.char > rg[1] and last_char.char < rg[2] then
4663                lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyl1
4664                local intraspace = Babel.intraspaces[lg]
4665                local intrapenalty = Babel.intrapenalties[lg]
4666                local n
4667                if intrapenalty ~= 0 then
4668                  n = node.new(14, 0) ^^ penalty
4669                  n.penalty = intrapenalty
4670                  node.insert_before(head, item, n)
4671                end
4672                n = node.new(12, 13) ^^ (glue, spaceskip)
4673                node.setglue(n, intraspace.b * quad,
4674                  intraspace.p * quad,
4675                  intraspace.m * quad)
4676                node.insert_before(head, item, n)
4677                node.remove(head, item)
4678              end
4679            end
4680          end
4681        end

```



```

4682     end
4683 }^^
4684 \bbl@luahyphenate}
4685 \catcode`\%=14
4686 \gdef\bbl@cjkintraspacespace{%
4687   \let\bbl@cjkintraspacespace\relax
4688   \directlua{
4689     Babel = Babel or {}
4690     require'babel-data-cjk.lua'
4691     Babel.cjk_enabled = true
4692     function Babel.cjk_linebreak(head)
4693       local GLYPH = node.id'glyph'
4694       local last_char = nil
4695       local quad = 655360      % 10 pt = 655360 = 10 * 65536
4696       local last_class = nil
4697       local last_lang = nil
4698
4699       for item in node.traverse(head) do
4700         if item.id == GLYPH then
4701
4702           local lang = item.lang
4703
4704           local LOCALE = node.get_attribute(item,
4705             luatexbase.registernumber'bbl@attr@locale')
4706           local props = Babel.locale_props[LOCALE]
4707
4708           local class = Babel.cjk_class[item.char].c
4709
4710           if class == 'cp' then class = 'cl' end % ]) as CL
4711           if class == 'id' then class = 'I' end
4712
4713           local br = 0
4714           if class and last_class and Babel.cjk_breaks[last_class][class] then
4715             br = Babel.cjk_breaks[last_class][class]
4716           end
4717
4718           if br == 1 and props.linebreak == 'c' and
4719             lang ~= \the\l@nohyphenation\space and
4720             last_lang ~= \the\l@nohyphenation then
4721             local intrapenalty = props.intrapenalty
4722             if intrapenalty ~= 0 then
4723               local n = node.new(14, 0)      % penalty
4724               n.penalty = intrapenalty
4725               node.insert_before(head, item, n)
4726             end
4727             local intraspacespace = props.intraspacespace
4728             local n = node.new(12, 13)      % (glue, spaceskip)
4729             node.setglue(n, intraspacespace.b * quad,
4730               intraspacespace.p * quad,
4731               intraspacespace.m * quad)
4732             node.insert_before(head, item, n)
4733           end
4734
4735           quad = font.getfont(item.font).size
4736           last_class = class
4737           last_lang = lang
4738         else % if penalty, glue or anything else
4739           last_class = nil
4740         end

```

```

4741     end
4742     lang.hyphenate(head)
4743 end
4744 }%
4745 \bbl@luahyphenate}
4746 \gdef\bbl@luahyphenate{%
4747 \let\bbl@luahyphenate\relax
4748 \directlua{
4749   luatexbase.add_to_callback('hyphenate',
4750   function (head, tail)
4751     if Babel.linebreaking.before then
4752       for k, func in ipairs(Babel.linebreaking.before) do
4753         func(head)
4754       end
4755     end
4756     if Babel.cjk_enabled then
4757       Babel.cjk_linebreak(head)
4758     end
4759     lang.hyphenate(head)
4760     if Babel.linebreaking.after then
4761       for k, func in ipairs(Babel.linebreaking.after) do
4762         func(head)
4763       end
4764     end
4765     if Babel.sea_enabled then
4766       Babel.sea_disc_to_space(head)
4767     end
4768   end,
4769   'Babel.hyphenate')
4770 }
4771 }
4772 \endgroup
4773 \def\bbl@provide@intraspace{%
4774 \bbl@ifunset{\bbl@intsp@language}{}%
4775 {\expandafter\ifx\csname\bbl@intsp@language\endcsname\@empty\else
4776 \bbl@xin@{\bbl@cl{lbrk}}{c}%
4777 \ifin@ % cjk
4778 \bbl@cjkintraspace
4779 \directlua{
4780   Babel = Babel or {}
4781   Babel.locale_props = Babel.locale_props or {}
4782   Babel.locale_props[\the\localeid].linebreak = 'c'
4783 }%
4784 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4785 \ifx\bbl@KVP@intrapenalty\@nil
4786 \bbl@intrapenalty0\@
4787 \fi
4788 \else % sea
4789 \bbl@seaintraspace
4790 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4791 \directlua{
4792   Babel = Babel or {}
4793   Babel.sea_ranges = Babel.sea_ranges or {}
4794   Babel.set_chranges('\bbl@cl{sbc}',
4795                       '\bbl@cl{chrng}')
4796 }%
4797 \ifx\bbl@KVP@intrapenalty\@nil
4798 \bbl@intrapenalty0\@
4799 \fi

```

```

4800     \fi
4801     \fi
4802     \ifx\bbbl@KVP@intrapenalty\@nil\else
4803         \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
4804     \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.

```

4805 \AddBabelHook{babel-fontspec}{afterextras}{\bbbl@switchfont}
4806 \AddBabelHook{babel-fontspec}{beforestart}{\bbbl@ckeckstdfonts}
4807 \DisableBabelHook{babel-fontspec}
4808 <<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.

```

4809 \directlua{
4810 Babel.script_blocks = {
4811   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4812             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4813   ['Armn'] = {{0x0530, 0x058F}},
4814   ['Beng'] = {{0x0980, 0x09FF}},
4815   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
4816   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4817   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4818             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4819   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4820   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4821             {0xAB00, 0xAB2F}},
4822   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4823   % Don't follow strictly Unicode, which places some Coptic letters in
4824   % the 'Greek and Coptic' block
4825   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4826   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4827             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4828             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4829             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4830             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4831             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4832   ['Hebr'] = {{0x0590, 0x05FF}},

```

```

4833 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4834             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4835 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4836 ['Knda'] = {{0x0C80, 0x0CFF}},
4837 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4838             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4839             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4840 ['Lao'] = {{0x0E80, 0x0EFF}},
4841 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4842             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4843             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4844 ['Mahj'] = {{0x11150, 0x1117F}},
4845 ['Mlym'] = {{0x0D00, 0x0D7F}},
4846 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4847 ['Orya'] = {{0x0B00, 0x0B7F}},
4848 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4849 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
4850 ['Taml'] = {{0x0B80, 0x0BFF}},
4851 ['Telu'] = {{0x0C00, 0x0C7F}},
4852 ['Tfng'] = {{0x2D30, 0x2D7F}},
4853 ['Thai'] = {{0x0E00, 0x0E7F}},
4854 ['Tibt'] = {{0x0F00, 0x0FFF}},
4855 ['Vaii'] = {{0xA500, 0xA63F}},
4856 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4857 }
4858
4859 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
4860 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4861 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4862
4863 function Babel.locale_map(head)
4864   if not Babel.locale_mapped then return head end
4865
4866   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
4867   local GLYPH = node.id('glyph')
4868   local inmath = false
4869   local toloc_save
4870   for item in node.traverse(head) do
4871     local toloc
4872     if not inmath and item.id == GLYPH then
4873       % Optimization: build a table with the chars found
4874       if Babel.chr_to_loc[item.char] then
4875         toloc = Babel.chr_to_loc[item.char]
4876       else
4877         for lc, maps in pairs(Babel.loc_to_scr) do
4878           for _, rg in pairs(maps) do
4879             if item.char >= rg[1] and item.char <= rg[2] then
4880               Babel.chr_to_loc[item.char] = lc
4881               toloc = lc
4882               break
4883             end
4884           end
4885         end
4886       end
4887       % Now, take action, but treat composite chars in a different
4888       % fashion, because they 'inherit' the previous locale. Not yet
4889       % optimized.
4890       if not toloc and
4891         (item.char >= 0x0300 and item.char <= 0x036F) or

```

