

Babel

Version 3.63.2449
2021/07/30

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

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	8
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	16
1.12	The base option	18
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	33
1.18	Dates	34
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Transforms	38
1.22	Selection based on BCP 47 tags	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	49
1.31	Current and future work	50
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	51
3	The interface between the core of babel and the language definition files	52
3.1	Guidelines for contributed languages	53
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	57
4	Changes	61
4.1	Changes in babel version 3.9	61

II	Source code	62
5	Identification and loading of required files	62
6	locale directory	62
7	Tools	63
7.1	Multiple languages	67
7.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	68
7.3	base	69
7.4	Conditional loading of shorthands	72
7.5	Cross referencing macros	73
7.6	Marks	76
7.7	Preventing clashes with other packages	77
7.7.1	ifthen	77
7.7.2	varioref	78
7.7.3	hhline	78
7.7.4	hyperref	78
7.7.5	fancyhdr	78
7.8	Encoding and fonts	79
7.9	Basic bidi support	81
7.10	Local Language Configuration	84
7.11	Language options	84
8	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	88
8.1	Tools	88
9	Multiple languages	89
9.1	Selecting the language	91
9.2	Errors	100
9.3	Hooks	103
9.4	Setting up language files	105
9.5	Shorthands	107
9.6	Language attributes	116
9.7	Support for saving macro definitions	118
9.8	Short tags	119
9.9	Hyphens	120
9.10	Multiencoding strings	121
9.11	Macros common to a number of languages	128
9.12	Making glyphs available	128
9.12.1	Quotation marks	128
9.12.2	Letters	130
9.12.3	Shorthands for quotation marks	130
9.12.4	Umlauts and tremas	131
9.13	Layout	133
9.14	Load engine specific macros	133
9.15	Creating and modifying languages	133
10	Adjusting the Babel behavior	155
11	Loading hyphenation patterns	157
12	Font handling with fontspec	161

13	Hooks for XeTeX and LuaTeX	165
13.1	XeTeX	165
13.2	Layout	167
13.3	LuaTeX	169
13.4	Southeast Asian scripts	175
13.5	CJK line breaking	176
13.6	Arabic justification	179
13.7	Common stuff	183
13.8	Automatic fonts and ids switching	183
13.9	Bidi	188
13.10	Layout	190
13.11	Lua: transforms	193
13.12	Lua: Auto bidi with basic and basic-r	201
14	Data for CJK	212
15	The ‘nil’ language	212
16	Support for Plain T_EX (plain.def)	213
16.1	Not renaming hyphen.tex	213
16.2	Emulating some L ^A T _E X features	214
16.3	General tools	214
16.4	Encoding related macros	218
17	Acknowledgements	221

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	6
You are loading directly a language style	8
Unknown language ‘LANG’	9
Argument of \language@active@arg” has an extra }	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	28
Package babel Info: The following fonts are not babel standard families	28

Part I

User guide

What is this document about? This user guide focuses on internationalization and localization with \LaTeX and pdf \TeX , xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-Plain \TeX . Part II describes the code, and usually it can be ignored.

What if I'm interested only in the latest changes? 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 site](#). The most recent features can be still unstable.

Can I help? Sure! If you are interested in the \TeX multilingual support, please join the [kadingira mail list](#). You can follow the development of babel in [GitHub](#) and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

It doesn't work for me! You can ask for help in some forums like tex.stackexchange, but if you have found a bug, I strongly beg you to report it in [GitHub](#), which is much better than just complaining on an e-mail list or a web forum. Remember *warnings are not errors* by themselves, they just warn about possible problems or incompatibilities.

How can I contribute a new language? See section 3.1 for contributing a language.

I only need learn the most basic features. The first subsections (1.1-1.3) describe the traditional way of loading a language (with ldf files), which is usually all you need. The alternative way based on ini files, which complements the previous one (it does *not* replace it, although it is still necessary in some languages), is described below; go to 1.13.

I don't like manuals. I prefer sample files. This manual contains lots of examples and tips, but in GitHub there are many [sample files](#).

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. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in \LaTeX for an option – in this case a language – to be recognized by several packages.

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 lmrroman. 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. It assumes UTF-8, the default encoding:

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

With this setting, the package `varioref` will also see the option `french` and will be able to use it.

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[russian]{article}

\usepackage{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 can 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.

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.

NOTE Although it has been customary to recommend placing `\title`, `\author` and other elements printed by `\maketitle` after `\begin{document}`, mainly because of shorthands, it is advisable to keep them in the preamble. Currently there is no real need to use shorthands in those macros.

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 can 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 with pdfTeX follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDFTEX

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
  
\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}
```

NOTE Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.22 for further details.

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 `\babelfont`, 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 `\babelfont` does *not* load any font until required, so that it can be used just in case.

EXAMPLE A trivial document with the default font in English and Spanish, and FreeSerif in Russian is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

NOTE Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or three-letter word is a valid name for a language (eg, `yi`). See section 1.22 for further details.

1.4 Modifiers

New 3.9c The basic behavior of some languages can be modified when loading babel by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accepts them). An example is (spaces are not significant and they can be added or removed):¹

```
\usepackage[latin.medieval, spanish.notilde.lcroman, danish]{babel}
```

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers. However, modifiers are a more general mechanism.

1.5 Troubleshooting

- Loading directly sty files in L^AT_EX (ie, `\usepackage{⟨language⟩}`) is deprecated and you will get the error:²

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.

```
! 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
```

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 e-Plain and pdf-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 those formats. Please, refer to [Using babel with Plain](#) for further details.

1.7 Basic language selectors

This section describes the commands to be used in the document to switch the language in multilingual documents. In most cases, only the two basic macros `\selectlanguage` and `\foreignlanguage` are necessary. The environments `otherlanguage`, `otherlanguage*` and `hyphenrules` are auxiliary, and described in the next section.

The main language is selected automatically when the document environment begins.

`\selectlanguage` $\{ \langle language \rangle \}$

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

NOTE For “historical reasons”, a macro name is converted to a language name without the leading `\`; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a “real” name is deprecated. **New 3.43** However, if the macro name does not match any language, it will get expanded as expected.

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

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.

WARNING `\selectlanguage` should not be used inside some boxed environments (like floats or minipage) to switch the language if you need the information written to the aux be correctly synchronized. This rarely happens, but if it were the case, you must use `other language` instead.

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

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

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

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

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

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

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

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

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

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

1.9 More on selection

`\babeltags` {*<tag1>* = *<language1>*, *<tag2>* = *<language2>*, ...}

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.

WARNING There is a clear drawback to this feature, namely, the ‘prefix’ `\text...` is heavily overloaded in \TeX and conflicts with existing macros may arise (`\textlatin`, `\textbar`, `\textit`, `\textcolor` and many others). The same applies to environments, because `arabic` conflicts with `\arabic`. Except if there is a reason for this ‘syntactical sugar’, the best option is to stick to the default selectors or to define your own alternatives.

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 the option `fontenc`.⁴ 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 discretionaries and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind 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, 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 deactivated with, eg, `\string`).

TROUBLESHOOTING A typical error when using shorthands is the following:

⁴With it, encoded strings may not work as expected.

! 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` $*\{\langle shorthands-list \rangle\}$

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.

WARNING It is worth emphasizing these macros are meant for temporary changes. Whenever possible and if there are not conflicts with other packages, shorthands must be always enabled (or disabled).

`\usesshorthands` $*\{\langle char \rangle\}$

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

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

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

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

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

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

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

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

```
\usesshorthands*{"}  
\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 can then set:

```
\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 $\{\langle language \rangle\}$

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).⁵ 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 $\{\langle shorthand \rangle\}$

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

⁵Actually, 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.

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

For your records, here is a list of shorthands, but you must double check them, as they may change.⁶

Languages with no shorthands Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh
Languages with only " as defined shorthand character Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

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

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.⁷

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

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

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

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

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

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

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

⁶Thanks to Enrico Gregorio

⁷This declaration serves to nothing, but it is preserved for backward compatibility.

1.11 Package options

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

KeepShorthandsActive Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.

activeacute For some languages babel supports this options to set ' as a shorthand in case it is not done by default.

activegrave Same for `.

shorthands= $\langle char \rangle \langle char \rangle \dots$ | off
The only language shorthands activated are those given, like, eg:

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

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

safe= none | ref | bib

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

math= active | normal

Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.

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

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

- headfoot=** `<language>`
- By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt 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.⁸
- strings=** `generic` | `unicode` | `encoded` | `<label>` | ``
- Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional \TeX , LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal \LaTeX 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.⁹ 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.¹⁰
- 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.¹¹
- 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. 1.24.
- layout=** New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

¹¹Providing `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.

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 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 Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the `...name` strings).

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

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

LUATEX/XETEX

```
\documentclass{book}

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

```

\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამხარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამხარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

New 3.49 Alternatively, you can tell babel to load all or some languages passed as options with `\babelprovide` and not from the `ldf` file in a few typical cases. Thus, `provide=*` means ‘load the main language with the `\babelprovide` mechanism instead of the `ldf` file’ applying the basic features, which in this case means `import, main`. There are (currently) three options:

- `provide=*` is the option just explained, for the main language;
- `provide+=*` is the same for additional languages (the main language is still the `ldf` file);
- `provide*=*` is the same for all languages, ie, main and additional.

EXAMPLE The preamble in the previous example can be more compactly written as:

```

\documentclass{book}
\usepackage[georgian, provide=*]{babel}
\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}

```

Or also:

```

\documentclass[georgian]{book}
\usepackage[provide=*]{babel}
\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}

```

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

```

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

```

Arabic Monolingual documents mostly work in `luatex`, but it must be fine tuned, particularly graphical elements like picture. In `xetex` babel resorts to the `bidi` package, which seems to work.

Hebrew Niqqud marks seem to work in both engines, but depending on the font cantillation marks might be misplaced (`xetex` or `luatex` with Harfbuzz seems better, but still problematic).

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

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

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

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1໐ 1໙ 1໑ 1໘ 1໓ 1໔} % Random
```

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

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

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

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

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

af	Afrikaans ^{ul}	bg	Bulgarian ^{ul}
agq	Aghem	bm	Bambara
ak	Akan	bn	Bangla ^{ul}
am	Amharic ^{ul}	bo	Tibetan ^u
ar	Arabic ^{ul}	brx	Bodo
ar-DZ	Arabic ^{ul}	bs-Cyrl	Bosnian
ar-MA	Arabic ^{ul}	bs-Latn	Bosnian ^{ul}
ar-SY	Arabic ^{ul}	bs	Bosnian ^{ul}
as	Assamese	ca	Catalan ^{ul}
asa	Asu	ce	Chechen
ast	Asturian ^{ul}	cgg	Chiga
az-Cyrl	Azerbaijani	chr	Cherokee
az-Latn	Azerbaijani	ckb	Central Kurdish
az	Azerbaijani ^{ul}	cop	Coptic
bas	Basaa	cs	Czech ^{ul}
be	Belarusian ^{ul}	cu	Church Slavic
bem	Bemba	cu-Cyrs	Church Slavic
bez	Bena	cu-Glag	Church Slavic

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

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

zh-Hant-MO	Chinese	zh	Chinese
zh-Hant	Chinese	zu	Zulu

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

aghem	burmese
akan	canadian
albanian	cantonese
american	catalan
amharic	centralatlastamazight
ancientgreek	centralkurdish
arabic	chechen
arabic-algeria	cherokee
arabic-DZ	chiga
arabic-morocco	chinese-hans-hk
arabic-MA	chinese-hans-mo
arabic-syria	chinese-hans-sg
arabic-SY	chinese-hans
armenian	chinese-hant-hk
assamese	chinese-hant-mo
asturian	chinese-hant
asu	chinese-simplified-hongkongsarchina
australian	chinese-simplified-macausarchina
austrian	chinese-simplified-singapore
azerbaijani-cyrillic	chinese-simplified
azerbaijani-cyrl	chinese-traditional-hongkongsarchina
azerbaijani-latin	chinese-traditional-macausarchina
azerbaijani-latn	chinese-traditional
azerbaijani	chinese
bafia	churchslavic
bambara	churchslavic-cyrs
basaa	churchslavic-oldcyrillic ¹²
basque	churchsslavic-glag
belarusian	churchsslavic-glagolitic
bemba	cognian
bena	cornish
bengali	croatian
bodo	czech
bosnian-cyrillic	danish
bosnian-cyrl	duala
bosnian-latin	dutch
bosnian-latn	dzongkha
bosnian	embu
brazilian	english-au
breton	english-australia
british	english-ca
bulgarian	english-canada

¹²The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

english-gb
english-newzealand
english-nz
english-unitedkingdom
english-unitedstates
english-us
english
esperanto
estonian
ewe
ewondo
faroese
filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi

kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali

newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
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
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq
telugu
teso
thai
tibetan
tigrinya
tongan
turkish
turkmen

ukenglish	vai-latn
ukrainian	vai-vai
uppersorbian	vai-vaii
urdu	vai
usenglish	vietnam
usorbian	vietnamese
uyghur	vunjo
uzbek-arab	walser
uzbek-arabic	welsh
uzbek-cyrillic	westernfrisian
uzbek-cyrl	yangben
uzbek-latin	yiddish
uzbek-latn	yoruba
uzbek	zarma
vai-latin	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=abcdefghijklj`. 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`.¹³

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

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

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

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

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

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

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

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

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 can replace the red line above with, say:

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

NOTE `\fontspec` is not touched at all, only the preset font families (`rm`, `sf`, `tt`, and the like). If a language is switched when an *ad hoc* font is active, or you select the font with this command, neither the script nor the language is passed. You must add them by hand. This is by design, for several reasons—for example, each font has its own set of features and a generic setting for several of them can be problematic, and also preserving 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 can 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. In the case of caption names a specific macro is provided, because this is perhaps the most frequent change:

`\setlocalecaption` $\{\langle\textit{language-name}\rangle\}\{\langle\textit{caption-name}\rangle\}\{\langle\textit{string}\rangle\}$

New 3.51 Here *caption-name* is the name as string without the trailing name. An example, which also shows caption names are often a stylistic choice, is:

```
\setlocalecaption{english}{contents}{Table of Contents}
```

This works not only with existing caption names, because it also serves to define new ones by setting the *caption-name* to the name of your choice (name will be postpended). Captions so defined or redefined behave with the ‘new way’ described in the following note.

NOTE There are a few alternative methods:

- With data imported 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.)

- 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. This redefinition is not activated until the language is selected.

- 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}
```

This redefinition is immediate.

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.

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

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

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

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

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

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

first loads danish.ldf, and then redefines the captions for danish (as provided by the ini file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some additional tools if provided by the ini file, like extra counters.

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: \chaptername not set for 'mylang'. Please,
(babel)                define it after the language has been loaded
(babel)                (typically in the preamble) with:
(babel)                \setlocalecaption{mylang}{chapter}{..}
(babel)                Reported on input line 26.
```

In most cases, you will only need to define a few macros. Note languages loaded on the fly are not yet available in the preamble.

EXAMPLE If you need a language named arhinish:

```
\usepackage[danish]{babel}
\babelprovide{arhinish}
\setlocalecaption{arhinish}{chapter}{Chapitula}
\setlocalecaption{arhinish}{refname}{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 and date (also line breaking rules in newly defined languages). 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 can be written:

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

There are about 250 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 may show a warning about the current lack of suitability of some features.

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

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

captions= $\langle\textit{language-tag}\rangle$

Loads only the strings. For example:

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

hyphenrules= $\langle\textit{language-list}\rangle$

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

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

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

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

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

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

New 3.58 Another special value is unhyphenated, which activates a line breking mode that allows spaces to be stretched to arbitrary amounts.

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

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

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

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

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

Remerber there is an alternative syntax for the latter:

```
\usepackage[italian, polytonicgreek, provide=*]{babel}
```

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

New 3.15 Sets the script name to be used by fontspec (eg, Devanagar i). 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.

alph= $\langle\text{counter-name}\rangle$

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

Alph= $\langle\text{counter-name}\rangle$

Same for `\Alph`.

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

onchar= `ids` | `fonts`

New 3.38 This option is much like an ‘event’ called when a character belonging to the script of this locale is found (as its name implies, it acts on characters, not on spaces). 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 or modified with `\babelcharproperty`.

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

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).

justification= `kashida` | `elongated` | `unhyphenated`

New 3.59 There are currently three options, mainly for the Arabic script. It sets the linebreaking and justification method, which can be based on the the ARABIC TATWEEL character or in the ‘justification alternatives’ OpenType table (`jalt`). For an explanation see the [babel site](#).

linebreaking= **New 3.59** Just a synonymous for justification.

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.

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

1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

NOTE With xetex you can use the option `Mapping` when defining a font.

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

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

- `\localenumerals{<style>}{<number>}`, like `\localenumerals{abjad}{15}`

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

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

The styles are:

Ancient Greek `lower.ancient, upper.ancient`
Amharic `afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebena, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yemsa`
Arabic `abjad, maghrebi.abjad`
Belarusan, Bulgarian, Macedonian, Serbian `lower, upper`
Bengali `alphabetic`
Coptic `epact, lower.letters`
Hebrew `letters (neither geresh nor gershayim yet)`
Hindi `alphabetic`
Armenian `lower.letter, upper.letter`
Japanese `hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha`
Georgian `letters`
Greek `lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)`
Khmer `consonant`
Korean `consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha`
Marathi `alphabetic`
Persian `abjad, alphabetic`
Russian `lower, lower.full, upper, upper.full`
Syriac `letters`
Tamil `ancient`
Thai `alphabetic`
Ukrainian `lower, lower.full, upper, upper.full`
Chinese `cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha`

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

\iflanguage `{\language}{\true}{\false}`

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

\localeinfo `{\field}`

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

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

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

`tag.bcp47` is the full BCP 47 tag (see the warning below).

`language.tag.bcp47` is the BCP 47 language tag.

`tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).

`script.name`, as provided by the Unicode CLDR.

`script.tag.bcp47` is the BCP 47 tag of the script used by this locale.

`script.tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).

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

\getlocaleproperty `*{\macro}{\locale}{\property}`

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

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

the macro `\hechap` will contain the string פֶּרֶק.

If the key does not exist, the macro is set to `\relax` and an error is raised. **New 3.47** With the starred version no error is raised, so that you can take your own actions with undefined properties.

Babel remembers which ini files have been loaded. There is a loop named

`\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that

`\LocaleForEach{\message{ **#1** }}` just shows the loaded ini's.

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

There are also some differences with `LaTeX`: (1) the character used is that set for the current font, while in `LaTeX` it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in `LaTeX`, but it can be changed to another value by redefining `\babenullhyphen`; (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.

NOTE To set hyphenation exceptions in the preamble before any language is explicitly set with a selector, use \babelhyphenation instead of \hyphenation. In the preamble the hyphenation rules are not always fully set up and an error can be raised.

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

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

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

New 3.9m *In luatex only*,¹⁴ 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

¹⁴With 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).

1.21 Transforms

Transforms (only luatex) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.¹⁵

It currently embraces `\babelprehyphenation` and `\babelposthyphenation`.

New 3.57 Several ini files predefine some transforms. They are activated with the key `transforms` in `\babelprovide`, either if the locale is being defined with this macro or the languages has been previously loaded as a class or package option, as the following example illustrates:

```
\usepackage[magyar]{babel}
\babelprovide[transforms = digraphs.hyphen]{magyar}
```

Here are the transforms currently predefined. (More to follow in future releases.)

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for dad (simple and T _E X-friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	<code>oneletter.nobreak</code>	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: <i>!?:;</i> .
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zsz</i> as <i>cs-cs, dz-dz</i> , etc.

¹⁵They are similar in concept, but not the same, as those in Unicode.

Indic scripts	danda.nobreak	Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.
Arabic, Persian	kashida.plain	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.
Serbian	transliteration.gajica	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.

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

New 3.37-3.39 With *luatex* it is 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. A few rules are currently provided (see above), 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 $([\text{t}\acute{o}])$, the replacement could be $\{1|\text{t}\acute{o}|\text{t}\acute{o}\}$, which maps $\text{t}\acute{}$ to t , and \acute{o} to $\text{t}\acute{o}$, so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`. See the [babel site](#) for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

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).

You are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.

\babelprehyphenation $\{\langle locale-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

New 3.44-3.52 It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences: (1) the first argument is the locale instead of the name of the hyphenation patterns; (2) in the search patterns = has no special meaning, while | stands for an ordinary space; (3) in the replacement, discretionaries are not accepted. This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`.

EXAMPLE You can 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
\babelprehyphenation{russian-latin}{([sz])h} % Create rule
{
  string = {1|sz|šž},
  remove
}

```

EXAMPLE The following rule prevent the word “a” from being at the end of a line:

```

\babelprehyphenation{english}{|a|}
{ }, { }, % Keep first space and a
{ insert, penalty = 10000 }, % Insert penalty
{ } % Keep last space
}

```

NOTE With luatex there is another approach to make text transformations, with the function `fonts.handlers.otf.addfeature`, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and babel by default recognizes this setting if the font has been declared with `\babelfont`. The *transforms* mechanism supplements rather than replaces OTF features.