```

4892         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4893         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
4894         toloc = toloc_save
4895     end
4896     if toloc and toloc > -1 then
4897         if Babel.locale_props[toloc].lg then
4898             item.lang = Babel.locale_props[toloc].lg
4899             node.set_attribute(item, LOCALE, toloc)
4900         end
4901         if Babel.locale_props[toloc]['/'..item.font] then
4902             item.font = Babel.locale_props[toloc]['/'..item.font]
4903         end
4904         toloc_save = toloc
4905     end
4906     elseif not inmath and item.id == 7 then
4907         item.replace = item.replace and Babel.locale_map(item.replace)
4908         item.pre      = item.pre and Babel.locale_map(item.pre)
4909         item.post      = item.post and Babel.locale_map(item.post)
4910     elseif item.id == node.id'math' then
4911         inmath = (item.subtype == 0)
4912     end
4913 end
4914 return head
4915 end
4916 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

4917 \newcommand\babelcharproperty[1]{%
4918   \count@=#1\relax
4919   \ifvmode
4920     \expandafter\bbl@chprop
4921   \else
4922     \bbl@error{\string\babelcharproperty\space can be used only in\%
4923       vertical mode (preamble or between paragraphs)}%
4924     {See the manual for futher info}%
4925   \fi}
4926 \newcommand\bbl@chprop[3][\the\count@]{%
4927   \@tempcnta=#1\relax
4928   \bbl@ifunset{\bbl@chprop@#2}%
4929   {\bbl@error{No property named '#2'. Allowed values are\%
4930     direction (bc), mirror (bmg), and linebreak (lb)}%
4931     {See the manual for futher info}}%
4932   }%
4933   \loop
4934     \bbl@cs{chprop@#2}{#3}%
4935     \ifnum\count@<\@tempcnta
4936       \advance\count@\@ne
4937     \repeat}
4938 \def\bbl@chprop@direction#1{%
4939   \directlua{
4940     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4941     Babel.characters[\the\count@]['d'] = '#1'
4942   }}
4943 \let\bbl@chprop@bc\bbl@chprop@direction
4944 \def\bbl@chprop@mirror#1{%
4945   \directlua{
4946     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4947     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

4948 }}
4949 \let\bbl@chprop@bmg\bbl@chprop@mirror
4950 \def\bbl@chprop@linebreak#1{%
4951   \directlua{
4952     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
4953     Babel.cjk_characters[\the\count@]['c'] = '#1'
4954   }}
4955 \let\bbl@chprop@lb\bbl@chprop@linebreak
4956 \def\bbl@chprop@locale#1{%
4957   \directlua{
4958     Babel.chr_to_loc = Babel.chr_to_loc or {}
4959     Babel.chr_to_loc[\the\count@] =
4960       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
4961   }}

```

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.

```

4962 \begingroup
4963 \catcode`\#=12
4964 \catcode`\%=12
4965 \catcode`\&=14
4966 \directlua{
4967   Babel.linebreaking.post_replacements = {}
4968   Babel.linebreaking.pre_replacements = {}
4969
4970   function Babel.str_to_nodes(fn, matches, base)
4971     local n, head, last
4972     if fn == nil then return nil end
4973     for s in string.utfvalues(fn(matches)) do
4974       if base.id == 7 then
4975         base = base.replace
4976       end
4977       n = node.copy(base)
4978       n.char = s
4979       if not head then
4980         head = n
4981       else
4982         last.next = n
4983       end
4984       last = n
4985     end
4986     return head
4987   end
4988
4989   function Babel.fetch_word(head, funct)
4990     local word_string = ''
4991     local word_nodes = {}

```

```

4992     local lang
4993     local item = head
4994     local inmath = false
4995
4996     while item do
4997
4998         if item.id == 29
4999             and not(item.char == 124) && ie, not |
5000             and not(item.char == 61) && ie, not =
5001             and not inmath
5002             and (item.lang == lang or lang == nil) then
5003             lang = lang or item.lang
5004             word_string = word_string .. unicode.utf8.char(item.char)
5005             word_nodes[#word_nodes+1] = item
5006
5007         elseif item.id == 7 and item.subtype == 2 and not inmath then
5008             word_string = word_string .. '='
5009             word_nodes[#word_nodes+1] = item
5010
5011         elseif item.id == 7 and item.subtype == 3 and not inmath then
5012             word_string = word_string .. '|'
5013             word_nodes[#word_nodes+1] = item
5014
5015         elseif item.id == node.id'math' then
5016             inmath = (item.subtype == 0)
5017
5018         elseif word_string == '' then
5019             && pass
5020
5021         else
5022             return word_string, word_nodes, item, lang
5023         end
5024
5025         item = item.next
5026     end
5027 end
5028
5029 function Babel.post_hyphenate_replace(head)
5030     local u = unicode.utf8
5031     local lbkr = Babel.linebreaking.post_replacements
5032     local word_head = head
5033
5034     while true do
5035         local w, wn, nw, lang = Babel.fetch_word(word_head)
5036         if not lang then return head end
5037
5038         if not lbkr[lang] then
5039             break
5040         end
5041
5042         for k=1, #lbkr[lang] do
5043             local p = lbkr[lang][k].pattern
5044             local r = lbkr[lang][k].replace
5045
5046             while true do
5047                 local matches = { u.match(w, p) }
5048                 if #matches < 2 then break end
5049
5050                 local first = table.remove(matches, 1)

```

```

5051         local last = table.remove(matches, #matches)
5052
5053         %% Fix offsets, from bytes to unicode.
5054         first = u.len(w:sub(1, first-1)) + 1
5055         last = u.len(w:sub(1, last-1))
5056
5057         local new %% used when inserting and removing nodes
5058         local changed = 0
5059
5060         %% This loop traverses the replace list and takes the
5061         %% corresponding actions
5062         for q = first, last do
5063             local crep = r[q-first+1]
5064             local char_node = wn[q]
5065             local char_base = char_node
5066
5067             if crep and crep.data then
5068                 char_base = wn[crep.data+first-1]
5069             end
5070
5071             if crep == {} then
5072                 break
5073             elseif crep == nil then
5074                 changed = changed + 1
5075                 node.remove(head, char_node)
5076             elseif crep and (crep.pre or crep.no or crep.post) then
5077                 changed = changed + 1
5078                 d = node.new(7, 0) %% (disc, discretionary)
5079                 d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5080                 d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5081                 d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5082                 d.attr = char_base.attr
5083                 if crep.pre == nil then %% TeXbook p96
5084                     d.penalty = crep.penalty or tex.hyphenpenalty
5085                 else
5086                     d.penalty = crep.penalty or tex.exhyphenpenalty
5087                 end
5088                 head, new = node.insert_before(head, char_node, d)
5089                 node.remove(head, char_node)
5090                 if q == 1 then
5091                     word_head = new
5092                 end
5093             elseif crep and crep.string then
5094                 changed = changed + 1
5095                 local str = crep.string(matches)
5096                 if str == '' then
5097                     if q == 1 then
5098                         word_head = char_node.next
5099                     end
5100                     head, new = node.remove(head, char_node)
5101                 elseif char_node.id == 29 and u.len(str) == 1 then
5102                     char_node.char = string.utfvalue(str)
5103                 else
5104                     local n
5105                     for s in string.utfvalues(str) do
5106                         if char_node.id == 7 then
5107                             log('Automatic hyphens cannot be replaced, just removed.')
5108                         else
5109                             n = node.copy(char_base)

```



```

5110         end
5111         n.char = s
5112         if q == 1 then
5113             head, new = node.insert_before(head, char_node, n)
5114             word_head = new
5115         else
5116             node.insert_before(head, char_node, n)
5117         end
5118     end
5119
5120     node.remove(head, char_node)
5121     end %% string length
5122     end %% if char and char.string
5123     end %% for char in match
5124     if changed > 20 then
5125         texio.write('Too many changes. Ignoring the rest.')
5126     elseif changed > 0 then
5127         w, wn, nw = Babel.fetch_word(word_head)
5128     end
5129
5130     end %% for match
5131     end %% for patterns
5132     word_head = nw
5133     end %% for words
5134     return head
5135 end
5136
5137 %%%
5138 %% Preliminary code for \babelprehyphenation
5139 %% TODO. Copypaste pattern. Merge with fetch_word
5140 function Babel.fetch_subtext(head, funct)
5141     local word_string = ''
5142     local word_nodes = {}
5143     local lang
5144     local item = head
5145     local inmath = false
5146
5147     while item do
5148
5149         if item.id == 29 then
5150             local locale = node.get_attribute(item, Babel.attr_locale)
5151
5152             if not(item.char == 124) %% ie, not | = space
5153                 and not inmath
5154                 and (locale == lang or lang == nil) then
5155                 lang = lang or locale
5156                 word_string = word_string .. unicode.utf8.char(item.char)
5157                 word_nodes[#word_nodes+1] = item
5158             end
5159
5160             if item == node.tail(head) then
5161                 item = nil
5162                 return word_string, word_nodes, item, lang
5163             end
5164
5165             elseif item.id == 12 and item.subtype == 13 and not inmath then
5166                 word_string = word_string .. '|'
5167                 word_nodes[#word_nodes+1] = item
5168

```

```

5169         if item == node.tail(head) then
5170             item = nil
5171             return word_string, word_nodes, item, lang
5172         end
5173
5174         elseif item.id == node.id'math' then
5175             inmath = (item.subtype == 0)
5176
5177         elseif word_string == '' then
5178             %% pass
5179
5180         else
5181             return word_string, word_nodes, item, lang
5182         end
5183
5184         item = item.next
5185     end
5186 end
5187
5188 &% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5189 function Babel.pre_hyphenate_replace(head)
5190     local u = unicode.utf8
5191     local lbkr = Babel.linebreaking.pre_replacements
5192     local word_head = head
5193
5194     while true do
5195         local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5196         if not lang then return head end
5197
5198         if not lbkr[lang] then
5199             break
5200         end
5201
5202         for k=1, #lbkr[lang] do
5203             local p = lbkr[lang][k].pattern
5204             local r = lbkr[lang][k].replace
5205
5206             while true do
5207                 local matches = { u.match(w, p) }
5208                 if #matches < 2 then break end
5209
5210                 local first = table.remove(matches, 1)
5211                 local last = table.remove(matches, #matches)
5212
5213                 &% Fix offsets, from bytes to unicode.
5214                 first = u.len(w:sub(1, first-1)) + 1
5215                 last = u.len(w:sub(1, last-1))
5216
5217                 local new &% used when inserting and removing nodes
5218                 local changed = 0
5219
5220                 &% This loop traverses the replace list and takes the
5221                 &% corresponding actions
5222                 for q = first, last do
5223                     local crep = r[q-first+1]
5224                     local char_node = wn[q]
5225                     local char_base = char_node
5226
5227                     if crep and crep.data then

```

```

5228         char_base = wn[crep.data+first-1]
5229     end
5230
5231     if crep == {} then
5232         break
5233     elseif crep == nil then
5234         changed = changed + 1
5235         node.remove(head, char_node)
5236     elseif crep and crep.string then
5237         changed = changed + 1
5238         local str = crep.string(matches)
5239         if str == '' then
5240             if q == 1 then
5241                 word_head = char_node.next
5242             end
5243             head, new = node.remove(head, char_node)
5244         elseif char_node.id == 29 and u.len(str) == 1 then
5245             char_node.char = string.utfvalue(str)
5246         else
5247             local n
5248             for s in string.utfvalues(str) do
5249                 if char_node.id == 7 then
5250                     log('Automatic hyphens cannot be replaced, just removed.')
5251                 else
5252                     n = node.copy(char_base)
5253                 end
5254                 n.char = s
5255                 if q == 1 then
5256                     head, new = node.insert_before(head, char_node, n)
5257                     word_head = new
5258                 else
5259                     node.insert_before(head, char_node, n)
5260                 end
5261             end
5262
5263             node.remove(head, char_node)
5264         end &% string length
5265     end &% if char and char.string
5266 end &% for char in match
5267 if changed > 20 then
5268     texio.write('Too many changes. Ignoring the rest.')
5269 elseif changed > 0 then
5270     &% For one-to-one can we modify directly the
5271     &% values without re-fetching? Very likely.
5272     w, wn, nw = Babel.fetch_subtext(word_head)
5273 end
5274
5275     end &% for match
5276 end &% for patterns
5277 word_head = nw
5278 end &% for words
5279 return head
5280 end
5281 &%% end of preliminary code for \babelprehyphenation
5282
5283 &% The following functions belong to the next macro
5284
5285 &% This table stores capture maps, numbered consecutively
5286 Babel.capture_maps = {}

```

```

5287
5288 function Babel.capture_func(key, cap)
5289   local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
5290   ret = ret:gsub('{{([0-9])|([^\]|+)|(-)}}', Babel.capture_func_map)
5291   ret = ret:gsub("%[%[%]%%.%.%", '')
5292   ret = ret:gsub("%.%.%.%.%.%", '')
5293   return key .. "[=function(m) return ]] .. ret .. [[ end]]
5294 end
5295
5296 function Babel.capt_map(from, mapno)
5297   return Babel.capture_maps[mapno][from] or from
5298 end
5299
5300 &% Handle the {n|abc|ABC} syntax in captures
5301 function Babel.capture_func_map(capno, from, to)
5302   local froms = {}
5303   for s in string.utfcharacters(from) do
5304     table.insert(froms, s)
5305   end
5306   local cnt = 1
5307   table.insert(Babel.capture_maps, {})
5308   local mlen = table.getn(Babel.capture_maps)
5309   for s in string.utfcharacters(to) do
5310     Babel.capture_maps[mlen][froms[cnt]] = s
5311     cnt = cnt + 1
5312   end
5313   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5314     (mlen) .. ").." .. "["
5315 end
5316 }

```

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, `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).`

```

5317 \catcode`\#=6
5318 \gdef\babelposthyphenation#1#2#3{&%
5319   \bbl@activateposthyphen
5320   \begingroup
5321     \def\babeltempa{\bbl@add@list\babeltempb}&%
5322     \let\babeltempb\@empty
5323     \bbl@foreach{#3}{&%
5324       \bbl@ifsamestring{##1}{remove}&%
5325       {\bbl@add@list\babeltempb{nil}}&%
5326       {\directlua{
5327         local rep = [[##1]]
5328         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5329         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5330         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5331         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5332         tex.print([[ \string\babeltempa{[]} .. rep .. []]])
5333       }}}&%

```