With xetex, where *transforms* are not available, there is still another approach, with font mappings, mainly meant to perform encoding conversions and transliterations. Mappings, however, are linked to fonts, not to languages.

1.22 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}

\begin{document}

```

```
Chapter in Danish: \chaptername.
```

```
\selectlanguage{de-AT}
```

```
\localedate{2020}{1}{30}
```

```
\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:

`autoload.bcp47` with values on and off.

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

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

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

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

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

1.23 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.¹⁶ Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.¹⁷

`\ensureascii` $\langle text \rangle$

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

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

¹⁷But still defined for backwards compatibility.

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.

1.24 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 can 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 can be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including the picture environment (with `pict2e`) and `pfg/tikz`. Also, indexes and the like are under study, as well as math (there is progress in the latter, too, but for example cases may fail).

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

    وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الاجريقي) بـ
    Arabia أو Aravia (بالاغريقية (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 فصحى العصر \textit{fuṣḥā l-‘aṣr} (MSA) and
    فصحى التراث \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

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

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

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`, so that the first column is the right one (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. 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:

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

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

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

Digits in pdfTeX must be marked up explicitly (unlike LaTeX with `bidi=basic` or `bidi=basic-r` and, usually, `xetex`). This command is provided to set $\{\langle lr\text{-}text\rangle\}$ 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 $\{\langle section\text{-}name\rangle\}$

Mainly for bidi text, but it can 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 $\{\langle cmd\rangle\}\{\langle local\text{-}language\rangle\}\{\langle before\rangle\}\{\langle after\rangle\}$

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.

1.25 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).

1.26 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 with a certain `{<name>}` 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 $\mathrm{T\!E\!X}$ 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).

1.27 Languages supported by babel with ldf files

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

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton
Bulgarian bulgarian
Catalan catalan
Croatian croatian
Czech czech
Danish danish

Dutch dutch
English english, USenglish, american, UKenglish, british, canadian, australian, newzealand
Esperanto esperanto
Estonian estonian
Finnish finnish
French french, francais, canadien, acadian
Galician galician
German austrian, german, germanb, ngerman, naustrian
Greek greek, polutonikogreek
Hebrew hebrew
Icelandic icelandic
Indonesian indonesian (bahasa, indon, bahasai)
Interlingua interlingua
Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu (bahasam)
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, brazilian (portuges, brazil)¹⁹
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppsorbian
Welsh welsh

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

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

```

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

```

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

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

1.28 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).

`\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 `\babelprovide`, or, if the last argument is empty, removes them. The last argument is the locale name:

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

1.29 Tweaking some features

`\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`, `justify.arabic`. For example, you can set `\babeladjust{bidi.text=off}` if you are using an alternative algorithm or with large sections not requiring it. Use with care, because these options do not deactivate other related options (like paragraph direction with `bidi.text`).

1.30 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`), \LaTeX will keep complaining about an undefined label. To prevent such problems, you can 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}}
```

- For the hyphenation to work correctly, lccodes cannot change, because T_EX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.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 T_EX, 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 a similar issue with floats, too. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make T_EX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

microtype Adjusts the typesetting according to some languages (kerning and spacing).
Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.31 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 L^AT_EX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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

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

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

New 3.51 `\babeladjust{ autoload.options = ... }` sets the options when a language is loaded on the fly (by default, no options). A typical value would be `import`, which defines captions, date, numerals, etc., but ignores the code in the tex file (for example, extended numerals in Greek).

Labels

New 3.48 There is some work in progress for babel to deal with labels, both with the relation to captions (chapters, part), and how counters are used to define them. It is still somewhat tentative because it is far from trivial – see the wiki for further details.

2 Loading languages with `language.dat`

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

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

2.1 Format

In that file the person who maintains a \TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored²⁴. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct \LaTeX that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

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.

```
% 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 can be set in `\extras⟨lang⟩`).

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\⟨lang⟩hyphenmins`, `\captions⟨lang⟩`, `\date⟨lang⟩`, `\extras⟨lang⟩` and `\noextras⟨lang⟩` (the last two may be left empty); where `⟨lang⟩` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are

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

discussed below. You must define all or none for a language (or a dialect); defining, say, `\date⟨lang⟩` but not `\captions⟨lang⟩` does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in \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⟨lang⟩` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras⟨lang⟩`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.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

Currently, the easiest way to contribute a new language is by taking one of the 500 or so ini templates available on GitHub as a basis. Just make a pull request or download it and then, after filling the fields, send it to me. Feel free to ask for help or to make feature requests.

As to ldf files, 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.

²⁶But not removed, for backward compatibility.

- 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, ot f, 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 for ldf files:

<http://www.texnia.com/incubator.html>. See also

<https://latex3.github.io/babel/guides/list-of-locale-templates.html>.

If you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

\noextras<lang> 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 \extras<lang>, a macro that brings TeX into a predefined state is needed. It will be no surprise that the name of this macro is \noextras<lang>.

<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]
\LdfInit{<language>}{captions<language>}

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

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

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

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

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

\StartBabelCommands*{<language>}{date}

```



```

\SetString\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

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

`\bbl@activate` The command `\bbl@activate` is used to change the way an active character expands.

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

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

`\bbl@add@special` The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]
`\bbl@remove@special` It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

Language definition files may want to redefine macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this²⁷.

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

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

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

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

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

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

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

3.8 Encoding-dependent strings

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

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

²⁷This mechanism was introduced by Bernd Raichle.

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

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

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

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

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

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

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

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

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

²⁸In future releases further categories may be added.

```

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

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

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

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


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

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

\EndBabelCommands

```

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

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

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 $\backslash EndBabelCommands$.

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

\SetString {*<macro-name>*}{*<string>*}

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

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

\SetStringLoop {*<macro-name>*}{*<string-list>*}

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

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

#1 is replaced by the roman numeral.

\SetCase [*<map-list>*]{*<toupper-code>*}{*<tolower-code>*}

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

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

\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
{\uccode`i=`I\relax
 \uccode`1=`I\relax}
{\lccode`I=`i\relax
 \lccode`I=`1\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 \TeX for two unrelated purposes: case transforms (upper/lower) and hyphenation. `\SetCase` handles the former, while hyphenation is handled by `\SetHyphenMap` and controlled with the package option `hyphenmap`. So, even if internally they are based on the same \TeX primitive (`\lccode`), babel sets them separately.

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

7 Tools

```
1 <<version=3.63.2449>>
2 <<date=2021/07/30>>
```

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

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

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

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

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

`\bbl@afterelse` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³⁰. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```
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}
```

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

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

```

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

```

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

```

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

```

`\bbl@ifblank` A tool from `url`, by Donald Arseneau, which tests if a string is empty or space. The companion macros tests if a macro is defined with some ‘real’ value, ie, not `\relax` and not empty,

```

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}
71 \def\bbl@ifset#1#2#3{%
72   \bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{#1}}{#3}{#2}}}

```

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).

```

73 \def\bbl@forkv#1#2{%
74   \def\bbl@kvcmd##1##2##3{#2}%
75   \bbl@kvnext#1,\@nil,}
76 \def\bbl@kvnext#1,{%

```

```

77 \ifx\@nil#1\relax\else
78 \bbl@ifblank{#1}{\bbl@forkv@eq#1=@empty=@nil{#1}}%
79 \expandafter\bbl@kvnext
80 \fi}
81 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
82 \bbl@trim@def\bbl@forkv@a{#1}%
83 \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).

```

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

```

\bbl@replace Returns implicitly \toks@ with the modified string.

```

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

```

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

```

105 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
106 \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
107 \def\bbl@tempa{#1}%
108 \def\bbl@tempb{#2}%
109 \def\bbl@tempe{#3}}
110 \def\bbl@sreplace#1#2#3{%
111 \begingroup
112 \expandafter\bbl@parsedef\meaning#1\relax
113 \def\bbl@tempc{#2}%
114 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
115 \def\bbl@tempd{#3}%
116 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
117 \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
118 \ifin@
119 \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
120 \def\bbl@tempc{% Expanded an executed below as 'uplevel'
121 \\makeatletter % "internal" macros with @ are assumed
122 \\scantokens{%
123 \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
124 \catcode64=\the\catcode64\relax}% Restore @

```

```

125     \else
126       \let\bbl@tempc\@empty % Not \relax
127     \fi
128     \bbl@exp{%      For the 'uplevel' assignments
129   \endgroup
130     \bbl@tempc}} % empty or expand to set #1 with changes
131 \fi

```

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

```

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

```

A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

```

154 \def\bbl@bsphack{%
155   \ifhmode
156     \hskip\z@skip
157     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158   \else
159     \let\bbl@esphack\@empty
160   \fi}

```

Another hackish tool, to apply case changes inside a protected macros. It's based on the internal `\let's` made by `\MakeUppercase` and `\MakeLowercase` between things like `\oe` and `\OE`.

```

161 \def\bbl@cased{%
162   \ifx\oe\OE
163     \expandafter\in@\expandafter
164       {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166       \bbl@afterelse\expandafter\MakeUppercase
167     \else
168       \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170   \else
171     \expandafter\@firstofone
172   \fi}

```

An alternative to `\IfFormatAtLeastTF` for old versions. Temporary.

```

173 \ifx\IfFormatAtLeastTF\undefined
174 \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
175 \else
176 \let\bbl@ifformatlater\IfFormatAtLeastTF
177 \fi

```

The following adds some code to `\extras...` both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with #'s.

```

178 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
179 \toks@{\expandafter\expandafter\expandafter{%
180 \csname extras\language\endcsname}%
181 \bbl@exp{\in@{#1}{\the\toks@}}}%
182 \ifin@ \else
183 \@temptokena{#2}%
184 \edef\bbl@tempc{\the\@temptokena\the\toks@}%
185 \toks@\expandafter{\bbl@tempc#3}%
186 \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
187 \fi}
188 <</Basic macros>>

```

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

```

189 <<*Make sure ProvidesFile is defined>> ≡
190 \ifx\ProvidesFile\undefined
191 \def\ProvidesFile#1[#2 #3 #4]{%
192 \wlog{File: #1 #4 #3 <#2>}%
193 \let\ProvidesFile\undefined}
194 \fi
195 <</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.

```

196 <<*Define core switching macros>> ≡
197 \ifx\language\undefined
198 \csname newcount\endcsname\language
199 \fi
200 <</Define core switching macros>>

```

`\last@language` Another counter is used to keep track of the allocated languages. \TeX and \LaTeX reserves for this purpose the count 19.

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

```

201 <<*Define core switching macros>> ≡
202 \countdef\last@language=19
203 \def\addlanguage{\csname newlanguage\endcsname}
204 <</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. 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 (L^AT_EX, 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.

```
205 (*package)
206 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
207 \ProvidesPackage{babel}[\langle\date\rangle\ \langle\version\rangle The Babel package]
208 \@ifpackagewith{babel}{debug}
209   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
210    \let\bbl@debug\@firstofone
211    \ifx\directlua\@undefined\else
212      \directlua{ Babel = Babel or {}
213                Babel.debug = true }%
214      \input{babel-debug.tex}%
215    \fi}
216 {\providecommand\bbl@trace[1]{}%
217  \let\bbl@debug\gobble
218  \ifx\directlua\@undefined\else
219    \directlua{ Babel = Babel or {}
220              Babel.debug = false }%
221  \fi}
222 \langle\Basic macros\rangle
223 % Temporarily repeat here the code for errors. TODO.
224 \def\bbl@error#1#2{%
225   \begingroup
226     \def\{\MessageBreak}%
227     \PackageError{babel}{#1}{#2}%
228   \endgroup}
229 \def\bbl@warning#1{%
230   \begingroup
231     \def\{\MessageBreak}%
232     \PackageWarning{babel}{#1}%
233   \endgroup}
234 \def\bbl@infowarn#1{%
235   \begingroup
236     \def\{\MessageBreak}%
237     \GenericWarning
238       {(babel) \@spaces\@spaces\@spaces}%
239       {Package babel Info: #1}%
240   \endgroup}
241 \def\bbl@info#1{%
242   \begingroup
243     \def\{\MessageBreak}%
244     \PackageInfo{babel}{#1}%
245   \endgroup}
246 \def\bbl@nocaption{\protect\bbl@nocaption@i}
247 % TODO - Wrong for \today !!! Must be a separate macro.
248 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
249   \global\@namedef{#2}{\textbf{?#1?}}%
250   \@nameuse{#2}%
251   \edef\bbl@tempa{#1}%
252   \bbl@sreplace\bbl@tempa{name}{}}%
253 \bbl@warning{%
254   \@backslashchar#1 not set for '\language'. Please,\%
255   define it after the language has been loaded\%
```

```

256 (typically in the preamble) with\\%
257 \string\setlocalecaption{\language\name}{\bbl@tempa}{..}\\%
258 Reported}}
259 \def\bbl@tentative{\protect\bbl@tentative@i}
260 \def\bbl@tentative@i#1{%
261 \bbl@warning{%
262 Some functions for '#1' are tentative.\\%
263 They might not work as expected and their behavior\\%
264 may change in the future.\\%
265 Reported}}
266 \def\@nolanerr#1{%
267 \bbl@error
268 {You haven't defined the language '#1' yet.\\%
269 Perhaps you misspelled it or your installation\\%
270 is not complete}%
271 {Your command will be ignored, type <return> to proceed}}
272 \def\@nopatterns#1{%
273 \bbl@warning
274 {No hyphenation patterns were preloaded for\\%
275 the language '#1' into the format.\\%
276 Please, configure your TeX system to add them and\\%
277 rebuild the format. Now I will use the patterns\\%
278 preloaded for \bbl@nulllanguage\space instead}}
279 % End of errors
280 \@ifpackagewith{babel}{silent}
281 {\let\bbl@info\@gobble
282 \let\bbl@infowarn\@gobble
283 \let\bbl@warning\@gobble}
284 {}
285 %
286 \def\AfterBabelLanguage#1{%
287 \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.

288 \ifx\bbl@languages\undefined\else
289 \begingroup
290 \catcode\^^I=12
291 \@ifpackagewith{babel}{showlanguages}{%
292 \begingroup
293 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
294 \wlog{<*languages>}%
295 \bbl@languages
296 \wlog{</languages>}%
297 \endgroup}{%
298 \endgroup
299 \def\bbl@elt#1#2#3#4{%
300 \ifnum#2=\z@
301 \gdef\bbl@nulllanguage{#1}%
302 \def\bbl@elt##1##2##3##4{}}%
303 \fi}%
304 \bbl@languages
305 \fi%

```

7.3 base

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

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

```

306 \bbl@trace{Defining option 'base'}
307 \@ifpackagewith{babel}{base}{%
308   \let\bbl@onlyswitch\@empty
309   \let\bbl@provide@locale\relax
310   \input babel.def
311   \let\bbl@onlyswitch\@undefined
312   \ifx\directlua\@undefined
313     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
314   \else
315     \input luababel.def
316     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
317   \fi
318   \DeclareOption{base}{}%
319   \DeclareOption{showlanguages}{}%
320   \ProcessOptions
321   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
322   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
323   \global\let\@ifl@ter@@\@ifl@ter
324   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
325   \endinput}{}%
326 % \end{macrocode}
327 %
328 % \subsection{\texttt{key=value} options and other general option}
329 %
330 %   The following macros extract language modifiers, and only real
331 %   package options are kept in the option list. Modifiers are saved
332 %   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
333 %   no modifiers have been given, the former is |\relax|. How
334 %   modifiers are handled are left to language styles; they can use
335 %   |\in@|, loop them with |\@for| or load |keyval|, for example.
336 %
337 %   \begin{macrocode}
338 \bbl@trace{key=value and another general options}
339 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
340 \def\bbl@tempb#1.#2{% Remove trailing dot
341   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
342 \def\bbl@tempd#1.#2@nnil{% TODO. Refactor lists?
343   \ifx\@empty#2%
344     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
345   \else
346     \in@{,provide=}{, #1}%
347     \ifin@
348       \edef\bbl@tempc{%
349         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
350   \else
351     \in@{=}{ #1}%
352     \ifin@
353       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
354   \else
355     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
356     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
357   \fi
358   \fi
359   \fi}
360 \let\bbl@tempc\@empty
361 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}

```

```
362 \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.

```
363 \DeclareOption{KeepShorthandsActive}{}
364 \DeclareOption{activeacute}{}
365 \DeclareOption{activegrave}{}
366 \DeclareOption{debug}{}
367 \DeclareOption{noconfigs}{}
368 \DeclareOption{showlanguages}{}
369 \DeclareOption{silent}{}
370 % \DeclareOption{mono}{}
371 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
372 \chardef\bbl@iniflag\z@
373 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
374 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
375 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
376 % A separate option
377 \let\bbl@autoload@options\@empty
378 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
379 % Don't use. Experimental. TODO.
380 \newif\ifbbl@single
381 \DeclareOption{selectors=off}{\bbl@singletrue}
382 <<More package options>>
```

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax <key>=<value>, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```
383 \let\bbl@opt@shorthands\@nnil
384 \let\bbl@opt@config\@nnil
385 \let\bbl@opt@main\@nnil
386 \let\bbl@opt@headfoot\@nnil
387 \let\bbl@opt@layout\@nnil
388 \let\bbl@opt@provide\@nnil
```

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

```
389 \def\bbl@tempa#1=#2\bbl@tempa{%
390   \bbl@csarg\ifx{opt@#1}\@nnil
391     \bbl@csarg\edef{opt@#1}{#2}%
392   \else
393     \bbl@error
394     {Bad option '#1=#2'. Either you have misspelled the\\%
395     key or there is a previous setting of '#1'. Valid\\%
396     keys are, among others, 'shorthands', 'main', 'bidi',\\%
397     'strings', 'config', 'headfoot', 'safe', 'math'.}%
398     {See the manual for further details.}
399   \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.

```
400 \let\bbl@language@opts\@empty
401 \DeclareOption*{%
402   \bbl@xin@{\string=}{\CurrentOption}%
403   \ifin@
404     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
405   \else
```



```

406 \bbl@xin@{\CurrentOption,}{\bbl@language@opts,}%
407 \ifin@
408 \bbl@exp{\bbl@replace\bbl@language@opts{\CurrentOption,}}}%
409 \fi
410 \edef\bbl@language@opts{\bbl@language@opts,\CurrentOption,}
411 \fi}

```

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

```

412 \ProcessOptions*
413 \ifx\bbl@opt@provide\@nnil
414 \let\bbl@opt@provide\@empty %%%% MOVE above
415 \else
416 \chardef\bbl@iniflag\@ne
417 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}%
418 \in@{,provide,}{, #1,}%
419 \ifin@
420 \def\bbl@opt@provide{#2}%
421 \bbl@replace\bbl@opt@provide{;}{,}%
422 \fi
423 \fi
424 %

```

7.4 Conditional loading of shorthands

If there is no shorthands=<chars>, the original babel macros are left untouched, but if there is, these macros are wrapped (in babel.def) to define only those given.

A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=...