```

5334 \directlua{
5335     local lbkr = Babel.linebreaking.post_replacements
5336     local u = unicode.utf8
5337     %% Convert pattern:
5338     local patt = string.gsub(#[#2]=, '%s', '')
5339     if not u.find(patt, '()', nil, true) then
5340         patt = '()' .. patt .. '()'
5341     end
5342     patt = u.gsub(patt, '{(.)}',
5343         function (n)
5344             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5345         end)
5346     lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5347     table.insert(lbkr[\the\csname l@#1\endcsname],
5348         { pattern = patt, replace = { \babeltempb } })
5349 }%%
5350 \endgroup}
5351 % TODO. Working !!! Copypaste pattern.
5352 \gdef\babelprehyphenation#1#2#3{%%
5353     \bbl@activateprehyphen
5354     \beginingroup
5355         \def\babeltempa{\bbl@add@list\babeltempb}%%
5356         \let\babeltempb\@empty
5357         \bbl@foreach{#3}{%%
5358             \bbl@ifsamestring{##1}{remove}%%
5359             {\bbl@add@list\babeltempb{nil}}%%
5360             {\directlua{
5361                 local rep = [#{##1}]
5362                 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5363                 tex.print([[\string\babeltempa{[]} .. rep .. [{}]])
5364             }}%%
5365         \directlua{
5366             local lbkr = Babel.linebreaking.pre_replacements
5367             local u = unicode.utf8
5368             %% Convert pattern:
5369             local patt = string.gsub(#[#2]=, '%s', '')
5370             if not u.find(patt, '()', nil, true) then
5371                 patt = '()' .. patt .. '()'
5372             end
5373             patt = u.gsub(patt, '{(.)}',
5374                 function (n)
5375                     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5376                 end)
5377             lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5378             table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5379                 { pattern = patt, replace = { \babeltempb } })
5380         }%%
5381     \endgroup}
5382 \endgroup}
5383 \def\bbl@activateposthyphen{%
5384     \let\bbl@activateposthyphen\relax
5385     \directlua{
5386         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5387     }}
5388 % TODO. Working !!!
5389 \def\bbl@activateprehyphen{%
5390     \let\bbl@activateprehyphen\relax
5391     \directlua{
5392         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)

```

5393 }}

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`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```
5394 \bbl@trace{Redefinitions for bidi layout}
5395 \ifx\@eqnnum\@undefined\else
5396   \ifx\bbl@attr@dir\@undefined\else
5397     \edef\@eqnnum{%
5398       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5399       \unexpanded\expandafter{\@eqnnum}}%
5400   \fi
5401 \fi
5402 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5403 \ifnum\bbl@bidimode>\z@
5404   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5405     \bbl@exp{%
5406       \mathdir\the\bodydir
5407       #1%           Once entered in math, set boxes to restore values
5408       \<ifmmode>%
5409       \everyvbox{%
5410         \the\everyvbox
5411         \bodydir\the\bodydir
5412         \mathdir\the\mathdir
5413         \everyhbox{\the\everyhbox}%
5414         \everyvbox{\the\everyvbox}}%
5415       \everyhbox{%
5416         \the\everyhbox
5417         \bodydir\the\bodydir
5418         \mathdir\the\mathdir
5419         \everyhbox{\the\everyhbox}%
5420         \everyvbox{\the\everyvbox}}%
5421       \<fi>}}%
5422   \def\@hangfrom#1{%
5423     \setbox\@tempboxa\hbox{{#1}}%
5424     \hangindent\wd\@tempboxa
5425     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5426       \shapemode\@ne
5427     \fi
5428     \noindent\box\@tempboxa}
5429 \fi
5430 \IfBabelLayout{tabular}
5431   {\let\bbl@OL@tabular\@tabular
5432     \bbl@replace\@tabular{$}\bbl@nextfake$}%
5433   \let\bbl@NL@tabular\@tabular
5434   \AtBeginDocument{%
```

```

5435 \ifx\bb1@NL@@tabular\@tabular\else
5436 \bb1@replace\@tabular{$}\bb1@nextfake$}%
5437 \let\bb1@NL@@tabular\@tabular
5438 \fi}}
5439 {}
5440 \IfBabelLayout{lists}
5441 {\let\bb1@OL@list\list
5442 \bb1@sreplace\list{\parshape}\bb1@listparshape}%
5443 \let\bb1@NL@list\list
5444 \def\bb1@listparshape#1#2#3{%
5445 \parshape #1 #2 #3 %
5446 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5447 \shapemode\tw@
5448 \fi}}
5449 {}
5450 \IfBabelLayout{graphics}
5451 {\let\bb1@pictresetdir\relax
5452 \def\bb1@pictsetdir{%
5453 \ifcase\bb1@thetextdir
5454 \let\bb1@pictresetdir\relax
5455 \else
5456 \textdir TLT\relax
5457 \def\bb1@pictresetdir{\textdir TRT\relax}%
5458 \fi}%
5459 \let\bb1@OL@@picture\@picture
5460 \let\bb1@OL@put\put
5461 \bb1@sreplace\@picture{\hskip-}\bb1@pictsetdir\hskip-}%
5462 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5463 \@killglue
5464 \raise#2\unitlength
5465 \hb@xt@z{\kern#1\unitlength\bb1@pictresetdir#3}\hss}}%
5466 \AtBeginDocument
5467 {\ifx\tikz@atbegin@node\undefined\else
5468 \let\bb1@OL@pgfpicture\pgfpicture
5469 \bb1@sreplace\pgfpicture{\pgfpicturetrue}\bb1@pictsetdir\pgfpicturetrue}%
5470 \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir}%
5471 \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5472 \fi}}
5473 {}

```

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

```

5474 \IfBabelLayout{counters}%
5475 {\let\bb1@OL@@textsuperscript\@textsuperscript
5476 \bb1@sreplace\@textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5477 \let\bb1@latinarabic=\@arabic
5478 \let\bb1@OL@@arabic\@arabic
5479 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
5480 \@ifpackagewith{babel}{bidi=default}%
5481 {\let\bb1@asciroman=\@roman
5482 \let\bb1@OL@@roman\@roman
5483 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
5484 \let\bb1@asciiRoman=\@Roman
5485 \let\bb1@OL@@roman\@Roman
5486 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}%
5487 \let\bb1@OL@labelenumii\labelenumii
5488 \def\labelenumii{}\theenumii}%
5489 \let\bb1@OL@p@enumiii\p@enumiii

```

```

5490      \def\p@enumiii{\p@enumii}\theenumii{}{}{}{}
5491 <<Footnote changes>>
5492 \IfBabelLayout{footnotes}%
5493   {\let\bbl@OL@footnote\footnote
5494    \BabelFootnote\footnote\languagename{}{}%
5495    \BabelFootnote\localfootnote\languagename{}{}%
5496    \BabelFootnote\mainfootnote{}{}{}}
5497   {}

```

Some \LaTeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5498 \IfBabelLayout{extras}%
5499   {\let\bbl@OL@underline\underline
5500    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5501    \let\bbl@OL@LaTeX2e\LaTeX2e
5502    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5503     \if b\expandafter\@car\@series\@nil\boldmath\fi
5504     \babelsublr}%
5505     \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
5506   {}
5507 </luatex>

```

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.

```

5508 (*basic-r)
5509 Babel = Babel or {}
5510
5511 Babel.bidi_enabled = true
5512
5513 require('babel-data-bidi.lua')
5514
5515 local characters = Babel.characters
5516 local ranges = Babel.ranges
5517
5518 local DIR = node.id("dir")
5519
5520 local function dir_mark(head, from, to, outer)
5521   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5522   local d = node.new(DIR)
5523   d.dir = '+' .. dir
5524   node.insert_before(head, from, d)
5525   d = node.new(DIR)
5526   d.dir = '-' .. dir
5527   node.insert_after(head, to, d)
5528 end
5529
5530 function Babel.bidi(head, ispar)
5531   local first_n, last_n      -- first and last char with nums
5532   local last_es              -- an auxiliary 'last' used with nums
5533   local first_d, last_d      -- first and last char in L/R block
5534   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):

```

5535   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5536   local strong_lr = (strong == 'l') and 'l' or 'r'
5537   local outer = strong
5538
5539   local new_dir = false
5540   local first_dir = false
5541   local inmath = false
5542
5543   local last_lr
5544
5545   local type_n = ''
5546
5547   for item in node.traverse(head) do
5548
5549     -- three cases: glyph, dir, otherwise
5550     if item.id == node.id'glyph'
5551       or (item.id == 7 and item.subtype == 2) then
5552
5553       local itemchar

```

```

5554     if item.id == 7 and item.subtype == 2 then
5555         itemchar = item.replace.char
5556     else
5557         itemchar = item.char
5558     end
5559     local chardata = characters[itemchar]
5560     dir = chardata and chardata.d or nil
5561     if not dir then
5562         for nn, et in ipairs(ranges) do
5563             if itemchar < et[1] then
5564                 break
5565             elseif itemchar <= et[2] then
5566                 dir = et[3]
5567                 break
5568             end
5569         end
5570     end
5571     dir = dir or 'l'
5572     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).

```

5573     if new_dir then
5574         attr_dir = 0
5575         for at in node.traverse(item.attr) do
5576             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5577                 attr_dir = at.value % 3
5578             end
5579         end
5580         if attr_dir == 1 then
5581             strong = 'r'
5582         elseif attr_dir == 2 then
5583             strong = 'al'
5584         else
5585             strong = 'l'
5586         end
5587         strong_lr = (strong == 'l') and 'l' or 'r'
5588         outer = strong_lr
5589         new_dir = false
5590     end
5591
5592     if dir == 'nsm' then dir = strong end -- W1

```

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

```

5593     dir_real = dir -- We need dir_real to set strong below
5594     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:

```

5595     if strong == 'al' then
5596         if dir == 'en' then dir = 'an' end -- W2
5597         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5598         strong_lr = 'r' -- W3
5599     end

```

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

```

5600     elseif item.id == node.id'dir' and not inmath then
5601         new_dir = true
5602         dir = nil
5603     elseif item.id == node.id'math' then
5604         inmath = (item.subtype == 0)
5605     else
5606         dir = nil          -- Not a char
5607     end

```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```

5608     if dir == 'en' or dir == 'an' or dir == 'et' then
5609         if dir ~= 'et' then
5610             type_n = dir
5611         end
5612         first_n = first_n or item
5613         last_n = last_es or item
5614         last_es = nil
5615     elseif dir == 'es' and last_n then -- W3+W6
5616         last_es = item
5617     elseif dir == 'cs' then          -- it's right - do nothing
5618     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5619         if strong_lr == 'r' and type_n ~= '' then
5620             dir_mark(head, first_n, last_n, 'r')
5621         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5622             dir_mark(head, first_n, last_n, 'r')
5623             dir_mark(head, first_d, last_d, outer)
5624             first_d, last_d = nil, nil
5625         elseif strong_lr == 'l' and type_n ~= '' then
5626             last_d = last_n
5627         end
5628         type_n = ''
5629         first_n, last_n = nil, nil
5630     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:

```

5631     if dir == 'l' or dir == 'r' then
5632         if dir ~= outer then
5633             first_d = first_d or item
5634             last_d = item
5635         elseif first_d and dir ~= strong_lr then
5636             dir_mark(head, first_d, last_d, outer)
5637             first_d, last_d = nil, nil
5638         end
5639     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.

```

5640   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5641       item.char = characters[item.char] and
5642           characters[item.char].m or item.char
5643   elseif (dir or new_dir) and last_lr ~= item then
5644       local mir = outer .. strong_lr .. (dir or outer)
5645       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5646           for ch in node.traverse(node.next(last_lr)) do
5647               if ch == item then break end
5648               if ch.id == node.id'glyph' and characters[ch.char] then
5649                   ch.char = characters[ch.char].m or ch.char
5650               end
5651           end
5652       end
5653   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).

```

5654   if dir == 'l' or dir == 'r' then
5655       last_lr = item
5656       strong = dir_real          -- Don't search back - best save now
5657       strong_lr = (strong == 'l') and 'l' or 'r'
5658   elseif new_dir then
5659       last_lr = nil
5660   end
5661 end

```

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

```

5662   if last_lr and outer == 'r' then
5663       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5664           if characters[ch.char] then
5665               ch.char = characters[ch.char].m or ch.char
5666           end
5667       end
5668   end
5669   if first_n then
5670       dir_mark(head, first_n, last_n, outer)
5671   end
5672   if first_d then
5673       dir_mark(head, first_d, last_d, outer)
5674   end

```

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

```

5675   return node.prev(head) or head
5676 end
5677 </basic-r>

```

And here the Lua code for bidi=basic:

```

5678 <(*basic)
5679 Babel = Babel or {}
5680
5681 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5682
5683 Babel.fontmap = Babel.fontmap or {}
5684 Babel.fontmap[0] = {}          -- 1

```

```

5685 Babel.fontmap[1] = {}      -- r
5686 Babel.fontmap[2] = {}      -- al/an
5687
5688 Babel.bidi_enabled = true
5689 Babel.mirroring_enabled = true
5690
5691 require('babel-data-bidi.lua')
5692
5693 local characters = Babel.characters
5694 local ranges = Babel.ranges
5695
5696 local DIR = node.id('dir')
5697 local GLYPH = node.id('glyph')
5698
5699 local function insert_implicit(head, state, outer)
5700     local new_state = state
5701     if state.sim and state.eim and state.sim ~= state.eim then
5702         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5703         local d = node.new(DIR)
5704         d.dir = '+' .. dir
5705         node.insert_before(head, state.sim, d)
5706         local d = node.new(DIR)
5707         d.dir = '-' .. dir
5708         node.insert_after(head, state.eim, d)
5709     end
5710     new_state.sim, new_state.eim = nil, nil
5711     return head, new_state
5712 end
5713
5714 local function insert_numeric(head, state)
5715     local new
5716     local new_state = state
5717     if state.san and state.ean and state.san ~= state.ean then
5718         local d = node.new(DIR)
5719         d.dir = '+TLT'
5720         _, new = node.insert_before(head, state.san, d)
5721         if state.san == state.sim then state.sim = new end
5722         local d = node.new(DIR)
5723         d.dir = '-TLT'
5724         _, new = node.insert_after(head, state.ean, d)
5725         if state.ean == state.eim then state.eim = new end
5726     end
5727     new_state.san, new_state.ean = nil, nil
5728     return head, new_state
5729 end
5730
5731 -- TODO - \hbox with an explicit dir can lead to wrong results
5732 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5733 -- was s made to improve the situation, but the problem is the 3-dir
5734 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5735 -- well.
5736
5737 function Babel.bidi(head, ispar, hdir)
5738     local d -- d is used mainly for computations in a loop
5739     local prev_d = ''
5740     local new_d = false
5741
5742     local nodes = {}
5743     local outer_first = nil

```

```

5744 local inmath = false
5745
5746 local glue_d = nil
5747 local glue_i = nil
5748
5749 local has_en = false
5750 local first_et = nil
5751
5752 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5753
5754 local save_outer
5755 local temp = node.get_attribute(head, ATDIR)
5756 if temp then
5757     temp = temp % 3
5758     save_outer = (temp == 0 and 'l') or
5759                  (temp == 1 and 'r') or
5760                  (temp == 2 and 'al')
5761 elseif ispar then -- Or error? Shouldn't happen
5762     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5763 else -- Or error? Shouldn't happen
5764     save_outer = ('TRT' == hdir) and 'r' or 'l'
5765 end
5766 -- when the callback is called, we are just _after_ the box,
5767 -- and the textdir is that of the surrounding text
5768 -- if not ispar and hdir ~= tex.textdir then
5769 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5770 -- end
5771 local outer = save_outer
5772 local last = outer
5773 -- 'al' is only taken into account in the first, current loop
5774 if save_outer == 'al' then save_outer = 'r' end
5775
5776 local fontmap = Babel.fontmap
5777
5778 for item in node.traverse(head) do
5779
5780     -- In what follows, #node is the last (previous) node, because the
5781     -- current one is not added until we start processing the neutrals.
5782
5783     -- three cases: glyph, dir, otherwise
5784     if item.id == GLYPH
5785         or (item.id == 7 and item.subtype == 2) then
5786
5787         local d_font = nil
5788         local item_r
5789         if item.id == 7 and item.subtype == 2 then
5790             item_r = item.replace -- automatic discs have just 1 glyph
5791         else
5792             item_r = item
5793         end
5794         local chardata = characters[item_r.char]
5795         d = chardata and chardata.d or nil
5796         if not d or d == 'nsm' then
5797             for nn, et in ipairs(ranges) do
5798                 if item_r.char < et[1] then
5799                     break
5800                 elseif item_r.char <= et[2] then
5801                     if not d then d = et[3]
5802                     elseif d == 'nsm' then d_font = et[3]