```

425 \bbl@trace{Conditional loading of shorthands}
426 \def\bbl@sh@string#1{%
427 \ifx#1\@empty\else
428 \ifx#1t\string~%
429 \else\ifx#1c\string,%
430 \else\string#1%
431 \fi\fi
432 \expandafter\bbl@sh@string
433 \fi}
434 \ifx\bbl@opt@shorthands\@nnil
435 \def\bbl@ifshorthand#1#2#3{#2}%
436 \else\ifx\bbl@opt@shorthands\@empty
437 \def\bbl@ifshorthand#1#2#3{#3}%
438 \else

```

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

```

439 \def\bbl@ifshorthand#1{%
440 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
441 \ifin@
442 \expandafter\@firstoftwo
443 \else
444 \expandafter\@secondoftwo
445 \fi}

```

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

```

446 \edef\bbl@opt@shorthands{%
447 \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

448 \bbl@ifshorthand{'}%
449 {\PassOptionsToPackage{activeacute}{babel}}{}
450 \bbl@ifshorthand{'}%
451 {\PassOptionsToPackage{activegrave}{babel}}{}
452 \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.

```

453 \ifx\bbl@opt@headfoot\@nnil\else
454 \g@addto@macro\@resetactivechars{%
455 \set@typeset@protect
456 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
457 \let\protect\noexpand}
458 \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.

```

459 \ifx\bbl@opt@safe\@undefined
460 \def\bbl@opt@safe{BR}
461 \fi

```

Make sure the language set with ‘main’ is the last one.

```

462 \ifx\bbl@opt@main\@nnil\else
463 \edef\bbl@language@opts{\bbl@language@opts,\bbl@opt@main,}
464 \fi

```

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

```

465 \bbl@trace{Defining IfBabelLayout}
466 \ifx\bbl@opt@layout\@nnil
467 \newcommand\IfBabelLayout[3]{#3}%
468 \else
469 \newcommand\IfBabelLayout[1]{%
470 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
471 \ifin@
472 \expandafter\@firstoftwo
473 \else
474 \expandafter\@secondoftwo
475 \fi}
476 \fi

```

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

```

477 \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.

```

478 <<(*More package options)>> \equiv
479 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
480 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
481 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
482 <</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.

```

483 \bbl@trace{Cross referencing macros}
484 \ifx\bbl@opt@safe\empty\else
485   \def\@newl@bel#1#2#3{%
486     {\@safe@activestrue
487       \bbl@ifunset{#1@#2}%
488       \relax
489       {\gdef\@multiplelabels{%
490         \@latex@warning@no@line{There were multiply-defined labels}}}%
491       \@latex@warning@no@line{Label `#2' multiply defined}}}%
492   \global\@namedef{#1@#2}{#3}}

```

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

```

493 \CheckCommand*\@testdef[3]{%
494   \def\reserved@a{#3}%
495   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
496   \else
497     \@tempswatrue
498   \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.

```

499 \def\@testdef#1#2#3{% TODO. With @samestring?
500   \@safe@activestrue
501   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
502   \def\bbl@tempb{#3}%
503   \@safe@activesfalse
504   \ifx\bbl@tempa\relax
505   \else
506     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
507   \fi
508   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
509   \ifx\bbl@tempa\bbl@tempb
510   \else
511     \@tempswatrue
512   \fi}
513 \fi

```

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

```

514 \bbl@xin@{R}\bbl@opt@safe
515 \ifin@
516   \bbl@redefineroobust\ref#1{%
517     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
518   \bbl@redefineroobust\pageref#1{%
519     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
520 \else
521   \let\org@ref\ref
522   \let\org@pageref\pageref
523 \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.

```
524 \bbl@xin@{B}\bbl@opt@safe
525 \ifin@
526 \bbl@redefine\@citex[#1]#2{%
527   \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
528   \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.

```
529 \AtBeginDocument{%
530   \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.)

```
531   \def\@citex[#1][#2]#3{%
532     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
533     \org@@citex[#1][#2]{\@tempa}}%
534   }{}}
```

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

```
535 \AtBeginDocument{%
536   \ifpackageloaded{cite}{%
537     \def\@citex[#1]#2{%
538       \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
539     }{}}
```

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

```
540 \bbl@redefine\nocite#1{%
541   \@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.

```
542 \bbl@redefine\bibcite{%
543   \bbl@cite@choice
544   \bibcite}
```

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

```
545 \def\bbl@bibcite#1#2{%
546   \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.

```
547 \def\bbl@cite@choice{%
548   \global\let\bibcite\bbl@bibcite}
```

```

549 \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
550 \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
551 \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.

```

552 \AtBeginDocument{\bbl@cite@choice}

```

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

```

553 \bbl@redefine\@bibitem#1{%
554   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
555 \else
556   \let\org@nocite\nocite
557   \let\org@@citex\@citex
558   \let\org@bibcite\bibcite
559   \let\org@@bibitem\@bibitem
560 \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.

```

561 \bbl@trace{Marks}
562 \IfBabelLayout{sectioning}
563   {\ifx\bbl@opt@headfoot\@nnil
564     \g@addto@macro\@resetactivechars{%
565       \set@typeset@protect
566       \expandafter\select@language@x\expandafter{\bbl@main@language}%
567       \let\protect\noexpand
568       \ifcase\bbl@bidimode\else % Only with bidi. See also above
569         \edef\thepage{%
570           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
571       \fi}%
572   \fi}
573 {\ifbbl@single\else
574   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
575   \markright#1{%
576     \bbl@ifblank{#1}%
577     {\org@markright{}}%
578     {\toks@{#1}%
579       \bbl@exp{%
580         \\org@markright{\\protect\\foreignlanguage{\language}\thepage}%
581         {\\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

582   \ifx\@mkboth\markboth
583     \def\bbl@tempc{\let\@mkboth\markboth}
584   \else
585     \def\bbl@tempc{}

```

```

586 \fi
587 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefineroobust
588 \markboth#1#2{%
589 \protected@edef\bbl@tempb##1{%
590 \protect\foreignlanguage
591 {\language}\protect\bbl@restore@actives##1}}%
592 \bbl@ifblank{#1}%
593 {\toks@{}}%
594 {\toks@\expandafter{\bbl@tempb{#1}}}%
595 \bbl@ifblank{#2}%
596 {\@temptokena{}}%
597 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
598 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
599 \bbl@tempc
600 \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.

```

601 \bbl@trace{Preventing clashes with other packages}
602 \bbl@xin@{R}\bbl@opt@safe
603 \ifin@
604 \AtBeginDocument{%
605 \ifpackageloaded{ifthen}{%
606 \bbl@redefine@long\ifthenelse#1#2#3{%
607 \let\bbl@temp@pref\pageref
608 \let\pageref\org@pageref
609 \let\bbl@temp@ref\ref
610 \let\ref\org@ref
611 \@safe@activestrue
612 \org@ifthenelse{#1}%
613 {\let\pageref\bbl@temp@pref
614 \let\ref\bbl@temp@ref
615 \@safe@activesfalse
616 #2}%
617 {\let\pageref\bbl@temp@pref
618 \let\ref\bbl@temp@ref
619 \@safe@activesfalse
620 #3}%
621 }%
622 }}%
623 }

```

7.7.2 varioref

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

```
624 \AtBeginDocument{%
625   \ifpackageloaded{varioref}{%
626     \bbl@redefine\@@vpageref#1[#2]#3{%
627       \@safe@activetrue
628       \org@@vpageref{#1}[#2]{#3}%
629       \@safe@activesfalse}%
630   \bbl@redefine\vrefpagenum#1#2{%
631     \@safe@activetrue
632     \org\vrefpagenum{#1}{#2}%
633     \@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.

```
634   \expandafter\def\csname Ref \endcsname#1{%
635     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
636   }{}%
637 }
638 \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.

```
639 \AtEndOfPackage{%
640   \AtBeginDocument{%
641     \ifpackageloaded{hhline}%
642       {\expandafter\ifx\csname normal@char\string\endcsname\relax
643         \else
644           \makeatletter
645           \def\@currname{hhline}\input{hhline.sty}\makeatother
646           \fi}%
647       {}}}
```

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? Commented out in 2020/07/27.

```
648 % \AtBeginDocument{%
649 %   \ifx\pdfstringdefDisableCommands\@undefined\else
650 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
651 %   \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.

```
652 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
653   \lowercase{\foreignlanguage{#1}}}
```

`\substitutefontfamily` This command is deprecated. Use the tools provides by \TeX . 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.

```
654 \def\substitutefontfamily#1#2#3{%
655   \lowercase{\immediate\openout15=#1#2.fd\relax}%
656   \immediate\write15{%
657     \string\ProvidesFile{#1#2.fd}%
658     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
659     \space generated font description file]^^J
660     \string\DeclareFontFamily{#1}{#2}{^^J
661       \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
662       \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
663       \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
664       \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
665       \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
666       \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
667       \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
668       \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
669     }%
670   \closeout15
671 }
672 \@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. Requested encodings are currently stored in `\@fontenc@load@list`. 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`

```
673 \bbl@trace{Encoding and fonts}
674 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
675 \newcommand\BabelNonText{TS1,T3,TS3}
676 \let\org@TeX\TeX
677 \let\org@LaTeX\LaTeX
678 \let\ensureascii\@firstofone
679 \AtBeginDocument{%
680   \def\@elt#1{, #1,}%
681   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
682   \let\@elt\relax
683   \let\bbl@tempb\@empty
684   \def\bbl@tempc{OT1}%
685   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
686     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
687   \bbl@foreach\bbl@tempa{%
688     \bbl@xin@{#1}{\BabelNonASCII}%
689     \ifin@
690       \def\bbl@tempb{#1}% Store last non-ascii
691     \else\bbl@xin@{#1}{\BabelNonText}% Pass
692     \ifin@
693       \def\bbl@tempc{#1}% Store last ascii
```



```

694     \fi
695     \fi}%
696 \ifx\bb1@tempb\@empty\else
697     \bb1@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
698     \ifin@\else
699         \edef\bb1@tempc{\cf@encoding}% The default if ascii wins
700     \fi
701     \edef\ensureascii#1{%
702         {\noexpand\fontencoding{\bb1@tempc}\noexpand\selectfont#1}}%
703     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
704     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
705 \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.

```

706 \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.

```

707 \AtBeginDocument{%
708     \@ifpackageloaded{fontspec}%
709     {\xdef\latinencoding{%
710         \ifx\UTFencname\@undefined
711             EU\ifcase\bb1@engine\or2\or1\fi
712         \else
713             \UTFencname
714         \fi}}%
715     {\gdef\latinencoding{OT1}%
716         \ifx\cf@encoding\bb1@t@one
717             \xdef\latinencoding{\bb1@t@one}%
718         \else
719             \def\@elt#1{, #1,}%
720             \edef\bb1@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
721             \let\@elt\relax
722             \bb1@xin@{,T1,}\bb1@tempa
723             \ifin@
724                 \xdef\latinencoding{\bb1@t@one}%
725             \fi
726         \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.

```

727 \DeclareRobustCommand{\latintext}{%
728     \fontencoding{\latinencoding}\selectfont
729     \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.

```

730 \ifx\@undefined\DeclareTextFontCommand
731     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
732 \else
733     \DeclareTextFontCommand{\textlatin}{\latintext}
734 \fi

```

For several functions, we need to execute some code with `\selectfont`. With \LaTeX 2021-06-01, there is a hook for this purpose, but in older versions the \LaTeX command is patched (the latter solution will be eventually removed).

```
735 \bbl@ifformatlater{2021-06-01}%
736 {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
737 {\def\bbl@patchfont#1{%
738   \expandafter\bbl@add\csname selectfont \endcsname{#1}%
739   \expandafter\bbl@tglobal\csname selectfont \endcsname}}
```

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.

```
740 \bbl@trace{Loading basic (internal) bidi support}
741 \ifodd\bbl@engine
742 \else % TODO. Move to txtbabel
743   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
744     \bbl@error
745     {The bidi method 'basic' is available only in\\%
746     luatex. I'll continue with 'bidi=default', so\\%
747     expect wrong results}%
748     {See the manual for further details.}%
749     \let\bbl@beforeforeign\leavevmode
750     \AtEndOfPackage{%
751       \EnableBabelHook{babel-bidi}%
752       \bbl@xebidipar}
753   \fi\fi
754   \def\bbl@loadxebidi#1{%
755     \ifx\RTLfootnotetext\@undefined
756       \AtEndOfPackage{%
757         \EnableBabelHook{babel-bidi}%
758         \ifx\fontspec\@undefined
759           \bbl@loadfontspec % bidi needs fontspec
760         \fi
761         \usepackage#1{bidi}}%
762     \fi}
763   \ifnum\bbl@bidimode>200
764     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
765       \bbl@tentative{bidi=bidi}
766       \bbl@loadxebidi{}
```

```

767 \or
768 \bbl@loadxebidi{[rldocument]}
769 \or
770 \bbl@loadxebidi{}
771 \fi
772 \fi
773 \fi
774 % TODO? Separate:
775 \ifnum\bbl@bidimode=\@ne
776 \let\bbl@beforeforeign\leavevmode
777 \ifodd\bbl@engine
778 \newattribute\bbl@attr@dir
779 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
780 \bbl@exp{\output{\bodydir\pagedir\the\output}}
781 \fi
782 \AtEndOfPackage{%
783 \EnableBabelHook{babel-bidi}%
784 \ifodd\bbl@engine\else
785 \bbl@xebidipar
786 \fi}
787 \fi

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

788 \bbl@trace{Macros to switch the text direction}
789 \def\bbl@alscripts{Arabic,Syriac,Thaana,}
790 \def\bbl@rscripts{% TODO. Base on codes ??
791 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
792 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
793 Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
794 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
795 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
796 Old South Arabian,}%
797 \def\bbl@provide@dirs#1{%
798 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
799 \ifin@
800 \global\bbl@csarg\chardef{wdir@#1}\@ne
801 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
802 \ifin@
803 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
804 \fi
805 \else
806 \global\bbl@csarg\chardef{wdir@#1}\z@
807 \fi
808 \ifodd\bbl@engine
809 \bbl@csarg\ifcase{wdir@#1}%
810 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
811 \or
812 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
813 \or
814 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
815 \fi
816 \fi}
817 \def\bbl@switchdir{%
818 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
819 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
820 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
821 \def\bbl@setdirs#1{% TODO - math
822 \ifcase\bbl@select@type % TODO - strictly, not the right test

```

```

823 \bbl@bodydir{#1}%
824 \bbl@pardir{#1}%
825 \fi
826 \bbl@textdir{#1}}
827% TODO. Only if \bbl@bidimode > 0?:
828 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
829 \DisableBabelHook{babel-bidi}

```

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

```

830 \ifodd\bbl@engine % luatex=1
831 \else % pdftex=0, xetex=2
832 \newcount\bbl@dirlevel
833 \chardef\bbl@thetextdir\z@
834 \chardef\bbl@thepardir\z@
835 \def\bbl@textdir#1{%
836 \ifcase#1\relax
837 \chardef\bbl@thetextdir\z@
838 \bbl@textdir@i\beginL\endL
839 \else
840 \chardef\bbl@thetextdir\@ne
841 \bbl@textdir@i\beginR\endR
842 \fi}
843 \def\bbl@textdir@i#1#2{%
844 \ifhmode
845 \ifnum\currentgrouplevel>\z@
846 \ifnum\currentgrouplevel=\bbl@dirlevel
847 \bbl@error{Multiple bidi settings inside a group}%
848 {I'll insert a new group, but expect wrong results.}%
849 \bgroup\aftergroup#2\aftergroup\egroup
850 \else
851 \ifcase\currentgrouptype\or % 0 bottom
852 \aftergroup#2% 1 simple {}
853 \or
854 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
855 \or
856 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
857 \or\or\or % vbox vtop align
858 \or
859 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
860 \or\or\or\or\or\or % output math disc insert vcent mathchoice
861 \or
862 \aftergroup#2% 14 \begingroup
863 \else
864 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
865 \fi
866 \fi
867 \bbl@dirlevel\currentgrouplevel
868 \fi
869 #1%
870 \fi}
871 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
872 \let\bbl@bodydir\@gobble
873 \let\bbl@pagedir\@gobble
874 \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).

```

875 \def\bbl@xebidipar{%

```

```

876 \let\bbl@xebidipar\relax
877 \TeXeTstate\@ne
878 \def\bbl@xeverypar{%
879 \ifcase\bbl@thepardir
880 \ifcase\bbl@thetextdir\else\beginR\fi
881 \else
882 {\setbox\z@\lastbox\beginR\box\z@}%
883 \fi}%
884 \let\bbl@severypar\everypar
885 \newtoks\everypar
886 \everypar=\bbl@severypar
887 \bbl@severypar{\bbl@xeverypar\the\everypar}}
888 \ifnum\bbl@bidimode>200
889 \let\bbl@textdir\i@gobbletwo
890 \let\bbl@xebidipar\@empty
891 \AddBabelHook{bidi}{foreign}{%
892 \def\bbl@tempa{\def\BabelText####1}%
893 \ifcase\bbl@thetextdir
894 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
895 \else
896 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
897 \fi}
898 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
899 \fi
900 \fi

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

901 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
902 \AtBeginDocument{%
903 \ifx\pdfstringdefDisableCommands\@undefined\else
904 \ifx\pdfstringdefDisableCommands\relax\else
905 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
906 \fi
907 \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`.

```

908 \bbl@trace{Local Language Configuration}
909 \ifx\loadlocalcfg\@undefined
910 \@ifpackagewith{babel}{noconfigs}%
911 {\let\loadlocalcfg\@gobble}%
912 {\def\loadlocalcfg#1{%
913 \InputIfFileExists{#1.cfg}%
914 {\typeout{*****^J%
915 * Local config file #1.cfg used^^J%
916 *}}}%
917 \@empty}}
918 \fi

```

7.11 Language options

Languages are loaded when processing the corresponding option *except* if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs

the ldf file and does some additional checks (\input works, too, but possible errors are not caught).

```

919 \bbl@trace{Language options}
920 \let\bbl@afterlang\relax
921 \let\BabelModifiers\relax
922 \let\bbl@loaded@empty
923 \def\bbl@load@language#1{%
924   \InputIfFileExists{#1.ldf}%
925   {\edef\bbl@loaded{\CurrentOption
926     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
927     \expandafter\let\expandafter\bbl@afterlang
928       \csname\CurrentOption.ldf-h@k\endcsname
929     \expandafter\let\expandafter\BabelModifiers
930       \csname bbl@mod@\CurrentOption\endcsname}%
931   {\bbl@error{%
932     Unknown option '\CurrentOption'. Either you misspelled it\\%
933     or the language definition file \CurrentOption.ldf was not found}}{%
934     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
935     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
936     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.

```

937 \def\bbl@try@load@lang#1#2#3{%
938   \IfFileExists{\CurrentOption.ldf}%
939   {\bbl@load@language{\CurrentOption}}}%
940   {#1\bbl@load@language{#2}#3}}
941 %
942 \DeclareOption{hebrew}{%
943   \input{rlbabel.def}}%
944   \bbl@load@language{hebrew}}
945 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
946 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
947 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
948 \DeclareOption{polutonikogreek}{%
949   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
950 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
951 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
952 \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.

```

953 \ifx\bbl@opt@config@nnil
954   \@ifpackagewith{babel}{noconfigs}{}%
955   {\InputIfFileExists{bblopts.cfg}%
956     {\typeout{*****^J%
957       * Local config file bblopts.cfg used^^J%
958       *}}}%
959   }{}%
960 \else
961   \InputIfFileExists{\bbl@opt@config.cfg}%
962   {\typeout{*****^J%
963     * Local config file \bbl@opt@config.cfg used^^J%
964     *}}}%
965   {\bbl@error{%
966     Local config file '\bbl@opt@config.cfg' not found}}{%
967     Perhaps you misspelled it.}}%

```

968 \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 existing languages (note this list also contains the language given with `main` as the last element). If not declared above, the names of the option and the file are the same. There are two steps – first process option names and collect the result, which then do the actual declarations.

To allow multiple overlapping replacements, commas in `\bbl@language@opts` are doubled.

```

969 \let\bbl@elt\relax
970 \let\bbl@tempe\@empty
971 \bbl@foreach\@classoptionslist{%
972   \bbl@xin@{,#1,$}\bbl@language@opts$}% Match last
973   \ifin@else
974     \bbl@xin@{,#1,}\bbl@language@opts$}% Match non-last
975   \ifin@
976     \bbl@replace\bbl@language@opts{,#1,},{,}%
977     \edef\bbl@tempe{\bbl@tempe\bbl@elt{0}{#1}}%
978   \else
979     \IfFileExists{#1.ldf}%
980     {\edef\bbl@tempe{\bbl@tempe\bbl@elt{1}{#1}}}%
981     {\ifnum\bbl@iniflag>\z@ % Optimization
982       \IfFileExists{babel-#1.tex}%
983       {\edef\bbl@tempe{\bbl@tempe\bbl@elt{2}{#1}}}%
984       {}}%
985     \fi}%
986   \fi
987 \fi}
988 %
989 \let\bbl@savemain\@empty
990 \bbl@foreach\bbl@language@opts{%
991   \edef\bbl@tempe{\bbl@tempe\bbl@elt{0}{#1}}%
992   \def\bbl@elt#1#2#3{%
993     \ifx#3\relax % if last
994       \bbl@ifunset{ds@#2}{}%
995       {\toks@\expandafter\expandafter\expandafter
996        {\csname ds@#2\endcsname}%
997        \bbl@exp{\def\\bbl@savemain{\\DeclareOption{#2}{\the\toks@}}}%
998        \bbl@add\bbl@savemain{\bbl@elt{#1}{#2}}% Save main
999        \DeclareOption{#2}{}%
1000      \else
1001        \ifnum\bbl@iniflag<\tw@ % other as ldf
1002          \ifnum#1<\tw@ % Class: if ldf exists 1. Package: always 0
1003            \bbl@ifunset{ds@#2}%
1004            {\DeclareOption{#2}{\bbl@load@language{#2}}}%
1005            {}%
1006          \fi
1007        \else % other as ini
1008          \ifodd#1\relax\else % Class: if ini exists 2. Package: always 0
1009            \DeclareOption{#2}{%
1010              \bbl@ldfinit
1011              \babelprovide[import]{#2}%
1012              \bbl@afterldf{}}% <- 'main' implicit here
1013            \fi
1014          \fi
1015        \fi
1016      #3}
1017 \bbl@tempe\relax % \relax catches last

```

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.
 If a main language has been set, store it for the third pass. 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 \TeX processes before):

```
1018 \def\AfterBabelLanguage#1{%
1019   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1020 \DeclareOption*{}
1021 \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.

```
1022 \bbl@trace{Option 'main'}
1023 \ifx\bbl@opt@main\@nnil   %% TODO. Revise
1024   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1025   \let\bbl@tempc\@empty
1026   \bbl@for\bbl@tempb\bbl@tempa{%
1027     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1028     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1029   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1030   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1031   \ifx\bbl@tempb\bbl@tempc\else
1032     \bbl@warning{%
1033       Last declared language option is '\bbl@tempc',\%
1034       but the last processed one was '\bbl@tempb'.\%
1035       The main language can't be set as both a global\%
1036       and a package option. Use 'main=\bbl@tempc' as\%
1037       option. Reported}%
1038   \fi
1039 \fi
1040 \def\bbl@elt#1#2{% main
1041   \ifodd\bbl@iniflag % as ini = 1,3
1042     \ifodd#1\relax\else % Class: if ini exists 2. Package: always 0
1043       \DeclareOption{#2}{%
1044         \bbl@ldfinit
1045         \babelprovide[\bbl@opt@provide,import]{#2}%
1046         \bbl@afterldf{}}%
1047     \fi
1048   \else % as ldf = 0,2
1049     \ifnum#1<\tw@ % Class: if ldf exists 1. Package: always 0
1050       \bbl@ifunset{ds@#2}%
1051       {\DeclareOption{#2}{\bbl@load@language{#2}}}%
1052       {}%
1053     \fi
1054   \fi}
1055 \bbl@savemain
1056 \DeclareOption*{}
1057 \ProcessOptions*
1058 \def\AfterBabelLanguage{%
1059   \bbl@error
1060   {Too late for \string\AfterBabelLanguage}%
1061   {Languages have been loaded, so I can do nothing}}
```

In order to catch the case where the user forgot to specify a language we check whether \bbl@main@language, has become defined. If not, no language has been loaded and an error message is displayed.


```

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

```

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

The kernel of the babel system is currently stored in babel.def. The file babel.def contains most of the code. The file hyphen.cfg is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns.

Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

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

8.1 Tools

```

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

```

The file babel.def expects some definitions made in the $\LaTeX 2_{\epsilon}$ style file. So, in 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).

```

1074 \ifx\AtBeginDocument\@undefined % TODO. change test.
1075   <<Emulate LaTeX>>
1076   \def\language{english}%
1077   \let\bbl@opt@shorthands\@nnil
1078   \def\bbl@ifshorthand#1#2#3#2}%
1079   \let\bbl@language@opts\@empty
1080   \ifx\babeloptionstrings\@undefined
1081     \let\bbl@opt@strings\@nnil
1082   \else
1083     \let\bbl@opt@strings\babeloptionstrings
1084   \fi
1085   \def\BabelStringsDefault{generic}
1086   \def\bbl@tempa{normal}
1087   \ifx\babeloptionmath\bbl@tempa
1088     \def\bbl@mathnormal{\noexpand\textormath}
1089   \fi
1090   \def\AfterBabelLanguage#1#2{}
1091   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1092   \let\bbl@afterlang\relax
1093   \def\bbl@opt@safe{BR}
1094   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1095   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1096   \expandafter\newif\csname ifbbl@single\endcsname
1097   \chardef\bbl@bidimode\z@

```

1098 \fi

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.

1099 <<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.

```
1100 \def\bbl@version{<<version>>}%
1101 \def\bbl@date{<<date>>}%
1102 \def\adddialect#1#2{%
1103   \global\chardef#1#2\relax
1104   \bbl@usehooks{adddialect}{#1}{#2}}%
1105   \begingroup
1106     \count@#1\relax
1107     \def\bbl@elt##1##2###3###4{%
1108       \ifnum\count@=##2\relax
1109         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
1110         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
1111                   set to \expandafter\string\csname l@##1\endcsname\\%
1112                   (\string\language\the\count@). Reported}%
1113         \def\bbl@elt####1####2####3####4}%
1114       \fi}%
1115   \bbl@cs{languages}%
1116   \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 an attempt 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.

```
1117 \def\bbl@fixname#1{%
1118   \begingroup
1119     \def\bbl@tempe{l@}%
1120     \edef\bbl@tempd{\noexpand\ifundefined{\noexpand\bbl@tempe#1}}%
1121     \bbl@tempd
1122     {\lowercase\expandafter{\bbl@tempd}%
1123      {\uppercase\expandafter{\bbl@tempd}%
1124       \@empty
1125       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1126        \uppercase\expandafter{\bbl@tempd}}}%
1127      {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1128       \lowercase\expandafter{\bbl@tempd}}}%
1129     \@empty
1130     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1131   \bbl@tempd
1132   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1133 \def\bbl@iflanguage#1{%
1134   \ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}
```

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

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

```

1135 \def\bbl@bcpcase#1#2#3#4\@#5{%
1136   \ifx\@empty#3%
1137     \uppercase{\def#5{#1#2}}%
1138   \else
1139     \uppercase{\def#5{#1}}%
1140     \lowercase{\edef#5{#5#2#3#4}}%
1141   \fi}
1142 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1143   \let\bbl@bcp\relax
1144   \lowercase{\def\bbl@tempa{#1}}%
1145   \ifx\@empty#2%
1146     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1147   \else\ifx\@empty#3%
1148     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1149     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1150       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1151     }%
1152     \ifx\bbl@bcp\relax
1153       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1154     \fi
1155   \else
1156     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1157     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1158     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1159       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1160     }%
1161     \ifx\bbl@bcp\relax
1162       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1163       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1164     }%
1165     \fi
1166     \ifx\bbl@bcp\relax
1167       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1168       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1169     }%
1170     \fi
1171     \ifx\bbl@bcp\relax
1172       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1173     \fi
1174   \fi\fi}
1175 \let\bbl@initoload\relax
1176 \def\bbl@provide@locale{%
1177   \ifx\babelprovide\@undefined
1178     \bbl@error{For a language to be defined on the fly 'base'\\%
1179               is not enough, and the whole package must be\\%
1180               loaded. Either delete the 'base' option or\\%
1181               request the languages explicitly}%
1182     {See the manual for further details.}%
1183   \fi
1184 % TODO. Option to search if loaded, with \LocaleForEach
1185 \let\bbl@auxname\language\name % Still necessary. TODO
1186 \bbl@ifunset{bbl@bcp@map@\language\name}{}% Move uplevel??
1187 {\edef\language\name{\@nameuse{bbl@bcp@map@\language\name}}}%
1188 \ifbbl@bcplallowed
1189   \expandafter\ifx\csname date\language\name\endcsname\relax

```

```

1190 \expandafter
1191 \bbl@bcplookup\language- \@empty- \@empty- \@empty\@@
1192 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1193 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1194 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1195 \expandafter\ifx\csname date\language\endcsname\relax
1196 \let\bbl@initoload\bbl@bcp
1197 \bbl@exp{\ \ \babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1198 \let\bbl@initoload\relax
1199 \fi
1200 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1201 \fi
1202 \fi
1203 \fi
1204 \expandafter\ifx\csname date\language\endcsname\relax
1205 \IfFileExists{babel-\language.tex}%
1206 {\bbl@exp{\ \ \babelprovide[\bbl@autoload@options]{\language}}}%
1207 {}%
1208 \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.

```

1209 \def\iflanguage#1{%
1210 \bbl@iflanguage{#1}%
1211 \ifnum\csname l@#1\endcsname=\language
1212 \expandafter\@firstoftwo
1213 \else
1214 \expandafter\@secondoftwo
1215 \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.

```

1216 \let\bbl@select@type\z@
1217 \edef\selectlanguage{%
1218 \noexpand\protect
1219 \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`.

```

1220 \ifx\@undefined\protect\let\protect\relax\fi

```

The following definition is preserved for backwards compatibility (eg, arabi, koma). It is related to a trick for 2.09, now discarded.

```

1221 \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.

```
1222 \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`

```
1223 \def\bbl@push@language{%
1224   \ifx\language\@undefined\else
1225     \ifx\currentgrouplevel\@undefined
1226       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1227     \else
1228       \ifnum\currentgrouplevel=\z@
1229         \xdef\bbl@language@stack{\language+}%
1230       \else
1231         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1232       \fi
1233     \fi
1234   \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 in `\bbl@language@stack`.

```
1235 \def\bbl@pop@lang#1+#2\@{%
1236   \edef\language{#1}%
1237   \xdef\bbl@language@stack{#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).

```
1238 \let\bbl@ifrestoring\@secondoftwo
1239 \def\bbl@pop@language{%
1240   \expandafter\bbl@pop@lang\bbl@language@stack\@
1241   \let\bbl@ifrestoring\@firstoftwo
1242   \expandafter\bbl@set@language\expandafter{\language}%
1243   \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).

```
1244 \chardef\localeid\z@
1245 \def\bbl@id@last{0} % No real need for a new counter
1246 \def\bbl@id@assign{%
1247   \bbl@ifunset\bbl@id@\language}%
1248   {\count@\bbl@id@last\relax
1249     \advance\count@\@ne
1250     \bbl@csarg\chardef{id@\language}\count@
1251     \edef\bbl@id@last{\the\count@}%
1252     \ifcase\bbl@engine\or
```

```

1253     \directlua{
1254         Babel = Babel or {}
1255         Babel.locale_props = Babel.locale_props or {}
1256         Babel.locale_props[\bbl@id@last] = {}
1257         Babel.locale_props[\bbl@id@last].name = '\language'
1258     }%
1259     \fi}%
1260 }%
1261 \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

1262 \expandafter\def\csname selectlanguage \endcsname#1{%
1263     \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
1264     \bbl@push@language
1265     \aftergroup\bbl@pop@language
1266     \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` or `\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.

`\bbl@savelastskip` is used to deal with skips before the write `whatsit` (as suggested by U Fischer). Adapted from `hyperref`, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in `luatex`, is to avoid the `\write` altogether when not needed).

```

1267 \def\BabelContentsFiles{toc,lof,lot}
1268 \def\bbl@set@language#1{% from selectlanguage, pop@
1269 % The old buggy way. Preserved for compatibility.
1270 \edef\language{%
1271     \ifnum\escapechar=\expandafter`\string#1\@empty
1272     \else\string#1\@empty\fi}%
1273 \ifcat\relax\noexpand#1%
1274     \expandafter\ifx\csname date\language\endcsname\relax
1275         \edef\language{#1}%
1276         \let\localename\language
1277     \else
1278         \bbl@info{Using '\string\language' instead of 'language' is\\
1279             deprecated. If what you want is to use a\\
1280             macro containing the actual locale, make\\
1281             sure it does not match any language.\\
1282             Reported}%
1283         \ifx\scantokens\@undefined
1284             \def\localename{??}%
1285         \else
1286             \scantokens\expandafter{\expandafter
1287                 \def\expandafter\localename\expandafter{\language}}%
1288         \fi
1289     \fi
1290 \else
1291     \def\localename{#1}% This one has the correct catcodes
1292 \fi
1293 \select@language{\language}%
1294 % write to auxs
1295 \expandafter\ifx\csname date\language\endcsname\relax\else
1296     \if@filesw

```

```

1297 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1298 \bbl@savelastskip
1299 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1300 \bbl@restorelastskip
1301 \fi
1302 \bbl@usehooks{write}{}}%
1303 \fi
1304 \fi}
1305 %
1306 \let\bbl@restorelastskip\relax
1307 \def\bbl@savelastskip{%
1308 \let\bbl@restorelastskip\relax
1309 \ifvmode
1310 \ifdim\lastskip=\z@
1311 \let\bbl@restorelastskip\nobreak
1312 \else
1313 \bbl@exp{%
1314 \def\\bbl@restorelastskip{%
1315 \skip@=\the\lastskip
1316 \\nobreak \vskip-\skip@ \vskip\skip@}}%
1317 \fi
1318 \fi}
1319 %
1320 \newif\ifbbl@bcpallowed
1321 \bbl@bcpallowedfalse
1322 \def\select@language#1{% from set@, babel@aux
1323 % set hmap
1324 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1325 % set name
1326 \edef\language#1}%
1327 \bbl@fixname\language
1328 % TODO. name@map must be here?
1329 \bbl@provide@locale
1330 \bbl@iflanguage\language{%
1331 \expandafter\ifx\csname date\language\endcsname\relax
1332 \bbl@error
1333 {Unknown language '\language'. Either you have\\%
1334 misspelled its name, it has not been installed,\\%
1335 or you requested it in a previous run. Fix its name,\\%
1336 install it or just rerun the file, respectively. In\\%
1337 some cases, you may need to remove the aux file}%
1338 {You may proceed, but expect wrong results}%
1339 \else
1340 % set type
1341 \let\bbl@select@type\z@
1342 \expandafter\bbl@switch\expandafter{\language}%
1343 \fi}}
1344 \def\babel@aux#1#2{%
1345 \select@language{#1}%
1346 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
1347 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
1348 \def\babel@toc#1#2{%
1349 \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.

```

1350 \newif\ifbbl@usedategroup
1351 \def\bbl@switch#1{% from select@, foreign@
1352 % make sure there is info for the language if so requested
1353 \bbl@ensureinfo{#1}%
1354 % restore
1355 \originalTeX
1356 \expandafter\def\expandafter\originalTeX\expandafter{%
1357 \csname noextras#1\endcsname
1358 \let\originalTeX\@empty
1359 \babel@beginsave}%
1360 \bbl@usehooks{afterreset}{}%
1361 \languageshorthands{none}%
1362 % set the locale id
1363 \bbl@id@assign
1364 % switch captions, date
1365 % No text is supposed to be added here, so we remove any
1366 % spurious spaces.
1367 \bbl@bsphack
1368 \ifcase\bbl@select@type
1369 \csname captions#1\endcsname\relax
1370 \csname date#1\endcsname\relax
1371 \else
1372 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1373 \ifin@
1374 \csname captions#1\endcsname\relax
1375 \fi
1376 \bbl@xin@{,date,}{,\bbl@select@opts,}%
1377 \ifin@ % if \foreign... within \<lang>date
1378 \csname date#1\endcsname\relax
1379 \fi
1380 \fi
1381 \bbl@esphack
1382 % switch extras
1383 \bbl@usehooks{beforeextras}{}%
1384 \csname extras#1\endcsname\relax
1385 \bbl@usehooks{afterextras}{}%
1386 % > babel-ensure
1387 % > babel-sh-<short>
1388 % > babel-bidi
1389 % > babel-fontspec
1390 % hyphenation - case mapping
1391 \ifcase\bbl@opt@hyphenmap\or
1392 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1393 \ifnum\bbl@hymapsel>4\else
1394 \csname\language @bbl@hyphenmap\endcsname
1395 \fi
1396 \chardef\bbl@opt@hyphenmap\z@
1397 \else
1398 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1399 \csname\language @bbl@hyphenmap\endcsname
1400 \fi

```



```

1401 \fi
1402 \let\bbl@hymapsel\@cclv
1403 % hyphenation - select rules
1404 \ifnum\csname l@\language\endcsname=\l@unhyphenated
1405   \edef\bbl@tempa{u}%
1406 \else
1407   \edef\bbl@tempa{\bbl@cl{lnbrk}}%
1408 \fi
1409 % linebreaking - handle u, e, k (v in the future)
1410 \bbl@xin@{/u}{/\bbl@tempa}%
1411 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
1412 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
1413 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
1414 \ifin@
1415   % unhyphenated/kashida/elongated = allow stretching
1416   \language\l@unhyphenated
1417   \babel@savevariable\emergencystretch
1418   \emergencystretch\maxdimen
1419   \babel@savevariable\hbadness
1420   \hbadness\@M
1421 \else
1422   % other = select patterns
1423   \bbl@patterns{#1}%
1424 \fi
1425 % hyphenation - mins
1426 \babel@savevariable\lefthyphenmin
1427 \babel@savevariable\righthyphenmin
1428 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1429   \set@hyphenmins\tw@\thr@@\relax
1430 \else
1431   \expandafter\expandafter\expandafter\set@hyphenmins
1432     \csname #1hyphenmins\endcsname\relax
1433 \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.

```

1434 \long\def\otherlanguage#1{%
1435   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1436   \csname selectlanguage \endcsname{#1}%
1437   \ignorespaces}

```

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

```

1438 \long\def\endotherlanguage{%
1439   \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`.

```

1440 \expandafter\def\csname otherlanguage*\endcsname{%
1441   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1442 \def\bbl@otherlanguage@s[#1]#2{%
1443   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1444   \def\bbl@select@opts{#1}%
1445   \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”.

```
1446 \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.

```
1447 \providecommand\bbl@beforeforeign{}
1448 \edef\foreignlanguage{%
1449   \noexpand\protect
1450   \expandafter\noexpand\csname foreignlanguage \endcsname}
1451 \expandafter\def\csname foreignlanguage \endcsname{%
1452   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1453 \providecommand\bbl@foreign@x[3][]{%
1454   \begingroup
1455     \def\bbl@select@opts{#1}%
1456     \let\BabelText\@firstofone
1457     \bbl@beforeforeign
1458     \foreign@language{#2}%
1459     \bbl@usehooks{foreign}{}%
1460     \BabelText{#3}% Now in horizontal mode!
1461   \endgroup}
1462 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
1463   \begingroup
1464     {\par}%
1465     \let\bbl@select@opts\@empty
1466     \let\BabelText\@firstofone
1467     \foreign@language{#1}%
1468     \bbl@usehooks{foreign*}{}%
1469     \bbl@dirparastext
1470     \BabelText{#2}% Still in vertical mode!
1471     {\par}%
1472   \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`.