```

```

5803         end
5804         break
5805     end
5806 end
5807 end
5808 d = d or 'l'
5809
5810 -- A short 'pause' in bidi for mapfont
5811 d_font = d_font or d
5812 d_font = (d_font == 'l' and 0) or
5813           (d_font == 'nsm' and 0) or
5814           (d_font == 'r' and 1) or
5815           (d_font == 'al' and 2) or
5816           (d_font == 'an' and 2) or nil
5817 if d_font and fontmap and fontmap[d_font][item_r.font] then
5818     item_r.font = fontmap[d_font][item_r.font]
5819 end
5820
5821 if new_d then
5822     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5823     if inmath then
5824         attr_d = 0
5825     else
5826         attr_d = node.get_attribute(item, ATDIR)
5827         attr_d = attr_d % 3
5828     end
5829     if attr_d == 1 then
5830         outer_first = 'r'
5831         last = 'r'
5832     elseif attr_d == 2 then
5833         outer_first = 'r'
5834         last = 'al'
5835     else
5836         outer_first = 'l'
5837         last = 'l'
5838     end
5839     outer = last
5840     has_en = false
5841     first_et = nil
5842     new_d = false
5843 end
5844
5845 if glue_d then
5846     if (d == 'l' and 'l' or 'r') ~= glue_d then
5847         table.insert(nodes, {glue_i, 'on', nil})
5848     end
5849     glue_d = nil
5850     glue_i = nil
5851 end
5852
5853 elseif item.id == DIR then
5854     d = nil
5855     new_d = true
5856
5857 elseif item.id == node.id'glue' and item.subtype == 13 then
5858     glue_d = d
5859     glue_i = item
5860     d = nil
5861

```

```

5862 elseif item.id == node.id'math' then
5863     inmath = (item.subtype == 0)
5864
5865 else
5866     d = nil
5867 end
5868
5869 -- AL <= EN/ET/ES      -- W2 + W3 + W6
5870 if last == 'al' and d == 'en' then
5871     d = 'an'           -- W3
5872 elseif last == 'al' and (d == 'et' or d == 'es') then
5873     d = 'on'           -- W6
5874 end
5875
5876 -- EN + CS/ES + EN      -- W4
5877 if d == 'en' and #nodes >= 2 then
5878     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5879         and nodes[#nodes-1][2] == 'en' then
5880         nodes[#nodes][2] = 'en'
5881     end
5882 end
5883
5884 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
5885 if d == 'an' and #nodes >= 2 then
5886     if (nodes[#nodes][2] == 'cs')
5887         and nodes[#nodes-1][2] == 'an' then
5888         nodes[#nodes][2] = 'an'
5889     end
5890 end
5891
5892 -- ET/EN                -- W5 + W7->l / W6->on
5893 if d == 'et' then
5894     first_et = first_et or (#nodes + 1)
5895 elseif d == 'en' then
5896     has_en = true
5897     first_et = first_et or (#nodes + 1)
5898 elseif first_et then    -- d may be nil here !
5899     if has_en then
5900         if last == 'l' then
5901             temp = 'l'    -- W7
5902         else
5903             temp = 'en'   -- W5
5904         end
5905     else
5906         temp = 'on'       -- W6
5907     end
5908     for e = first_et, #nodes do
5909         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5910     end
5911     first_et = nil
5912     has_en = false
5913 end
5914
5915 if d then
5916     if d == 'al' then
5917         d = 'r'
5918         last = 'al'
5919     elseif d == 'l' or d == 'r' then
5920         last = d

```



```

5921     end
5922     prev_d = d
5923     table.insert(nodes, {item, d, outer_first})
5924 end
5925
5926     outer_first = nil
5927
5928 end
5929
5930 -- TODO -- repeated here in case EN/ET is the last node. Find a
5931 -- better way of doing things:
5932 if first_et then      -- dir may be nil here !
5933     if has_en then
5934         if last == 'l' then
5935             temp = 'l'      -- W7
5936         else
5937             temp = 'en'     -- W5
5938         end
5939     else
5940         temp = 'on'        -- W6
5941     end
5942     for e = first_et, #nodes do
5943         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5944     end
5945 end
5946
5947 -- dummy node, to close things
5948 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5949
5950 ----- NEUTRAL -----
5951
5952 outer = save_outer
5953 last = outer
5954
5955 local first_on = nil
5956
5957 for q = 1, #nodes do
5958     local item
5959
5960     local outer_first = nodes[q][3]
5961     outer = outer_first or outer
5962     last = outer_first or last
5963
5964     local d = nodes[q][2]
5965     if d == 'an' or d == 'en' then d = 'r' end
5966     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5967
5968     if d == 'on' then
5969         first_on = first_on or q
5970     elseif first_on then
5971         if last == d then
5972             temp = d
5973         else
5974             temp = outer
5975         end
5976         for r = first_on, q - 1 do
5977             nodes[r][2] = temp
5978             item = nodes[r][1]      -- MIRRORING
5979             if Babel.mirroring_enabled and item.id == GLYPH

```

```

5980         and temp == 'r' and characters[item.char] then
5981             local font_mode = font.fonts[item.font].properties.mode
5982             if font_mode ~= 'harf' and font_mode ~= 'plug' then
5983                 item.char = characters[item.char].m or item.char
5984             end
5985         end
5986     end
5987     first_on = nil
5988 end
5989
5990 if d == 'r' or d == 'l' then last = d end
5991 end
5992
5993 ----- IMPLICIT, REORDER -----
5994
5995 outer = save_outer
5996 last = outer
5997
5998 local state = {}
5999 state.has_r = false
6000
6001 for q = 1, #nodes do
6002
6003     local item = nodes[q][1]
6004
6005     outer = nodes[q][3] or outer
6006
6007     local d = nodes[q][2]
6008
6009     if d == 'nsm' then d = last end           -- W1
6010     if d == 'en' then d = 'an' end
6011     local isdir = (d == 'r' or d == 'l')
6012
6013     if outer == 'l' and d == 'an' then
6014         state.san = state.san or item
6015         state.ean = item
6016     elseif state.san then
6017         head, state = insert_numeric(head, state)
6018     end
6019
6020     if outer == 'l' then
6021         if d == 'an' or d == 'r' then        -- im -> implicit
6022             if d == 'r' then state.has_r = true end
6023             state.sim = state.sim or item
6024             state.eim = item
6025         elseif d == 'l' and state.sim and state.has_r then
6026             head, state = insert_implicit(head, state, outer)
6027         elseif d == 'l' then
6028             state.sim, state.eim, state.has_r = nil, nil, false
6029         end
6030     else
6031         if d == 'an' or d == 'l' then
6032             if nodes[q][3] then -- nil except after an explicit dir
6033                 state.sim = item -- so we move sim 'inside' the group
6034             else
6035                 state.sim = state.sim or item
6036             end
6037             state.eim = item
6038         elseif d == 'r' and state.sim then

```

```

6039         head, state = insert_implicit(head, state, outer)
6040     elseif d == 'r' then
6041         state.sim, state.eim = nil, nil
6042     end
6043 end
6044
6045 if isdir then
6046     last = d          -- Don't search back - best save now
6047 elseif d == 'on' and state.san then
6048     state.san = state.san or item
6049     state.ean = item
6050 end
6051
6052 end
6053
6054 return node.prev(head) or head
6055 end
6056 </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.