```
1473 \def\foreign@language#1{%
1474   % set name
1475   \edef\language{#1}%
```

```

1476 \ifbbl@usedategroup
1477   \bbl@add\bbl@select@opts{,date,}%
1478   \bbl@usedategroupfalse
1479 \fi
1480 \bbl@fixname\language
1481 % TODO. name@map here?
1482 \bbl@provide@locale
1483 \bbl@iflanguage\language{
1484   \expandafter\ifx\csname date\language\endcsname\relax
1485     \bbl@warning % TODO - why a warning, not an error?
1486     {Unknown language '#1'. Either you have\\%
1487      misspelled its name, it has not been installed,\\%
1488      or you requested it in a previous run. Fix its name,\\%
1489      install it or just rerun the file, respectively. In\\%
1490      some cases, you may need to remove the aux file.\\%
1491      I'll proceed, but expect wrong results.\\%
1492      Reported}%
1493   \fi
1494   % set type
1495   \let\bbl@select@type\@ne
1496   \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.

```

1497 \let\bbl@hyphlist\@empty
1498 \let\bbl@hyphenation@\relax
1499 \let\bbl@pttnlist\@empty
1500 \let\bbl@patterns@\relax
1501 \let\bbl@hymapsel=\@cclv
1502 \def\bbl@patterns#1{%
1503   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1504     \csname l@#1\endcsname
1505     \edef\bbl@tempa{#1}%
1506   \else
1507     \csname l@#1:\f@encoding\endcsname
1508     \edef\bbl@tempa{#1:\f@encoding}%
1509   \fi
1510   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1511 % > luatex
1512 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1513   \begingroup
1514     \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1515     \ifin@ \else
1516       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1517       \hyphenation{%
1518         \bbl@hyphenation@
1519         \@ifundefined{bbl@hyphenation@#1}%
1520         \@empty
1521         {\space\csname bbl@hyphenation@#1\endcsname}}%
1522       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1523     \fi
1524   \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 `otherlanguage*`.

```

1525 \def\hyphenrules#1{%
1526   \edef\bbl@tempf{#1}%
1527   \bbl@fixname\bbl@tempf
1528   \bbl@iflanguage\bbl@tempf{%
1529     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1530     \ifx\languageshorthands\undefined\else
1531       \languageshorthands{none}%
1532     \fi
1533     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1534       \set@hyphenmins\tw@\thr@@\relax
1535     \else
1536       \expandafter\expandafter\expandafter\set@hyphenmins
1537       \csname\bbl@tempf hyphenmins\endcsname\relax
1538     \fi}}
1539 \let\endhyphenrules\@empty

```

\providehyphenmins The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\(lang)hyphenmins` is already defined this command has no effect.

```

1540 \def\providehyphenmins#1#2{%
1541   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1542     \@namedef{#1hyphenmins}{#2}%
1543   \fi}

```

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

```

1544 \def\set@hyphenmins#1#2{%
1545   \lefthyphenmin#1\relax
1546   \righthyphenmin#2\relax}

```

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

```

1547 \ifx\ProvidesFile\undefined
1548   \def\ProvidesLanguage#1[#2 #3 #4]{%
1549     \wlog{Language: #1 #4 #3 <#2>}%
1550   }
1551 \else
1552   \def\ProvidesLanguage#1{%
1553     \begingroup
1554     \catcode`\ 10 %
1555     \@makeother\/%
1556     \ifnextchar[%]
1557       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1558   \def\@provideslanguage#1[#2]{%
1559     \wlog{Language: #1 #2}%
1560     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1561   \endgroup}
1562 \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`.

```

1563 \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.

```
1564 \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’:

```
1565 \providecommand\setlocale{%
1566   \bbl@error
1567   {Not yet available}%
1568   {Find an armchair, sit down and wait}}
1569 \let\uselocale\setlocale
1570 \let\locale\setlocale
1571 \let\selectlocale\setlocale
1572 \let\localename\setlocale
1573 \let\textlocale\setlocale
1574 \let\textlanguage\setlocale
1575 \let\language\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.

```
1576 \edef\bbl@nulllanguage{\string\language=0}
1577 \ifx\PackageError\@undefined % TODO. Move to Plain
1578   \def\bbl@error#1#2{%
1579     \begingroup
1580       \newlinechar=`^^J
1581       \def\{^^J(babel) }%
1582       \errhelp{#2}\errmessage{\{#1}%
1583     \endgroup}
1584   \def\bbl@warning#1{%
1585     \begingroup
1586       \newlinechar=`^^J
1587       \def\{^^J(babel) }%
1588       \message{\{#1}%
1589     \endgroup}
1590   \let\bbl@infowarn\bbl@warning
1591   \def\bbl@info#1{%
1592     \begingroup
1593       \newlinechar=`^^J
1594       \def\{^^J}%
1595       \wlog{#1}%
1596     \endgroup}
1597 \fi
1598 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1599 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1600   \global\@namedef{#2}{\textbf{?#1?}}%
1601   \@nameuse{#2}%
1602   \edef\bbl@tempa{#1}%
1603   \bbl@sreplace\bbl@tempa{name}{}}%
```

```

1604 \bbl@warning{% TODO.
1605 \@backslashchar#1 not set for '\language'. Please,\\%
1606 define it after the language has been loaded\\%
1607 (typically in the preamble) with:\\%
1608 \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
1609 Reported}}
1610 \def\bbl@tentative{\protect\bbl@tentative@i}
1611 \def\bbl@tentative@i#1{%
1612 \bbl@warning{%
1613 Some functions for '#1' are tentative.\\%
1614 They might not work as expected and their behavior\\%
1615 could change in the future.\\%
1616 Reported}}
1617 \def\@nolanerr#1{%
1618 \bbl@error
1619 {You haven't defined the language '#1' yet.\\%
1620 Perhaps you misspelled it or your installation\\%
1621 is not complete}%
1622 {Your command will be ignored, type <return> to proceed}}
1623 \def\@nopatterns#1{%
1624 \bbl@warning
1625 {No hyphenation patterns were preloaded for\\%
1626 the language '#1' into the format.\\%
1627 Please, configure your TeX system to add them and\\%
1628 rebuild the format. Now I will use the patterns\\%
1629 preloaded for \bbl@nulllanguage\space instead}}
1630 \let\bbl@usehooks\@gobbletwo
1631 \ifx\bbl@onlyswitch\@empty\endinput\fi
1632 % Here ended switch.def

Here ended switch.def.

1633 \ifx\directlua\@undefined\else
1634 \ifx\bbl@luapatterns\@undefined
1635 \input luababel.def
1636 \fi
1637 \fi
1638 <<Basic macros>>
1639 \bbl@trace{Compatibility with language.def}
1640 \ifx\bbl@languages\@undefined
1641 \ifx\directlua\@undefined
1642 \openin1 = language.def % TODO. Remove hardcoded number
1643 \ifeof1
1644 \closein1
1645 \message{I couldn't find the file language.def}
1646 \else
1647 \closein1
1648 \begingroup
1649 \def\addlanguage#1#2#3#4#5{%
1650 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1651 \global\expandafter\let\csname l@#1\endcsname\expandafter\endcsname
1652 \csname lang@#1\endcsname
1653 \fi}%
1654 \def\uselanguage#1{}%
1655 \input language.def
1656 \endgroup
1657 \fi
1658 \fi
1659 \chardef\l@english\z@
1660 \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.

```

1661 \def\addto#1#2{%
1662   \ifx#1\@undefined
1663     \def#1{#2}%
1664   \else
1665     \ifx#1\relax
1666       \def#1{#2}%
1667     \else
1668       {\toks@\expandafter{#1#2}%
1669        \xdef#1{\the\toks@}}%
1670   \fi
1671 \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`?

```

1672 \def\bbl@withactive#1#2{%
1673   \begingroup
1674   \lccode`~=#2\relax
1675   \lowercase{\endgroup#1~}}

```

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

```

1676 \def\bbl@redefine#1{%
1677   \edef\bbl@tempa{\bbl@stripslash#1}%
1678   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1679   \expandafter\def\csname\bbl@tempa\endcsname{
1680 \@onlypreamble\bbl@redefine

```

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

```

1681 \def\bbl@redefine@long#1{%
1682   \edef\bbl@tempa{\bbl@stripslash#1}%
1683   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1684   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1685 \@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_`.

```

1686 \def\bbl@redefineroobust#1{%
1687   \edef\bbl@tempa{\bbl@stripslash#1}%
1688   \bbl@ifunset{\bbl@tempa\space}%
1689   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1690    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1691   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1692   \@namedef{\bbl@tempa\space}%
1693 \@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.

```

1694 \bbl@trace{Hooks}
1695 \newcommand\AddBabelHook[3][\%
1696   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{\%
1697   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}\%
1698   \expandafter\bbl@tempa\bbl@evargs,##3=\@empty
1699   \bbl@ifunset{bbl@ev@#2@#3@#1}\%
1700     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}\%
1701     {\bbl@csarg\let{ev@#2@#3@#1}\relax}\%
1702   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1703 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1704 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1705 \def\bbl@usehooks#1#2{\%
1706   \ifx\UseHook\@undefined\else\UseHook{babel/#1}\fi
1707   \def\bbl@elth##1{\%
1708     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}\%
1709     \bbl@cs{ev@#1@#2}}\%
1710   \ifx\language\@undefined\else % Test required for Plain (?)
1711     \ifx\UseHook\@undefined\else\UseHook{babel/#1/\language}\fi
1712     \def\bbl@elth##1{\%
1713       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}\%
1714       \bbl@cl{ev@#1@#2}}\%
1715   \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).

```

1716 \def\bbl@evargs{\% <- don't delete this comma
1717   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1718   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1719   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1720   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1721   beforestart=0,language=2}
1722 \ifx\NewHook\@undefined\else
1723   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1724   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1725 \fi

```

`\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.

```

1726 \bbl@trace{Defining babelensure}
1727 \newcommand\babelensure[2][\% TODO - revise test files
1728   \AddBabelHook{babel-ensure}{afterextras}\%
1729   \ifcase\bbl@select@type
1730     \bbl@cl{e}\%
1731   \fi}\%
1732 \begingroup
1733   \let\bbl@ens@include\@empty

```



```

1734 \let\bb1@ens@exclude\@empty
1735 \def\bb1@ens@fontenc{\relax}%
1736 \def\bb1@tempb##1{%
1737   \ifx\@empty##1\else\noexpand##1\expandafter\bb1@tempb\fi}%
1738 \edef\bb1@tempa{\bb1@tempb#1\@empty}%
1739 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ens@##1}{##2}}%
1740 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
1741 \def\bb1@tempc{\bb1@ensure}%
1742 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1743   \expandafter{\bb1@ens@include}}%
1744 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1745   \expandafter{\bb1@ens@exclude}}%
1746 \toks@\expandafter{\bb1@tempc}%
1747 \bb1@exp{%
1748 \endgroup
1749 \def\<bb1@e@#2>{\the\toks@{\bb1@ens@fontenc}}}%
1750 \def\bb1@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1751 \def\bb1@tempb##1{% elt for (excluding) \bb1@captionslist list
1752   \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1753     \edef##1{\noexpand\bb1@nocaption
1754       {\bb1@stripslash##1}{\language\bb1@stripslash##1}}%
1755   \fi
1756   \ifx##1\@empty\else
1757     \in@{##1}{#2}%
1758     \ifin@\else
1759       \bb1@ifunset{bb1@ensure@\language}%
1760       {\bb1@exp{%
1761         \\DeclareRobustCommand\<bb1@ensure@\language>[1]{%
1762           \\foreignlanguage{\language}%
1763             {\ifx\relax#3\else
1764               \\fontencoding{#3}\\selectfont
1765               \fi
1766               #####1}}}%
1767       {}}%
1768       \toks@\expandafter{##1}%
1769       \edef##1{%
1770         \bb1@csarg\noexpand{ensure@\language}%
1771         {\the\toks@}}%
1772       \fi
1773       \expandafter\bb1@tempb
1774     \fi}%
1775 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
1776 \def\bb1@tempa##1{% elt for include list
1777   \ifx##1\@empty\else
1778     \bb1@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1779     \ifin@\else
1780       \bb1@tempb##1\@empty
1781     \fi
1782     \expandafter\bb1@tempa
1783   \fi}%
1784 \bb1@tempa#1\@empty}
1785 \def\bb1@captionslist{%
1786 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1787 \contentsname\listfigurename\listtablename\indexname\figurename
1788 \tablename\partname\enc1name\ccname\headtoname\pagename\seename
1789 \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 `@`-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`.

```

1790 \bbl@trace{Macros for setting language files up}
1791 \def\bbl@ldfinit{%
1792   \let\bbl@screset\@empty
1793   \let\BabelStrings\bbl@opt@string
1794   \let\BabelOptions\@empty
1795   \let\BabelLanguages\relax
1796   \ifx\originalTeX\@undefined
1797     \let\originalTeX\@empty
1798   \else
1799     \originalTeX
1800   \fi}
1801 \def\LdfInit#1#2{%
1802   \chardef\atcatcode=\catcode`\@
1803   \catcode`\@=11\relax
1804   \chardef\eqcatcode=\catcode`\=
1805   \catcode`\==12\relax
1806   \expandafter\if\expandafter\@backslashchar
1807     \expandafter\@car\string#2\@nil
1808   \ifx#2\@undefined\else
1809     \ldf@quit{#1}%
1810   \fi
1811 \else
1812   \expandafter\ifx\csname#2\endcsname\relax\else
1813     \ldf@quit{#1}%
1814   \fi
1815 \fi
1816 \bbl@ldfinit}

```

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

```

1817 \def\ldf@quit#1{%
1818   \expandafter\main@language\expandafter{#1}%
1819   \catcode`\@=\atcatcode \let\atcatcode\relax
1820   \catcode`\==\eqcatcode \let\eqcatcode\relax
1821   \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.

```

1822 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1823   \bbl@afterlang
1824   \let\bbl@afterlang\relax
1825   \let\BabelModifiers\relax
1826   \let\bbl@screset\relax}%
1827 \def\ldf@finish#1{%
1828   \loadlocalcfg{#1}%
1829   \bbl@afterldf{#1}%
1830   \expandafter\main@language\expandafter{#1}%
1831   \catcode`\@=\atcatcode \let\atcatcode\relax
1832   \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 L^AT_EX.

```

1833 \@onlypreamble\LdfInit
1834 \@onlypreamble\ldf@quit
1835 \@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.

```

1836 \def\main@language#1{%
1837   \def\bbl@main@language{#1}%
1838   \let\language\name\bbl@main@language % TODO. Set localename
1839   \bbl@id@assign
1840   \bbl@patterns{\language\name}}

```

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.

```

1841 \def\bbl@beforestart{%
1842   \def\@nolanerr##1{%
1843     \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1844   \bbl@usehooks{beforestart}{}%
1845   \global\let\bbl@beforestart\relax}
1846 \AtBeginDocument{%
1847   {\@nameuse{bbl@beforestart}}% Group!
1848   \if@filesw
1849     \providecommand\babel@aux[2]{}%
1850     \immediate\write\@mainaux{%
1851       \string\providecommand\string\babel@aux[2]{}%
1852       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1853   \fi
1854   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1855   \ifbbl@single % must go after the line above.
1856     \renewcommand\selectlanguage[1]{}%
1857     \renewcommand\foreignlanguage[2]{#2}%
1858     \global\let\babel@aux\@gobbletwo % Also as flag
1859   \fi
1860   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1861 \def\select@language@x#1{%
1862   \ifcase\bbl@select@type
1863     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1864   \else

```

```

1865 \select@language{#1}%
1866 \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.

```

1867 \bbl@trace{Shorhands}
1868 \def\bbl@add@special#1{% 1:a macro like "\", "\?", etc.
1869 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1870 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1871 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1872 \begingroup
1873 \catcode`#1\active
1874 \nfss@catcodes
1875 \ifnum\catcode`#1=\active
1876 \endgroup
1877 \bbl@add\nfss@catcodes{\@makeother#1}%
1878 \else
1879 \endgroup
1880 \fi
1881 \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.

```

1882 \def\bbl@remove@special#1{%
1883 \begingroup
1884 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1885 \else\noexpand##1\noexpand##2\fi}%
1886 \def\do{\x\do}%
1887 \def\@makeother{\x\@makeother}%
1888 \edef\x{\endgroup
1889 \def\noexpand\dospecials{\dospecials}%
1890 \expandafter\ifx\curname @sanitize\endcurname\relax\else
1891 \def\noexpand\@sanitize{\@sanitize}%
1892 \fi}%
1893 \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).

```

1894 \def\bbl@active@def#1#2#3#4{%
1895   \@namedef{#3#1}{%
1896     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1897       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1898     \else
1899       \bbl@afterfi\csname#2@sh@#1\endcsname
1900     \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.

```

1901 \long\@namedef{#3@arg#1}##1{%
1902   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1903     \bbl@afterelse\csname#4#1\endcsname##1%
1904   \else
1905     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1906   \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.

```

1907 \def\initiate@active@char#1{%
1908   \bbl@ifunset{active@char\string#1}%
1909   {\bbl@withactive
1910     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1911   {}}

```

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 and preserving some degree of protection).

```

1912 \def\@initiate@active@char#1#2#3{%
1913   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1914   \ifx#1\@undefined
1915     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1916   \else
1917     \bbl@csarg\let{oridef@#2}#1%
1918     \bbl@csarg\edef{oridef@#2}{%
1919       \let\noexpand#1%
1920       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1921   \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).

```

1922 \ifx#1#3\relax
1923   \expandafter\let\csname normal@char#2\endcsname#3%
1924 \else
1925   \bbl@info{Making #2 an active character}%
1926   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1927   \@namedef{normal@char#2}{%
1928     \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1929   \else
1930     \@namedef{normal@char#2}{#3}%
1931   \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).

```

1932 \bbl@restoreactive{#2}%
1933 \AtBeginDocument{%
1934   \catcode`#2\active
1935   \if@filesw
1936     \immediate\write\@mainaux{\catcode`\string#2\active}%
1937   \fi}%
1938 \expandafter\bbl@add@special\csname#2\endcsname
1939 \catcode`#2\active
1940 \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⟩`).

```

1941 \let\bbl@tempa\@firstoftwo
1942 \if\string^#2%
1943   \def\bbl@tempa{\noexpand\textormath}%
1944 \else
1945   \ifx\bbl@mathnormal\@undefined\else
1946     \let\bbl@tempa\bbl@mathnormal
1947   \fi
1948 \fi
1949 \expandafter\edef\csname active@char#2\endcsname{%
1950   \bbl@tempa
1951     {\noexpand\if@safe@actives
1952       \noexpand\expandafter
1953         \expandafter\noexpand\csname normal@char#2\endcsname
1954       \noexpand\else
1955         \noexpand\expandafter
1956         \expandafter\noexpand\csname bbl@doactive#2\endcsname
1957       \noexpand\fi}%
1958   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1959 \bbl@csarg\edef{doactive#2}{%
1960   \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!).

```

1961 \bbl@csarg\edef{active@#2}{%
1962   \noexpand\active@prefix\noexpand#1%
1963   \expandafter\noexpand\csname active@char#2\endcsname}%
1964 \bbl@csarg\edef{normal@#2}{%
1965   \noexpand\active@prefix\noexpand#1%
1966   \expandafter\noexpand\csname normal@char#2\endcsname}%
1967 \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.

```

1968 \bbl@active@def#2\user@group{user@active}{language@active}%

```

```

1969 \bbl@active@def#2\language@group{language@active}{system@active}%
1970 \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.

```

1971 \expandafter\edef\csname\user@group @sh@#2@\endcsname
1972 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1973 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1974 {\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.

```

1975 \if\string'#2%
1976 \let\prim@s\bbl@prim@s
1977 \let\active@math@prime#1%
1978 \fi
1979 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

1980 <<{*More package options}>> ≡
1981 \DeclareOption{math=active}{}
1982 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1983 <</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* the end of the ldf.

```

1984 \@ifpackagewith{babel}{KeepShorthandsActive}%
1985 {\let\bbl@restoreactive\@gobble}%
1986 {\def\bbl@restoreactive#1{%
1987   \bbl@exp{%
1988     \\\AfterBabelLanguage\\\CurrentOption
1989     {\catcode`#1=\the\catcode`#1\relax}%
1990     \\\AtEndOfPackage
1991     {\catcode`#1=\the\catcode`#1\relax}}}%
1992 \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.

```

1993 \def\bbl@sh@select#1#2{%
1994   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1995     \bbl@afterelse\bbl@scndcs
1996   \else
1997     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1998   \fi}

```

`\active@prefix` The command \active@prefix which is used in the expansion of active characters has a function similar to \OT1-cmd in that it \protects the active character whenever \protect is *not* \@typeset@protect. The \@gobble is needed to remove a token such as \activechar: (when the

double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

1999 \begingroup
2000 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2001 {\gdef\active@prefix#1{%
2002   \ifx\protect\@typeset@protect
2003   \else
2004     \ifx\protect\@unexpandable@protect
2005       \noexpand#1%
2006     \else
2007       \protect#1%
2008     \fi
2009     \expandafter\@gobble
2010   \fi}}
2011 {\gdef\active@prefix#1{%
2012   \ifincsname
2013     \string#1%
2014     \expandafter\@gobble
2015   \else
2016     \ifx\protect\@typeset@protect
2017     \else
2018       \ifx\protect\@unexpandable@protect
2019         \noexpand#1%
2020       \else
2021         \protect#1%
2022       \fi
2023       \expandafter\expandafter\expandafter\@gobble
2024     \fi
2025   \fi}}
2026 \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⟩`.

```

2027 \newif\if@safe@actives
2028 \@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.

```

2029 \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`.

```

2030 \chardef\bbl@activated\z@
2031 \def\bbl@activate#1{%
2032   \chardef\bbl@activated\@ne
2033   \bbl@withactive{\expandafter\let\expandafter}#1%
2034   \csname bbl@active@\string#1\endcsname}
2035 \def\bbl@deactivate#1{%
2036   \chardef\bbl@activated\tw@
2037   \bbl@withactive{\expandafter\let\expandafter}#1%
2038   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2039 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2040 \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.

The auxiliary macro `\babel@texpdf` improves the interoperativity with `hyperref` and takes 4 arguments: (1) The \TeX code in text mode, (2) the string for `hyperref`, (3) the \TeX code in math mode, and (4), which is currently ignored, but it's meant for a string in math mode, like a minus sign instead of an hyphen (currently `hyperref` doesn't discriminate the mode). This macro may be used in `ldf` files.

```

2041 \def\babel@texpdf#1#2#3#4{%
2042   \ifx\texorpdfstring\undefined
2043     \textormath{#1}{#3}%
2044   \else
2045     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2046     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2047   \fi}
2048 %
2049 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2050 \def\@decl@short#1#2#3\@nil#4{%
2051   \def\bbl@tempa{#3}%
2052   \ifx\bbl@tempa\@empty
2053     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2054     \bbl@ifunset{#1@sh@\string#2@}{}%
2055     {\def\bbl@tempa{#4}%
2056      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2057      \else
2058        \bbl@info
2059          {Redefining #1 shorthand \string#2\\%
2060           in language \CurrentOption}%
2061      \fi}%
2062     \@namedef{#1@sh@\string#2@}{#4}%
2063   \else
2064     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2065     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2066     {\def\bbl@tempa{#4}%
2067      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2068      \else
2069        \bbl@info
2070          {Redefining #1 shorthand \string#2\string#3\\%
2071           in language \CurrentOption}%
2072      \fi}%
2073     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2074   \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.

```

2075 \def\textormath{%
2076   \ifmmode
2077     \expandafter\@secondoftwo
2078   \else
2079     \expandafter\@firstoftwo
2080   \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’.

```

2081 \def\user@group{user}
2082 \def\language@group{english} % TODO. I don't like defaults
2083 \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.

2084 \def\useshorthands{%
2085   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2086 \def\bbl@usesh@s#1{%
2087   \bbl@usesh@x
2088   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2089   {#1}}
2090 \def\bbl@usesh@x#1#2{%
2091   \bbl@ifshorthand{#2}%
2092   {\def\user@group{user}%
2093     \initiate@active@char{#2}%
2094     #1%
2095     \bbl@activate{#2}}%
2096   {\bbl@error
2097     {I can't declare a shorthand turned off (\string#2)}
2098     {Sorry, but you can't use shorthands which have been\\
2099     turned off in the package options}}}

\defineshorthand Currently we only support two groups of user level shorthands, named internally user and
user@<lang> (language-dependent user shorthands). By default, only the first one is taken into
account, but if the former is also used (in the optional argument of \defineshorthand) a new level is
inserted for it (user@generic, done by \bbl@set@user@generic); we make also sure {} and
\protect are taken into account in this new top level.

2100 \def\user@language@group{user@\language@group}
2101 \def\bbl@set@user@generic#1#2{%
2102   \bbl@ifunset{user@generic@active#1}%
2103   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2104     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2105     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2106       \expandafter\noexpand\csname normal@char#1\endcsname}%
2107     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2108       \expandafter\noexpand\csname user@active#1\endcsname}}%
2109   \@empty}
2110 \newcommand\defineshorthand[3][user]{%
2111   \edef\bbl@tempa{\zap@space#1 \@empty}%
2112   \bbl@for\bbl@tempb\bbl@tempa{%
2113     \if*\expandafter\@car\bbl@tempb\@nil
2114       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2115       \@expandtwoargs
2116       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2117     \fi
2118     \declare@shorthand{\bbl@tempb}{#2}{#3}}}

\languageshorthands A user level command to change the language from which shorthands are used. Unfortunately, babel
currently does not keep track of defined groups, and therefore there is no way to catch a possible
change in casing to fix it in the same way languages names are fixed. [TODO].

2119 \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".

2120 \def\aliasshorthand#1#2{%

```

```

2121 \bbl@ifshorthand{#2}%
2122 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2123   \ifx\document\@notprerr
2124     \notshorthand{#2}%
2125   \else
2126     \initiate@active@char{#2}%
2127     \expandafter\let\csname active@char\string#2\expandafter\endcsname
2128       \csname active@char\string#1\endcsname
2129     \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2130       \csname normal@char\string#1\endcsname
2131     \bbl@activate{#2}%
2132   \fi
2133 \fi}%
2134 {\bbl@error
2135   {Cannot declare a shorthand turned off (\string#2)}
2136   {Sorry, but you cannot use shorthands which have been\\%
2137     turned off in the package options}}}

```

\@notshorthand

```

2138 \def\@notshorthand#1{%
2139   \bbl@error{%
2140     The character '\string #1' should be made a shorthand character;\\%
2141     add the command \string\useshorthands\string{#1\string} to
2142     the preamble.\\%
2143     I will ignore your instruction}%
2144   {You may proceed, but expect unexpected results}}

```

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

```

2145 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2146 \DeclareRobustCommand*\shorthandoff{%
2147   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2148 \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.

```

2149 \def\bbl@switch@sh#1#2{%
2150   \ifx#2\@nnil\else
2151     \bbl@ifunset{\bbl@active@\string#2}%
2152     {\bbl@error
2153       {I can't switch '\string#2' on or off--not a shorthand}%
2154       {This character is not a shorthand. Maybe you made\\%
2155         a typing mistake? I will ignore your instruction.}}%
2156     {\ifcase#1%   off, on, off*
2157       \catcode`#212\relax
2158     \or
2159       \catcode`#2\active
2160       \bbl@ifunset{\bbl@shdef@\string#2}%
2161       {}%
2162       {\bbl@withactive{\expandafter\let\expandafter}#2%
2163         \csname bbl@shdef@\string#2\endcsname
2164         \bbl@csarg\let{shdef@\string#2}\relax}%
2165     \ifcase\bbl@activated\or

```

```

2166         \bbl@activate{#2}%
2167     \else
2168         \bbl@deactivate{#2}%
2169     \fi
2170 \or
2171     \bbl@ifunset{bbl@shdef@\string#2}%
2172     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
2173     {}%
2174     \csname bbl@oricat@\string#2\endcsname
2175     \csname bbl@oridef@\string#2\endcsname
2176 \fi}%
2177 \bbl@afterfi\bbl@switch@sh#1%
2178 \fi}

```

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

```

2179 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2180 \def\bbl@putsh#1{%
2181     \bbl@ifunset{bbl@active@\string#1}%
2182     {\bbl@putsh@i#1\@empty\@nnil}%
2183     {\csname bbl@active@\string#1\endcsname}}
2184 \def\bbl@putsh@i#1#2\@nnil{%
2185     \csname\language@group @sh@\string#1@%
2186     \ifx\@empty#2\else\string#2\fi\endcsname}
2187 \ifx\bbl@opt@shorthands\@nnil\else
2188     \let\bbl@s@initiate@active@char\initiate@active@char
2189     \def\initiate@active@char#1{%
2190         \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2191     \let\bbl@s@switch@sh\bbl@switch@sh
2192     \def\bbl@switch@sh#1#2{%
2193         \ifx#2\@nnil\else
2194             \bbl@afterfi
2195             \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2196         \fi}
2197     \let\bbl@s@activate\bbl@activate
2198     \def\bbl@activate#1{%
2199         \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2200     \let\bbl@s@deactivate\bbl@deactivate
2201     \def\bbl@deactivate#1{%
2202         \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2203 \fi

```

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

```

2204 \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.

```

2205 \def\bbl@prim@s{%
2206     \prime\futurelet\@let@token\bbl@pr@m@s}
2207 \def\bbl@if@primes#1#2{%
2208     \ifx#1\@let@token
2209         \expandafter\@firstoftwo
2210     \else\ifx#2\@let@token
2211         \bbl@afterelse\expandafter\@firstoftwo
2212     \else
2213         \bbl@afterfi\expandafter\@secondoftwo
2214     \fi\fi}

```

```

2215 \begingroup
2216 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2217 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\'
2218 \lowercase{%
2219 \gdef\bbl@pr@m@s{%
2220 \bbl@if@primes""%
2221 \pr@@@s
2222 {\bbl@if@primes*^\pr@@@t\egroup}}
2223 \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).

```

2224 \initiate@active@char{~}
2225 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2226 \bbl@activate{~}

```

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

```

2227 \expandafter\def\csname OT1dqpos\endcsname{127}
2228 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

2229 \ifx\f@encoding\undefined
2230 \def\f@encoding{OT1}
2231 \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.

```

2232 \bbl@trace{Language attributes}
2233 \newcommand\languageattribute[2]{%
2234 \def\bbl@tempc{#1}%
2235 \bbl@fixname\bbl@tempc
2236 \bbl@iflanguage\bbl@tempc{%
2237 \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.

```

2238 \ifx\bbl@known@attribs\undefined
2239 \in@false
2240 \else
2241 \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,%
2242 \fi
2243 \ifin@
2244 \bbl@warning{%
2245 You have more than once selected the attribute '##1'\%
2246 for language #1. Reported}%
2247 \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 T_EX-code.

```

2248      \bbl@exp{%
2249        \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2250      \edef\bbl@tempa{\bbl@tempc-##1}%
2251      \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2252      {\curname\bbl@tempc @attr##1\endcsname}%
2253      {\@attrerr{\bbl@tempc}{##1}}%
2254      \fi}}
2255 \@onlypreamble\languageattribute

```

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

```

2256 \newcommand*{\@attrerr}[2]{%
2257   \bbl@error
2258   {The attribute #2 is unknown for language #1.}%
2259   {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}.

```

2260 \def\bbl@declare@ttribute#1#2#3{%
2261   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2262   \ifin@
2263     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2264   \fi
2265   \bbl@add@list\bbl@attributes{#1-#2}%
2266   \expandafter\def\curname#1@attr#2\endcsname{#3}}

```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret T_EX code based on whether a certain attribute was set. This command should appear inside the argument to \AtBeginDocument because the attributes are set in the document preamble, *after* babel is loaded. The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```

2267 \def\bbl@ifattributeset#1#2#3#4{%
2268   \ifx\bbl@known@attribs\undefined
2269     \in@false
2270   \else
2271     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2272   \fi
2273   \ifin@
2274     \bbl@afterelse#3%
2275   \else
2276     \bbl@afterfi#4%
2277   \fi}

```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the T_EX-code to be executed when the attribute is known and the T_EX-code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```

2278 \def\bbl@ifknown@ttrib#1#2{%
2279   \let\bbl@tempa\@secondoftwo
2280   \bbl@loopx\bbl@tempb{#2}{%
2281     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2282   \ifin@
2283     \let\bbl@tempa\@firstoftwo
2284   \else

```

```

2285 \fi}%
2286 \bbl@tempa}

```

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

```

2287 \def\bbl@clear@ttribs{%
2288 \ifx\bbl@attributes\undefined\else
2289 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2290 \expandafter\bbl@clear@ttrib\bbl@tempa.
2291 }%
2292 \let\bbl@attributes\undefined
2293 \fi}
2294 \def\bbl@clear@ttrib#1-#2.{%
2295 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2296 \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`

```

2297 \bbl@trace{Macros for saving definitions}
2298 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2299 \newcount\babel@savecnt
2300 \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.

```

2301 \def\babel@save#1{%
2302 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2303 \toks@\expandafter{\originalTeX\let#1=}
2304 \bbl@exp{%
2305 \def\\originalTeX{\the\toks@<babel@number\babel@savecnt>\relax}}%
2306 \advance\babel@savecnt@one}
2307 \def\babel@savevariable#1{%
2308 \toks@\expandafter{\originalTeX #1=}
2309 \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@nonfrenchspacing` switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in `\babelprovide`. This new method should be ideally the default one.

```

2310 \def\bbl@frenchspacing{%
2311 \ifnum\the\sffcode`.=\@m

```

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

```

2312 \let\bbl@nonfrenchspacing\relax
2313 \else
2314 \frenchspacing
2315 \let\bbl@nonfrenchspacing\nonfrenchspacing
2316 \fi}
2317 \let\bbl@nonfrenchspacing\nonfrenchspacing
2318 \let\bbl@elt\relax
2319 \edef\bbl@fs@chars{%
2320 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2321 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2322 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
2323 \def\bbl@pre@fs{%
2324 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
2325 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
2326 \def\bbl@post@fs{%
2327 \bbl@save@sfcodes
2328 \edef\bbl@tempa{\bbl@cl{frspc}}}%
2329 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
2330 \if u\bbl@tempa % do nothing
2331 \else\if n\bbl@tempa % non french
2332 \def\bbl@elt##1##2##3{%
2333 \ifnum\sfcode`##1=##2\relax
2334 \babel@savevariable{\sfcode`##1}%
2335 \sfcode`##1=##3\relax
2336 \fi}%
2337 \bbl@fs@chars
2338 \else\if y\bbl@tempa % french
2339 \def\bbl@elt##1##2##3{%
2340 \ifnum\sfcode`##1=##3\relax
2341 \babel@savevariable{\sfcode`##1}%
2342 \sfcode`##1=##2\relax
2343 \fi}%
2344 \bbl@fs@chars
2345 \fi\fi\fi}

```

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.

```

2346 \bbl@trace{Short tags}
2347 \def\babeltags#1{%
2348 \edef\bbl@tempa{\zap@space#1 \@empty}%
2349 \def\bbl@tempb##1=##2\@{#}%
2350 \edef\bbl@tempc{%
2351 \noexpand\newcommand
2352 \expandafter\noexpand\csname ##1\endcsname{%
2353 \noexpand\protect
2354 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2355 \noexpand\newcommand
2356 \expandafter\noexpand\csname text##1\endcsname{%
2357 \noexpand\foreignlanguage{##2}}}
2358 \bbl@tempc}%
2359 \bbl@for\bbl@tempa\bbl@tempa{%
2360 \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.

```

2361 \bbl@trace{Hyphens}
2362 \@onlypreamble\babelhyphenation
2363 \AtEndOfPackage{%
2364   \newcommand\babelhyphenation[2][\@empty]{%
2365     \ifx\bbl@hyphenation@relax
2366       \let\bbl@hyphenation@\@empty
2367     \fi
2368     \ifx\bbl@hyphlist\@empty\else
2369       \bbl@warning{%
2370         You must not intermingle \string\selectlanguage\space and\%
2371         \string\babelhyphenation\space or some exceptions will not\%
2372         be taken into account. Reported}%
2373       \fi
2374       \ifx\@empty#1%
2375         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2376       \else
2377         \bbl@vforeach{#1}{%
2378           \def\bbl@tempa{##1}%
2379           \bbl@fixname\bbl@tempa
2380           \bbl@iflanguage\bbl@tempa{%
2381             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2382               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2383               {}%
2384               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2385               #2}}}%
2386         \fi}}

```

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

```

2387 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2388 \def\bbl@t@one{T1}
2389 \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`.

```

2390 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2391 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2392 \def\bbl@hyphen{%
2393   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2394 \def\bbl@hyphen@i#1#2{%
2395   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2396   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2397   {\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.

³² \TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

2398 \def\bbl@usehyphen#1{%
2399   \leavevmode
2400   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2401   \nobreak\hskip\z@skip}
2402 \def\bbl@usehyphen#1{%
2403   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2404 \def\bbl@hyphenchar{%
2405   \ifnum\hyphenchar\font=\m@ne
2406     \babeinullhyphen
2407   \else
2408     \char\hyphenchar\font
2409   \fi}

```

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

```

2410 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2411 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2412 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2413 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2414 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2415 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2416 \def\bbl@hy@repeat{%
2417   \bbl@usehyphen{%
2418     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2419 \def\bbl@hy@repeat{%
2420   \bbl@usehyphen{%
2421     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2422 \def\bbl@hy@empty{\hskip\z@skip}
2423 \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.

```

2424 \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.

```

2425 \bbl@trace{Multiencoding strings}
2426 \def\bbl@tglobal#1{\global\let#1#1}
2427 \def\bbl@recatcode#1{% TODO. Used only once?
2428   \@tempcnta="7F
2429   \def\bbl@tempa{%
2430     \ifnum\@tempcnta>"FF\else
2431       \catcode\@tempcnta=#1\relax
2432       \advance\@tempcnta\@ne
2433       \expandafter\bbl@tempa
2434     \fi}%
2435   \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).

```
2436 \@ifpackagewith{babel}{nocase}%
2437   {\let\bbl@patchuclc\relax}%
2438   {\def\bbl@patchuclc{%
2439     \global\let\bbl@patchuclc\relax
2440     \g@addto@macro\@uclclist{\reserved@b\reserved@b\bbl@uclc}}%
2441     \gdef\bbl@uclc##1{%
2442       \let\bbl@encoded\bbl@encoded@uclc
2443       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2444       {##1}%
2445       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2446         \csname\language @bbl@uclc\endcsname}%
2447       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2448     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2449     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2450 \langle\langle *More package options\rangle\rangle \equiv
2451 \DeclareOption{nocase}{}
2452 \rangle\rangle\langle\langle /More package options\rangle\rangle
```

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

```
2453 \langle\langle *More package options\rangle\rangle \equiv
2454 \let\bbl@opt@strings\@nnil % accept strings=value
2455 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2456 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2457 \def\BabelStringsDefault{generic}
2458 \rangle\rangle\langle\langle /More package options\rangle\rangle
```

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.

```
2459 \@onlypreamble\StartBabelCommands
2460 \def\StartBabelCommands{%
2461   \begingroup
2462   \bbl@recatcode{11}%
2463   \langle\langle Macros local to BabelCommands\rangle\rangle
2464   \def\bbl@provstring##1##2{%
2465     \providecommand##1{##2}%
2466     \bbl@tglobal##1}%
2467   \global\let\bbl@scafter\@empty
2468   \let\StartBabelCommands\bbl@startcmds
2469   \ifx\BabelLanguages\relax
2470     \let\BabelLanguages\CurrentOption
2471   \fi
2472   \begingroup
2473   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2474   \StartBabelCommands}
2475 \def\bbl@startcmds{%
2476   \ifx\bbl@screset\@nnil\else
2477     \bbl@usehooks{stopcommands}{}%
2478   \fi
```

```

2479 \endgroup
2480 \begingroup
2481 \@ifstar
2482 {\ifx\bbbl@opt@strings\@nnil
2483   \let\bbbl@opt@strings\BabelStringsDefault
2484   \fi
2485   \bbbl@startcmds@i}%
2486 \bbbl@startcmds@i}
2487 \def\bbbl@startcmds@i#1#2{%
2488 \edef\bbbl@L{\zap@space#1 \@empty}%
2489 \edef\bbbl@G{\zap@space#2 \@empty}%
2490 \bbbl@startcmds@ii}
2491 \let\bbbl@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.