```

6057 <nil>
6058 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6059 \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.

```

6060 \ifx\l@nil\@undefined
6061   \newlanguage\l@nil
6062   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
6063   \let\bbl@elt\relax
6064   \edef\bbl@languages{% Add it to the list of languages
6065     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
6066 \fi

```

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

```

6067 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 6068 \let\captionnil\@empty
6069 \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.

```
6070 \ldf@finish{nil}
6071 \</nil>
```

16 Support for Plain \TeX (plain.def)

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based \TeX -format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `locallyhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with $\text{\textit{iniTeX}}$, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing $\text{\textit{iniTeX}}$ sees, we need to set some category codes just to be able to change the definition of `\input`.

```
6072 (*bplain | blplain)
6073 \catcode`\{=1 % left brace is begin-group character
6074 \catcode`\}=2 % right brace is end-group character
6075 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
6076 \openin 0 hyphen.cfg
6077 \ifeof0
6078 \else
6079 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that’s done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
6080 \def\input #1 {%
6081 \let\input\input
6082 \a hyphen.cfg
6083 \let\input\undefined
6084 }
6085 \fi
6086 \</bplain | blplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
6087 \bplain\la plain.tex
6088 \bplain\la lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
6089 \bplain\def\fmtname{babel-plain}
6090 \bplain\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of $\text{\LaTeX} 2_{\epsilon}$ that are needed for `babel`.

```
6091 \langle\langle *Emulate LaTeX \rangle\rangle \equiv
6092 % == Code for plain ==
6093 \def\@empty{}
6094 \def\loadlocalcfg#1{%
6095   \openin0#1.cfg
6096   \ifeof0
6097     \closein0
6098   \else
6099     \closein0
6100     {\immediate\write16{*****}%
6101      \immediate\write16{* Local config file #1.cfg used}%
6102      \immediate\write16{*}%
6103     }
6104     \input #1.cfg\relax
6105   \fi
6106   \@endoflfd}
```

16.3 General tools

A number of \LaTeX macro's that are needed later on.

```
6107 \long\def\@firstofone#1{#1}
6108 \long\def\@firstoftwo#1#2{#1}
6109 \long\def\@secondoftwo#1#2{#2}
6110 \def\@nnil{\@nil}
6111 \def\@gobbletwo#1#2{}
6112 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6113 \def\@star@or@long#1{%
6114   \@ifstar
6115   {\let\l@ngrel@x\relax#1}%
6116   {\let\l@ngrel@x\long#1}}
6117 \let\l@ngrel@x\relax
6118 \def\@car#1#2\@nil{#1}
6119 \def\@cdr#1#2\@nil{#2}
6120 \let\@typeset@protect\relax
6121 \let\protected@edef\edef
6122 \long\def\@gobble#1{}
6123 \edef\@backslashchar{\expandafter\@gobble\string\}
6124 \def\strip@prefix#1>{}
6125 \def\g@addto@macro#1#2{{%
6126   \toks@\expandafter{#1#2}%
6127   \xdef#1{\the\toks@}}}
```

```

6128 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6129 \def\@nameuse#1{\csname #1\endcsname}
6130 \def\@ifundefined#1{%
6131   \expandafter\ifx\csname#1\endcsname\relax
6132     \expandafter\@firstoftwo
6133   \else
6134     \expandafter\@secondoftwo
6135   \fi}
6136 \def\@expandtwoargs#1#2#3{%
6137   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6138 \def\zap@space#1 #2{%
6139   #1%
6140   \ifx#2\@empty\else\expandafter\zap@space\fi
6141   #2}
6142 \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}`.

```

6143 \ifx\@preamblecmds\@undefined
6144   \def\@preamblecmds{}
6145 \fi
6146 \def\@onlypreamble#1{%
6147   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6148     \@preamblecmds\do#1}}
6149 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

6150 \def\begin{document}{%
6151   \@begin{document}hook
6152   \global\let\@begin{document}hook\@undefined
6153   \def\do##1{\global\let##1\@undefined}%
6154   \@preamblecmds
6155   \global\let\do\noexpand}
6156 \ifx\@begin{document}hook\@undefined
6157   \def\@begin{document}hook{}
6158 \fi
6159 \@onlypreamble\@begin{document}hook
6160 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

6161 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6162 \@onlypreamble\AtEndOfPackage
6163 \def\@endofldf{}
6164 \@onlypreamble\@endofldf
6165 \let\bbl@afterlang\@empty
6166 \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.

```

6167 \catcode`\&=\z@
6168 \ifx&\if@files\@undefined
6169   \expandafter\let\csname if@files\endcsname
6170   \csname iffalse\endcsname
6171 \fi
6172 \catcode`\&=4

```

Mimick L^AT_EX's commands to define control sequences.

```

6173 \def\newcommand{\@star@or@long\new@command}
6174 \def\new@command#1{%
6175   \@testopt{\@newcommand#1}0}
6176 \def\@newcommand#1[#2]{%
6177   \@ifnextchar [{\@xargdef#1[#2]]%
6178               {\@argdef#1[#2]}}
6179 \long\def\@argdef#1[#2]#3{%
6180   \@yargdef#1\@ne{#2}{#3}}
6181 \long\def\@xargdef#1[#2][#3]#4{%
6182   \expandafter\def\expandafter#1\expandafter{%
6183     \expandafter\@protected@testopt\expandafter #1%
6184     \csname\string#1\expandafter\endcsname{#3}}%
6185   \expandafter\@yargdef \csname\string#1\endcsname
6186   \tw@{#2}{#4}}
6187 \long\def\@yargdef#1#2#3{%
6188   \@tempcnta#3\relax
6189   \advance \@tempcnta \@ne
6190   \let\@hash@\relax
6191   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6192   \@tempcntb #2%
6193   \@whilenum\@tempcntb <\@tempcnta
6194   \do{%
6195     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6196     \advance\@tempcntb \@ne}%
6197   \let\@hash@###
6198   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6199 \def\providecommand{\@star@or@long\provide@command}
6200 \def\provide@command#1{%
6201   \begingroup
6202     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
6203   \endgroup
6204   \expandafter\@ifundefined\@gtempa
6205     {\def\reserved@a{\new@command#1}}%
6206     {\let\reserved@a\relax
6207      \def\reserved@a{\new@command\reserved@a}}%
6208   \reserved@a}%
6209 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6210 \def\declare@robustcommand#1{%
6211   \edef\reserved@a{\string#1}%
6212   \def\reserved@b{#1}%
6213   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6214   \edef#1{%
6215     \ifx\reserved@a\reserved@b
6216       \noexpand\x@protect
6217       \noexpand#1%
6218     \fi
6219     \noexpand\protect
6220     \expandafter\noexpand\csname
6221       \expandafter\@gobble\string#1 \endcsname
6222   }%
6223   \expandafter\new@command\csname
6224     \expandafter\@gobble\string#1 \endcsname
6225 }
6226 \def\x@protect#1{%
6227   \ifx\protect\@typeset@protect\else
6228     \@x@protect#1%
6229   \fi

```

```

6230 }
6231 \catcode`\&=\z@ % Trick to hide conditionals
6232 \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`.

```

6233 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6234 \catcode`\&=4
6235 \ifx\in@\@undefined
6236 \def\in@#1#2{%
6237   \def\in@@##1#1##2##3\in@{%
6238     \ifx\in@@##2\in@false\else\in@true\fi}%
6239   \in@@#2#1\in@\in@@}
6240 \else
6241 \let\bbl@tempa\@empty
6242 \fi
6243 \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).

```

6244 \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.

```

6245 \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 $\text{\LaTeX} 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

6246 \ifx\@tempcnta\@undefined
6247 \csname newcount\endcsname\@tempcnta\relax
6248 \fi
6249 \ifx\@tempcntb\@undefined
6250 \csname newcount\endcsname\@tempcntb\relax
6251 \fi

```

To prevent wasting two counters in $\text{\LaTeX} 2.09$ (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

6252 \ifx\bye\@undefined
6253 \advance\count10 by -2\relax
6254 \fi
6255 \ifx\@ifnextchar\@undefined
6256 \def\@ifnextchar#1#2#3{%
6257   \let\reserved@d=#1%
6258   \def\reserved@a{#2}\def\reserved@b{#3}%
6259   \futurelet\@let@token\@ifnch}
6260 \def\@ifnch{%
6261   \ifx\@let@token\@sptoken
6262     \let\reserved@c\@xifnch
6263   \else
6264     \ifx\@let@token\reserved@d
6265       \let\reserved@c\reserved@a

```



```

6266     \else
6267     \let\reserved@c\reserved@b
6268     \fi
6269 \fi
6270 \reserved@c}
6271 \def\:\let\sptoken= } \: % this makes \@sptoken a space token
6272 \def\:\@xifnch} \expandafter\def\:\{\futurelet\@let@token\@ifnch}
6273 \fi
6274 \def\@testopt#1#2{%
6275   \ifnextchar[{\#1}{\#1[#2]}}
6276 \def\@protected@testopt#1{%
6277   \ifx\protect\@typeset@protect
6278     \expandafter\@testopt
6279   \else
6280     \@x@protect#1%
6281   \fi}
6282 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6283   #2\relax}\fi}
6284 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6285   \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

6286 \def\DeclareTextCommand{%
6287   \@dec@text@cmd\providecommand
6288 }
6289 \def\ProvideTextCommand{%
6290   \@dec@text@cmd\providecommand
6291 }
6292 \def\DeclareTextSymbol#1#2#3{%
6293   \@dec@text@cmd\chardef#1{#2}#3\relax
6294 }
6295 \def\@dec@text@cmd#1#2#3{%
6296   \expandafter\def\expandafter#2%
6297     \expandafter{%
6298       \csname#3-cmd\expandafter\endcsname
6299       \expandafter#2%
6300       \csname#3\string#2\endcsname
6301     }%
6302 %   \let\@ifdefinable\rc@ifdefinable
6303   \expandafter#1\csname#3\string#2\endcsname
6304 }
6305 \def\@current@cmd#1{%
6306   \ifx\protect\@typeset@protect\else
6307     \noexpand#1\expandafter\@gobble
6308   \fi
6309 }
6310 \def\@changed@cmd#1#2{%
6311   \ifx\protect\@typeset@protect
6312     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6313       \expandafter\ifx\csname ?\string#1\endcsname\relax
6314         \expandafter\def\csname ?\string#1\endcsname{%
6315           \@changed@x@err{#1}%
6316         }%
6317       \fi
6318     \global\expandafter\let
6319     \csname\cf@encoding\string#1\expandafter\endcsname

```

```

6320         \csname ?\string#1\endcsname
6321     \fi
6322     \csname\cf@encoding\string#1%
6323     \expandafter\endcsname
6324 \else
6325     \noexpand#1%
6326 \fi
6327 }
6328 \def\@changed@x@err#1{%
6329     \errhelp{Your command will be ignored, type <return> to proceed}%
6330     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6331 \def\DeclareTextCommandDefault#1{%
6332     \DeclareTextCommand#1?%
6333 }
6334 \def\ProvideTextCommandDefault#1{%
6335     \ProvideTextCommand#1?%
6336 }
6337 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6338 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6339 \def\DeclareTextAccent#1#2#3{%
6340     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6341 }
6342 \def\DeclareTextCompositeCommand#1#2#3#4{%
6343     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6344     \edef\reserved@b{\string##1}%
6345     \edef\reserved@c{%
6346         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6347     \ifx\reserved@b\reserved@c
6348         \expandafter\expandafter\expandafter\ifx
6349             \expandafter\@car\reserved@a\relax\relax\@nil
6350             \@text@composite
6351     \else
6352         \edef\reserved@b##1{%
6353             \def\expandafter\noexpand
6354                 \csname#2\string#1\endcsname####1{%
6355                 \noexpand\@text@composite
6356                 \expandafter\noexpand\csname#2\string#1\endcsname
6357                 ####1\noexpand\@empty\noexpand\@text@composite
6358                 {##1}%
6359             }%
6360         }%
6361         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6362     \fi
6363     \expandafter\def\csname\expandafter\string\csname
6364         #2\endcsname\string#1-\string#3\endcsname{#4}
6365 \else
6366     \errhelp{Your command will be ignored, type <return> to proceed}%
6367     \errmessage{\string\DeclareTextCompositeCommand\space used on
6368         inappropriate command \protect#1}
6369 \fi
6370 }
6371 \def\@text@composite#1#2#3\@text@composite{%
6372     \expandafter\@text@composite@x
6373     \csname\string#1-\string#2\endcsname
6374 }
6375 \def\@text@composite@x#1#2{%
6376     \ifx#1\relax
6377         #2%
6378     \else

```

```

6379      #1%
6380      \fi
6381 }
6382 %
6383 \def\@strip@args#1:#2-#3\@strip@args{#2}
6384 \def\DeclareTextComposite#1#2#3#4{%
6385     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6386     \bgroup
6387         \lccode` \@=#4%
6388         \lowercase{%
6389     \egroup
6390     \reserved@a @%
6391 }%
6392 }
6393 %
6394 \def\UseTextSymbol#1#2{%
6395     \let\@curr@enc\cf@encoding
6396 %     \@use@text@encoding{#1}%
6397     #2%
6398 %     \@use@text@encoding\@curr@enc
6399 }
6400 \def\UseTextAccent#1#2#3{%
6401 %     \let\@curr@enc\cf@encoding
6402 %     \@use@text@encoding{#1}%
6403 %     #2{\@use@text@encoding\@curr@enc\selectfont#3}%
6404 %     \@use@text@encoding\@curr@enc
6405 }
6406 \def\@use@text@encoding#1{%
6407 %     \edef\font@encoding{#1}%
6408 %     \xdef\font@name{%
6409 %         \csname\curr@fontshape/\font@size\endcsname
6410 %     }%
6411 %     \pickup@font
6412 %     \font@name
6413 %     \@@enc@update
6414 }
6415 \def\DeclareTextSymbolDefault#1#2{%
6416     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6417 }
6418 \def\DeclareTextAccentDefault#1#2{%
6419     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6420 }
6421 \def\cf@encoding{OT1}

```

Currently we only use the $\text{\LaTeX 2}_{\epsilon}$ method for accents for those that are known to be made active in *some* language definition file.

```

6422 \DeclareTextAccent{"}{OT1}{127}
6423 \DeclareTextAccent{'}{OT1}{19}
6424 \DeclareTextAccent{^}{OT1}{94}
6425 \DeclareTextAccent`}{OT1}{18}
6426 \DeclareTextAccent~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for `PLAIN \TeX` .

```

6427 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6428 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
6429 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
6430 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
6431 \DeclareTextSymbol{\i}{OT1}{16}
6432 \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`.

```
6433 \ifx\scriptsize\@undefined
6434   \let\scriptsize\sevenrm
6435 \fi
6436 % End of code for plain
6437 <</Emulate LaTeX>>
```

A proxy file:

```
6438 <plain>
6439 \input babel.def
6440 </plain>
```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] 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 \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).