```

2492 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
2493 \let\SetString@gobbletwo
2494 \let\bbbl@stringdef@gobbletwo
2495 \let\AfterBabelCommands@gobble
2496 \ifx\@empty#1%
2497 \def\bbbl@sc@label{generic}%
2498 \def\bbbl@encstring##1##2{%
2499 \ProvideTextCommandDefault##1{##2}%
2500 \bbbl@toglobal##1%
2501 \expandafter\bbbl@toglobal\csname\string?string##1\endcsname}%
2502 \let\bbbl@sctest\in@true
2503 \else
2504 \let\bbbl@sc@charset\space % <- zapped below
2505 \let\bbbl@sc@fontenc\space % <- " "
2506 \def\bbbl@tempa##1=##2\@nil{%
2507 \bbbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
2508 \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
2509 \def\bbbl@tempa##1 ##2{% space -> comma
2510 ##1%
2511 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
2512 \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
2513 \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
2514 \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
2515 \def\bbbl@encstring##1##2{%
2516 \bbbl@foreach\bbbl@sc@fontenc{%
2517 \bbbl@ifunset{T@###1}%
2518 {}%
2519 {\ProvideTextCommand##1{###1}{##2}%
2520 \bbbl@toglobal##1%
2521 \expandafter
2522 \bbbl@toglobal\csname###1\string##1\endcsname}}}%
2523 \def\bbbl@sctest{%
2524 \bbbl@xin@{\bbbl@opt@strings,}{\bbbl@sc@label,\bbbl@sc@fontenc,}}%
2525 \fi
2526 \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
2527 \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded

```

```

2528 \let\AfterBabelCommands\bbl@aftercmds
2529 \let\SetString\bbl@setstring
2530 \let\bbl@stringdef\bbl@encstring
2531 \else % ie, strings=value
2532 \bbl@sctest
2533 \ifin@
2534 \let\AfterBabelCommands\bbl@aftercmds
2535 \let\SetString\bbl@setstring
2536 \let\bbl@stringdef\bbl@provstring
2537 \fi\fi\fi
2538 \bbl@scswitch
2539 \ifx\bbl@G\@empty
2540 \def\SetString##1##2{%
2541 \bbl@error{Missing group for string \string##1}%
2542 {You must assign strings to some category, typically\\%
2543 captions or extras, but you set none}}%
2544 \fi
2545 \ifx\@empty#1%
2546 \bbl@usehooks{defaultcommands}{}%
2547 \else
2548 \@expandtwoargs
2549 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2550 \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).

```

2551 \def\bbl@forlang#1#2{%
2552 \bbl@for#1\bbl@L{%
2553 \bbl@xin@{, #1,}{, \BabelLanguages,}%
2554 \ifin@#2\relax\fi}}
2555 \def\bbl@scswitch{%
2556 \bbl@forlang\bbl@tempa{%
2557 \ifx\bbl@G\@empty\else
2558 \ifx\SetString\@gobbletwo\else
2559 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2560 \bbl@xin@{, \bbl@GL,}{, \bbl@screset,}%
2561 \ifin@\else
2562 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2563 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
2564 \fi
2565 \fi
2566 \fi}}
2567 \AtEndOfPackage{%
2568 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2569 \let\bbl@scswitch\relax}
2570 \onlypreamble\EndBabelCommands
2571 \def\EndBabelCommands{%
2572 \bbl@usehooks{stopcommands}{}%
2573 \endgroup
2574 \endgroup
2575 \bbl@scafter}
2576 \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.

```

2577 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2578   \bbl@forlang\bbl@tempa{%
2579     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2580     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2581       {\bbl@exp{%
2582         \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2583       }%
2584     \def\BabelString{#2}%
2585     \bbl@usehooks{stringprocess}{}%
2586     \expandafter\bbl@stringdef
2587     \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`.

```

2588 \ifx\bbl@opt@strings\relax
2589   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2590   \bbl@patchuclc
2591   \let\bbl@encoded\relax
2592   \def\bbl@encoded@uclc#1{%
2593     \@inmathwarn#1%
2594     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2595       \expandafter\ifx\csname ?\string#1\endcsname\relax
2596         \TextSymbolUnavailable#1%
2597       \else
2598         \csname ?\string#1\endcsname
2599       \fi
2600     \else
2601       \csname\cf@encoding\string#1\endcsname
2602     \fi}
2603 \else
2604   \def\bbl@scset#1#2{\def#1{#2}}
2605 \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.

```

2606 << *Macros local to BabelCommands >> ≡
2607 \def\SetStringLoop##1##2{%
2608   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2609   \count@\z@
2610   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2611     \advance\count@\@ne
2612     \toks@\expandafter{\bbl@tempa}%
2613     \bbl@exp{%
2614       \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2615       \count@=\the\count@\relax}}}%
2616 <</Macros local to BabelCommands >>
```

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

```

2617 \def\bbl@aftercmds#1{%
2618   \toks@\expandafter{\bbl@scafter#1}%
2619   \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.

```
2620 <<*Macros local to BabelCommands>> ≡
2621 \newcommand\SetCase[3][]{%
2622   \bbl@patchuclc
2623   \bbl@forlang\bbl@tempa{%
2624     \expandafter\bbl@encstring
2625     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2626     \expandafter\bbl@encstring
2627     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2628     \expandafter\bbl@encstring
2629     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2630 <</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.

```
2631 <<*Macros local to BabelCommands>> ≡
2632 \newcommand\SetHyphenMap[1]{%
2633   \bbl@forlang\bbl@tempa{%
2634     \expandafter\bbl@stringdef
2635     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2636 <</Macros local to BabelCommands>>
```

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

```
2637 \newcommand\BabelLower[2]{% one to one.
2638   \ifnum\lccode#1=#2\else
2639     \babel@savevariable{\lccode#1}%
2640     \lccode#1=#2\relax
2641   \fi}
2642 \newcommand\BabelLowerMM[4]{% many-to-many
2643   \@tempcnta=#1\relax
2644   \@tempcntb=#4\relax
2645   \def\bbl@tempa{%
2646     \ifnum\@tempcnta>#2\else
2647       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2648       \advance\@tempcnta#3\relax
2649       \advance\@tempcntb#3\relax
2650       \expandafter\bbl@tempa
2651     \fi}%
2652   \bbl@tempa}
2653 \newcommand\BabelLowerM0[4]{% many-to-one
2654   \@tempcnta=#1\relax
2655   \def\bbl@tempa{%
2656     \ifnum\@tempcnta>#2\else
2657       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2658       \advance\@tempcnta#3
2659       \expandafter\bbl@tempa
2660     \fi}%
2661   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2662 <<*More package options>> ≡
2663 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2664 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2665 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2666 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2667 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
```

2668 <</More package options>>

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

```
2669 \AtEndOfPackage{%
2670   \ifx\bbbl@opt@hyphenmap\undefined
2671     \bbbl@xin@{,}{\bbbl@language@opts}%
2672     \chardef\bbbl@opt@hyphenmap\ifin@4\else\ne\fi
2673   \fi}
```

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```
2674 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2675   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
2676 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
2677   \bbbl@trim@def\bbbl@tempa{#2}%
2678   \bbbl@xin@{.template}{\bbbl@tempa}%
2679   \ifin@
2680     \bbbl@ini@captions@template{#3}{#1}%
2681   \else
2682     \edef\bbbl@tempd{%
2683       \expandafter\expandafter\expandafter
2684       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2685     \bbbl@xin@
2686       {\expandafter\string\csname #2name\endcsname}%
2687       {\bbbl@tempd}%
2688     \ifin@ % Renew caption
2689       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
2690     \ifin@
2691       \bbbl@exp{%
2692         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2693         {\bbbl@scset\<#2name>\<#1#2name>}%
2694         {}}%
2695       \else % Old way converts to new way
2696         \bbbl@ifunset{#1#2name}%
2697         {\bbbl@exp{%
2698           \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2699           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2700           {\def\<#2name>{\<#1#2name>}}%
2701           {}}}%
2702         {}%
2703       \fi
2704     \else
2705       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2706       \ifin@ % New way
2707         \bbbl@exp{%
2708           \\bbbl@add\<captions#1>{\bbbl@scset\<#2name>\<#1#2name>}%
2709           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2710           {\bbbl@scset\<#2name>\<#1#2name>}%
2711           {}}%
2712         \else % Old way, but defined in the new way
2713           \bbbl@exp{%
2714             \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2715             \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2716             {\def\<#2name>{\<#1#2name>}}%
2717             {}}%
2718           \fi%
2719         \fi
2720       \@namedef{#1#2name}{#3}%

```



```

2721 \toks@\expandafter{\bbl@captionslist}%
2722 \bbl@exp{\in@{<#2name>}{\the\toks@}}%
2723 \ifin@else
2724 \bbl@exp{\bbl@add\bbl@captionslist{<#2name>}}%
2725 \bbl@tglobal\bbl@captionslist
2726 \fi
2727 \fi}
2728 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

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.

```

2729 \bbl@trace{Macros related to glyphs}
2730 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2731 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2732 \setbox\z@hbox{\lower\dimen\z@ \box\z@}\ht\z@ht\tw@ \dp\z@dp\tw@}

```

`\save@sfc@q` The macro `\save@sfc@q` is used to save and reset the current space factor.

```

2733 \def\save@sfc@q#1{\leavevmode
2734 \begingroup
2735 \edef\SF{\spacefactor\the\spacefactor}#1\SF
2736 \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.

```

2737 \ProvideTextCommand{\quotedblbase}{OT1}{%
2738 \save@sfc@q{\set@low@box{\textquotedblright\}}%
2739 \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

2740 \ProvideTextCommandDefault{\quotedblbase}{%
2741 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2742 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2743 \save@sfc@q{\set@low@box{\textquoteright\}}%
2744 \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

2745 \ProvideTextCommandDefault{\quotesinglbase}{%
2746 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2747 \ProvideTextCommand{\guillemetleft}{OT1}{%
2748 \ifmmode
2749 \ll
2750 \else

```

```

2751 \save@sf@q{\nobreak
2752 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2753 \fi}
2754 \ProvideTextCommand{\guillemetright}{OT1}{%
2755 \ifmmode
2756 \gg
2757 \else
2758 \save@sf@q{\nobreak
2759 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2760 \fi}
2761 \ProvideTextCommand{\guillemotleft}{OT1}{%
2762 \ifmmode
2763 \ll
2764 \else
2765 \save@sf@q{\nobreak
2766 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2767 \fi}
2768 \ProvideTextCommand{\guillemotright}{OT1}{%
2769 \ifmmode
2770 \gg
2771 \else
2772 \save@sf@q{\nobreak
2773 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2774 \fi}

```

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

```

2775 \ProvideTextCommandDefault{\guillemetleft}{%
2776 \UseTextSymbol{OT1}{\guillemetleft}}
2777 \ProvideTextCommandDefault{\guillemetright}{%
2778 \UseTextSymbol{OT1}{\guillemetright}}
2779 \ProvideTextCommandDefault{\guillemotleft}{%
2780 \UseTextSymbol{OT1}{\guillemotleft}}
2781 \ProvideTextCommandDefault{\guillemotright}{%
2782 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2783 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2784 \ifmmode
2785 <%
2786 \else
2787 \save@sf@q{\nobreak
2788 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2789 \fi}
2790 \ProvideTextCommand{\guilsinglright}{OT1}{%
2791 \ifmmode
2792 >%
2793 \else
2794 \save@sf@q{\nobreak
2795 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2796 \fi}

```

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

```

2797 \ProvideTextCommandDefault{\guilsinglleft}{%
2798 \UseTextSymbol{OT1}{\guilsinglleft}}
2799 \ProvideTextCommandDefault{\guilsinglright}{%
2800 \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
`\IJ` fonts. Therefore we fake it for the OT1 encoding.

```
2801 \DeclareTextCommand{\ij}{OT1}{%
2802   i\kern-0.02em\bbl@allowhyphens j}
2803 \DeclareTextCommand{\IJ}{OT1}{%
2804   I\kern-0.02em\bbl@allowhyphens J}
2805 \DeclareTextCommand{\ij}{T1}{\char188}
2806 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2807 \ProvideTextCommandDefault{\ij}{%
2808   \UseTextSymbol{OT1}{\ij}}
2809 \ProvideTextCommandDefault{\IJ}{%
2810   \UseTextSymbol{OT1}{\IJ}}
```

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

```
2811 \def\crrtic@{\hrule height0.1ex width0.3em}
2812 \def\crttic@{\hrule height0.1ex width0.33em}
2813 \def\ddj@{%
2814   \setbox0\hbox{d}\dimen@=\ht0
2815   \advance\dimen@1ex
2816   \dimen@.45\dimen@
2817   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2818   \advance\dimen@ii.5ex
2819   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2820 \def\DDJ@{%
2821   \setbox0\hbox{D}\dimen@=.55\ht0
2822   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2823   \advance\dimen@ii.15ex % correction for the dash position
2824   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2825   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2826   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2827 %
2828 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2829 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2830 \ProvideTextCommandDefault{\dj}{%
2831   \UseTextSymbol{OT1}{\dj}}
2832 \ProvideTextCommandDefault{\DJ}{%
2833   \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.

```
2834 \DeclareTextCommand{\SS}{OT1}{SS}
2835 \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 2836 \ProvideTextCommandDefault{\glq}{%
2837 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2838 \ProvideTextCommand{\grq}{T1}{%
2839 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2840 \ProvideTextCommand{\grq}{TU}{%
2841 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2842 \ProvideTextCommand{\grq}{OT1}{%
2843 \save@sf@q{\kern-.0125em
2844 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2845 \kern.07em\relax}}
2846 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2847 \ProvideTextCommandDefault{\glqq}{%
2848 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2849 \ProvideTextCommand{\grqq}{T1}{%
2850 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2851 \ProvideTextCommand{\grqq}{TU}{%
2852 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2853 \ProvideTextCommand{\grqq}{OT1}{%
2854 \save@sf@q{\kern-.07em
2855 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2856 \kern.07em\relax}}
2857 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2858 \ProvideTextCommandDefault{\flq}{%
2859 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2860 \ProvideTextCommandDefault{\frq}{%
2861 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2862 \ProvideTextCommandDefault{\flqq}{%
2863 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2864 \ProvideTextCommandDefault{\frqq}{%
2865 \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
`\umlautlow` default will be `\umlauthigh` (the normal positioning).

```
2866 \def\umlauthigh{%
2867 \def\bbl@umlauta##1{\leavevmode\bggroup%
2868 \expandafter\accent\csname\fontencoding dpos\endcsname
2869 ##1\bbl@allowhyphens\egroup}%
2870 \let\bbl@umlaute\bbl@umlauta}
2871 \def\umlautlow{%
2872 \def\bbl@umlauta{\protect\lower@umlaut}}
```

```

2873 \def\umlaute\lower@%
2874 \def\bbl@umlaute{\protect\lower@umlaut}}
2875 \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.

```

2876 \expandafter\ifx\csname U@D\endcsname\relax
2877 \csname newdimen\endcsname\U@D
2878 \fi

```

The following code fools T_EX's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

2879 \def\lower@umlaut#1{%
2880 \leavevmode\bggroup
2881 \U@D 1ex%
2882 {\setbox\z@\hbox{%
2883 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2884 \dimen@ -.45ex\advance\dimen@\ht\z@
2885 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2886 \expandafter\accent\csname\fontencoding dqpos\endcsname
2887 \fontdimen5\font\U@D #1%
2888 \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bbl@umlaut` 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@umlaut` and/or `\bbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2889 \AtBeginDocument{%
2890 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlaut{a}}%
2891 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2892 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2893 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2894 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlaut{o}}%
2895 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlaut{u}}%
2896 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlaut{A}}%
2897 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2898 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2899 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlaut{O}}%
2900 \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.

```

2901 \ifx\l@english\undefined
2902 \chardef\l@english\z@
2903 \fi
2904 % The following is used to cancel rules in ini files (see Amharic).
2905 \ifx\l@unhyphenated\undefined
2906 \newlanguage\l@unhyphenated
2907 \fi

```

9.13 Layout

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

```
2908 \bbl@trace{Bidi layout}
2909 \providecommand\IfBabelLayout[3]{#3}%
2910 \newcommand\BabelPatchSection[1]{%
2911   \@ifundefined{#1}{}{%
2912     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2913     \@namedef{#1}{%
2914       \ifstar{\bbl@presec@s{#1}}%
2915       {\@dblarg{\bbl@presec@x{#1}}}}}%
2916 \def\bbl@presec@x#1[#2]#3{%
2917   \bbl@exp{%
2918     \\\select@language@x{\bbl@main@language}%
2919     \\\bbl@cs{sspre@#1}%
2920     \\\bbl@cs{ss@#1}%
2921     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2922     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2923     \\\select@language@x{\language}}}%
2924 \def\bbl@presec@s#1#2{%
2925   \bbl@exp{%
2926     \\\select@language@x{\bbl@main@language}%
2927     \\\bbl@cs{sspre@#1}%
2928     \\\bbl@cs{ss@#1}*%
2929     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2930     \\\select@language@x{\language}}}%
2931 \IfBabelLayout{sectioning}%
2932   {\BabelPatchSection{part}%
2933    \BabelPatchSection{chapter}%
2934    \BabelPatchSection{section}%
2935    \BabelPatchSection{subsection}%
2936    \BabelPatchSection{subsubsection}%
2937    \BabelPatchSection{paragraph}%
2938    \BabelPatchSection{subparagraph}%
2939    \def\babel@toc#1{%
2940      \select@language@x{\bbl@main@language}}}%
2941 \IfBabelLayout{captions}%
2942   {\BabelPatchSection{caption}}}
```

9.14 Load engine specific macros

```
2943 \bbl@trace{Input engine specific macros}
2944 \ifcase\bbl@engine
2945   \input txtbabel.def
2946 \or
2947   \input luababel.def
2948 \or
2949   \input xebabel.def
2950 \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.

```
2951 \bbl@trace{Creating languages and reading ini files}
2952 \let\bbl@extend@ini\@gobble
2953 \newcommand\babelprovide[2][]{%
2954   \let\bbl@savelangname\language
```

```

2955 \edef\bbl@savelocaleid{\the\localeid}%
2956 % Set name and locale id
2957 \edef\languagename{#2}%
2958 \bbl{id@assign
2959 % Initialize keys
2960 \let\bbl@KVP@captions\@nil
2961 \let\bbl@KVP@date\@nil
2962 \let\bbl@KVP@import\@nil
2963 \let\bbl@KVP@main\@nil
2964 \let\bbl@KVP@script\@nil
2965 \let\bbl@KVP@language\@nil
2966 \let\bbl@KVP@hyphenrules\@nil
2967 \let\bbl@KVP@linebreaking\@nil
2968 \let\bbl@KVP@justification\@nil
2969 \let\bbl@KVP@mapfont\@nil
2970 \let\bbl@KVP@maparabic\@nil
2971 \let\bbl@KVP@mapdigits\@nil
2972 \let\bbl@KVP@intraspace\@nil
2973 \let\bbl@KVP@intrapenalty\@nil
2974 \let\bbl@KVP@onchar\@nil
2975 \let\bbl@KVP@transforms\@nil
2976 \global\let\bbl@release@transforms\@empty
2977 \let\bbl@KVP@alph\@nil
2978 \let\bbl@KVP@Alph\@nil
2979 \let\bbl@KVP@labels\@nil
2980 \bbl@csarg\let{KVP@labels*}\@nil
2981 \global\let\bbl@inidata\@empty
2982 \global\let\bbl@extend@ini\@gobble
2983 \gdef\bbl@key@list{;}%
2984 \bbl@forkv{#1}{% TODO - error handling
2985   \in@{/}{##1}%
2986   \ifin@
2987     \global\let\bbl@extend@ini\bbl@extend@ini@aux
2988     \bbl@renewinikey##1\@{##2}%
2989   \else
2990     \bbl@csarg\def{KVP@##1}{##2}%
2991   \fi}%
2992 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2993 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel#2}\@n\@tw}%
2994 % == init ==
2995 \ifx\bbl@screset\@undefined
2996   \bbl@ldfinit
2997 \fi
2998 % ==
2999 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3000 \ifcase\bbl@howloaded
3001   \let\bbl@lbkflag\@empty % new
3002 \else
3003   \ifx\bbl@KVP@hyphenrules\@nil\else
3004     \let\bbl@lbkflag\@empty
3005   \fi
3006   \ifx\bbl@KVP@import\@nil\else
3007     \let\bbl@lbkflag\@empty
3008   \fi
3009 \fi
3010 % == import, captions ==
3011 \ifx\bbl@KVP@import\@nil\else
3012   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
3013   {\ifx\bbl@initoload\relax

```

```

3014         \begingroup
3015         \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3016         \bb1@input@texini{#2}%
3017     \endgroup
3018     \else
3019         \xdef\bb1@KVP@import{\bb1@initoload}%
3020     \fi}%
3021 {}%
3022 \fi
3023 \ifx\bb1@KVP@captions\@nil
3024     \let\bb1@KVP@captions\bb1@KVP@import
3025 \fi
3026 % ==
3027 \ifx\bb1@KVP@transforms\@nil\else
3028     \bb1@replace\bb1@KVP@transforms{ },}%
3029 \fi
3030 % == Load ini ==
3031 \ifcase\bb1@howloaded
3032     \bb1@provide@new{#2}%
3033 \else
3034     \bb1@ifblank{#1}%
3035     {}% With \bb1@load@basic below
3036     {\bb1@provide@renew{#2}}%
3037 \fi
3038 % Post tasks
3039 % -----
3040 % == subsequent calls after the first provide for a locale ==
3041 \ifx\bb1@inidata\@empty\else
3042     \bb1@extend@ini{#2}%
3043 \fi
3044 % == ensure captions ==
3045 \ifx\bb1@KVP@captions\@nil\else
3046     \bb1@ifunset{\bb1@extracaps@#2}%
3047     {\bb1@exp{\labelensure[exclude=\\today]{#2}}}%
3048     {\toks@\expandafter\expandafter\expandafter
3049      {\csname \bb1@extracaps@#2\endcsname}%
3050     \bb1@exp{\labelensure[exclude=\\today,include=\\the\toks@]{#2}}%
3051     \bb1@ifunset{\bb1@ensure@\\language}%
3052     {\bb1@exp%
3053      \\\DeclareRobustCommand\<\bb1@ensure@\\language>[1]{%
3054       \\\foreignlanguage{\\language}%
3055       {####1}}}%
3056     {}%
3057     \bb1@exp%
3058     \\\bb1@tglobal\<\bb1@ensure@\\language>%
3059     \\\bb1@tglobal\<\bb1@ensure@\\language\space>}%
3060 \fi
3061 % ==
3062 % At this point all parameters are defined if 'import'. Now we
3063 % execute some code depending on them. But what about if nothing was
3064 % imported? We just set the basic parameters, but still loading the
3065 % whole ini file.
3066 \bb1@load@basic{#2}%
3067 % == script, language ==
3068 % Override the values from ini or defines them
3069 \ifx\bb1@KVP@script\@nil\else
3070     \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
3071 \fi
3072 \ifx\bb1@KVP@language\@nil\else

```



```

3073 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
3074 \fi
3075 % == onchar ==
3076 \ifx\bbl@KVP@onchar\@nil\else
3077 \bbl@luahyphenate
3078 \directlua{
3079   if Babel.locale_mapped == nil then
3080     Babel.locale_mapped = true
3081     Babel.linebreaking.add_before(Babel.locale_map)
3082     Babel.loc_to_scr = {}
3083     Babel.chr_to_loc = Babel.chr_to_loc or {}
3084   end}%
3085 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3086 \ifin@
3087 \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
3088 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}}%
3089 \fi
3090 \bbl@exp{\bbl@add\bbl@starthyphens
3091   {\bbl@patterns@lua{\languagename}}}%
3092 % TODO - error/warning if no script
3093 \directlua{
3094   if Babel.script_blocks['\bbl@cl{sbc}'] then
3095     Babel.loc_to_scr[\the\localeid] =
3096       Babel.script_blocks['\bbl@cl{sbc}']
3097     Babel.locale_props[\the\localeid].lc = \the\localeid\space
3098     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
3099   end
3100 }%
3101 \fi
3102 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3103 \ifin@
3104 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
3105 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
3106 \directlua{
3107   if Babel.script_blocks['\bbl@cl{sbc}'] then
3108     Babel.loc_to_scr[\the\localeid] =
3109       Babel.script_blocks['\bbl@cl{sbc}']
3110   end}%
3111 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
3112 \AtBeginDocument{%
3113   \bbl@patchfont{\bbl@mapselect}}%
3114   {\selectfont}}%
3115 \def\bbl@mapselect{%
3116   \let\bbl@mapselect\relax
3117   \edef\bbl@prefontid{\fontid\font}}%
3118 \def\bbl@mapdir##1{%
3119   {\def\languagename{##1}%
3120     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3121     \bbl@switchfont
3122     \directlua{
3123       Babel.locale_props[\the\csname bbl@id@##1\endcsname]
3124         [\bbl@prefontid] = \fontid\font\space}}}%
3125   \fi
3126   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
3127   \fi
3128 % TODO - catch non-valid values
3129 \fi
3130 % == mapfont ==
3131 % For bidi texts, to switch the font based on direction

```

```

3132 \ifx\bb1@KVP@mapfont\@nil\else
3133   \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}}}%
3134   {\bb1@error{Option '\bb1@KVP@mapfont' unknown for\%
3135     mapfont. Use 'direction'.%
3136     {See the manual for details.}}}%
3137   \bb1@ifunset{\bb1@lsys@\language\name}{\bb1@provide@lsys{\language\name}}}%
3138   \bb1@ifunset{\bb1@wdir@\language\name}{\bb1@provide@dirs{\language\name}}}%
3139   \ifx\bb1@mapselect\@undefined % TODO. See onchar.
3140     \AtBeginDocument{%
3141       \bb1@patchfont{\bb1@mapselect}}%
3142       {\selectfont}}%
3143     \def\bb1@mapselect{%
3144       \let\bb1@mapselect\relax
3145       \edef\bb1@prefontid{\fontid\font}}%
3146     \def\bb1@mapdir##1{%
3147       {\def\language\name{##1}%
3148         \let\bb1@ifrestoring\@firstoftwo % avoid font warning
3149         \bb1@switchfont
3150         \directlua{Babel.fontmap
3151           [\the\csname \bb1@wdir@##1\endcsname]%
3152           [\bb1@prefontid]=\fontid\font}}}%
3153     \fi
3154     \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language\name}}}%
3155   \fi
3156   % == Line breaking: intraspace, intrapenalty ==
3157   % For CJK, East Asian, Southeast Asian, if interspace in ini
3158   \ifx\bb1@KVP@intraspace\@nil\else % We can override the ini or set
3159     \bb1@csarg\edef{intsp@#2}{\bb1@KVP@intraspace}%
3160   \fi
3161   \bb1@provide@intraspace
3162   % == Line breaking: CJK quotes ==
3163   \ifcase\bb1@engine\or
3164     \bb1@xin@{/c}{/\bb1@cl{lnbrk}}%
3165     \ifin@
3166       \bb1@ifunset{\bb1@quote@\language\name}}}%
3167       {\directlua{
3168         Babel.locale_props[\the\localeid].cjk_quotes = {}
3169         local cs = 'op'
3170         for c in string.utfvalues(
3171           [[\csname \bb1@quote@\language\name\endcsname]]) do
3172           if Babel.cjk_characters[c].c == 'qu' then
3173             Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
3174           end
3175           cs = (cs == 'op') and 'cl' or 'op'
3176         end
3177       }}%
3178     \fi
3179   \fi
3180   % == Line breaking: justification ==
3181   \ifx\bb1@KVP@justification\@nil\else
3182     \let\bb1@KVP@linebreaking\bb1@KVP@justification
3183   \fi
3184   \ifx\bb1@KVP@linebreaking\@nil\else
3185     \bb1@xin@{,\bb1@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3186     \ifin@
3187       \bb1@csarg\xdef
3188         {\lnbrk@\language\name}{\expandafter\@car\bb1@KVP@linebreaking\@nil}%
3189     \fi
3190   \fi

```

```

3191 \bbl@xin@{/e}{/\bbl@c1{lbrk}}%
3192 \ifin@else\bbl@xin@{/k}{/\bbl@c1{lbrk}}\fi
3193 \ifin@bbl@arabicjust\fi
3194 % == Line breaking: hyphenate.other.(locale|script) ==
3195 \ifx\bbl@lbrkflag\empty
3196   \bbl@ifunset{bbl@hyotl@language}{}%
3197   {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
3198     \bbl@startcommands*\language}{}%
3199     \bbl@csarg\bbl@foreach{hyotl@language}{%
3200       \ifcase\bbl@engine
3201         \ifnum##1<257
3202           \SetHyphenMap{\BabelLower{##1}{##1}}%
3203         \fi
3204       \else
3205         \SetHyphenMap{\BabelLower{##1}{##1}}%
3206       \fi}%
3207   \bbl@endcommands}%
3208 \bbl@ifunset{bbl@hyots@language}{}%
3209 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
3210   \bbl@csarg\bbl@foreach{hyots@language}{%
3211     \ifcase\bbl@engine
3212       \ifnum##1<257
3213         \global\lccode##1=##1\relax
3214       \fi
3215     \else
3216       \global\lccode##1=##1\relax
3217     \fi}}%
3218 \fi
3219 % == Counters: maparabic ==
3220 % Native digits, if provided in ini (TeX level, xe and lua)
3221 \ifcase\bbl@engine\else
3222   \bbl@ifunset{bbl@dgnat@language}{}%
3223   {\expandafter\ifx\csname bbl@dgnat@language\endcsname\empty\else
3224     \expandafter\expandafter\expandafter
3225     \bbl@setdigits\csname bbl@dgnat@language\endcsname
3226     \ifx\bbl@KVP@maparabic\@nil\else
3227       \ifx\bbl@latinarabic\@undefined
3228         \expandafter\let\expandafter\@arabic
3229         \csname bbl@counter@language\endcsname
3230       \else % ie, if layout=counters, which redefines \@arabic
3231         \expandafter\let\expandafter\bbl@latinarabic
3232         \csname bbl@counter@language\endcsname
3233       \fi
3234     \fi
3235   \fi}%
3236 \fi
3237 % == Counters: mapdigits ==
3238 % Native digits (lua level).
3239 \ifodd\bbl@engine
3240   \ifx\bbl@KVP@mapdigits\@nil\else
3241     \bbl@ifunset{bbl@dgnat@language}{}%
3242     {\RequirePackage{luatexbase}%
3243       \bbl@activate@preotf
3244       \directlua{
3245         Babel = Babel or {} %% -> presets in luababel
3246         Babel.digits_mapped = true
3247         Babel.digits = Babel.digits or {}
3248         Babel.digits[\the\localeid] =
3249           table.pack(string.utfvalue('\bbl@c1{dgnat}'))

```

```

3250         if not Babel.numbers then
3251             function Babel.numbers(head)
3252                 local LOCALE = Babel.attr_locale
3253                 local GLYPH = node.id'glyph'
3254                 local inmath = false
3255                 for item in node.traverse(head) do
3256                     if not inmath and item.id == GLYPH then
3257                         local temp = node.get_attribute(item, LOCALE)
3258                         if Babel.digits[temp] then
3259                             local chr = item.char
3260                             if chr > 47 and chr < 58 then
3261                                 item.char = Babel.digits[temp][chr-47]
3262                             end
3263                         end
3264                     elseif item.id == node.id'math' then
3265                         inmath = (item.subtype == 0)
3266                     end
3267                 end
3268                 return head
3269             end
3270         end
3271     } }%
3272     \fi
3273 \fi
3274 % == Counters: alph, Alph ==
3275 % What if extras<lang> contains a \babel@save\@alph? It won't be
3276 % restored correctly when exiting the language, so we ignore
3277 % this change with the \bbl@alph@saved trick.
3278 \ifx\bbl@KVP@alph@nil\else
3279     \bbl@extras@wrap{\\bbl@alph@saved}%
3280     {\let\bbl@alph@saved\@alph}%
3281     {\let\@alph\bbl@alph@saved
3282     \babel@save\@alph}%
3283     \bbl@exp{%
3284         \\bbl@add\<extras\language\name>{%
3285             \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language\name>}}%
3286 \fi
3287 \ifx\bbl@KVP@Alph@nil\else
3288     \bbl@extras@wrap{\\bbl@Alph@saved}%
3289     {\let\bbl@Alph@saved\@Alph}%
3290     {\let\@Alph\bbl@Alph@saved
3291     \babel@save\@Alph}%
3292     \bbl@exp{%
3293         \\bbl@add\<extras\language\name>{%
3294             \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language\name>}}%
3295 \fi
3296 % == require.babel in ini ==
3297 % To load or reload the babel-*.tex, if require.babel in ini
3298 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3299     \bbl@ifunset\bbl@rqtex@\language\name}{%
3300         {\xdef\babel@beforestart\bbl@rqtex@\language\name\endcsname\@empty\else
3301             \let\babel@beforestart\@gobbletwo
3302             \chardef\atcatcode=\catcode`\@
3303             \catcode`\@=11\relax
3304             \bbl@input@texini{\bbl@cs{rqtex@\language\name}}%
3305             \catcode`\@=\atcatcode
3306             \let\atcatcode\relax
3307             \global\bbl@csarg\let{rqtex@\language\name}\relax
3308         \fi}%

```

```

3309 \fi
3310 % == frenchspacing ==
3311 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
3312 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
3313 \ifin@
3314 \bbbl@extras@wrap{\bbbl@pre@fs}%
3315 {\bbbl@pre@fs}%
3316 {\bbbl@post@fs}%
3317 \fi
3318 % == Release saved transforms ==
3319 \bbbl@release@transforms\relax % \relax closes the last item.
3320 % == main ==
3321 \ifx\bbbl@KVP@main\@nil % Restore only if not 'main'
3322 \let\language\bbbl@savelangname
3323 \chardef\localeid\bbbl@savelocaleid\relax
3324 \fi}

```

Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbbl@startcommands opens a group.

```

3325 \def\bbbl@provide@new#1{%
3326 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3327 \namedef{extras#1}{}%
3328 \namedef{noextras#1}{}%
3329 \bbbl@startcommands*{#1}{captions}%
3330 \ifx\bbbl@KVP@captions\@nil % and also if import, implicit
3331 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
3332 \ifx##1\@empty\else
3333 \bbbl@exp{%
3334 \SetString\##1{%
3335 \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
3336 \expandafter\bbbl@tempb
3337 \fi}%
3338 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
3339 \else
3340 \ifx\bbbl@initoload\relax
3341 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
3342 \else
3343 \bbbl@read@ini{\bbbl@initoload}2% % Same
3344 \fi
3345 \fi
3346 \StartBabelCommands*{#1}{date}%
3347 \ifx\bbbl@KVP@import\@nil
3348 \bbbl@exp{%
3349 \SetString\today{\bbbl@nocaption{today}{#1today}}}%
3350 \else
3351 \bbbl@savetoday
3352 \bbbl@savedate
3353 \fi
3354 \bbbl@endcommands
3355 \bbbl@load@basic{#1}%
3356 % == hyphenmins == (only if new)
3357 \bbbl@exp{%
3358 \gdef\<#1hyphenmins>{%
3359 {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
3360 {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}%
3361 % == hyphenrules (also in renew) ==
3362 \bbbl@provide@hyphens{#1}%
3363 \ifx\bbbl@KVP@main\@nil\else
3364 \expandafter\main@language\expandafter{#1}%

```

```

3365 \fi}
3366 %
3367 \def\bbbl@provide@renew#1{%
3368 \ifx\bbbl@KVP@captions\@nil\else
3369 \StartBabelCommands*{#1}{captions}%
3370 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
3371 \EndBabelCommands
3372 \fi
3373 \ifx\bbbl@KVP@import\@nil\else
3374 \StartBabelCommands*{#1}{date}%
3375 \bbbl@savetoday
3376 \bbbl@savestate
3377 \EndBabelCommands
3378 \fi
3379 % == hyphenrules (also in new) ==
3380 \ifx\bbbl@lbfkflag\@empty
3381 \bbbl@provide@hyphens{#1}%
3382 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

```

3383 \def\bbbl@load@basic#1{%
3384 \ifcase\bbbl@howloaded\or\or
3385 \ifcase\csname bbl@llevel@\language\endcsname
3386 \bbbl@csarg\let\lname@\language\relax
3387 \fi
3388 \fi
3389 \bbbl@ifunset{\bbbl@lname@#1}%
3390 {\def\BabelBeforeIni##1##2{%
3391 \begingroup
3392 \let\bbbl@ini@captions@aux\@gobbletwo
3393 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3394 \bbbl@read@ini{##1}1%
3395 \ifx\bbbl@initoload\relax\endinput\fi
3396 \endgroup}%
3397 \begingroup % boxed, to avoid extra spaces:
3398 \ifx\bbbl@initoload\relax
3399 \bbbl@input@texini{##1}%
3400 \else
3401 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
3402 \fi
3403 \endgroup}%
3404 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3405 \def\bbbl@provide@hyphens#1{%
3406 \let\bbbl@tempa\relax
3407 \ifx\bbbl@KVP@hyphenrules\@nil\else
3408 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3409 \bbbl@foreach\bbbl@KVP@hyphenrules{%
3410 \ifx\bbbl@tempa\relax % if not yet found
3411 \bbbl@ifsamestring{##1}{+}%
3412 {\bbbl@exp{\addlanguage\<l@##1>}}}%
3413 }%
3414 \bbbl@ifunset{l@##1}%
3415 }%
3416 {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}}%
3417 \fi}%

```

```

3418 \fi
3419 \ifx\bbbl@tempa\relax %           if no opt or no language in opt found
3420   \ifx\bbbl@KVP@import\@nil
3421     \ifx\bbbl@initoload\relax\else
3422       \bbbl@exp{%                   and hyphenrules is not empty
3423         \\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3424         }%
3425         {\let\\bbbl@tempa\<l@\bbbl@cl{hyphr}>}}%
3426   \fi
3427 \else % if importing
3428   \bbbl@exp{%                       and hyphenrules is not empty
3429     \\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3430     }%
3431     {\let\\bbbl@tempa\<l@\bbbl@cl{hyphr}>}}%
3432 \fi
3433 \fi
3434 \bbbl@ifunset{\bbbl@tempa}%         ie, relax or undefined
3435 {\bbbl@ifunset{l@#1}%              no hyphenrules found - fallback
3436   {\bbbl@exp{\\adddialect\<l@#1>\language}}%
3437   }%
3438   {\bbbl@exp{\\adddialect\<l@#1>\bbbl@tempa}}% found in opt list or ini

```

The reader of babel-...tex files. We reset temporarily some catcodes.

```

3439 \def\bbbl@input@texini#1{%
3440   \bbbl@bsphack
3441   \bbbl@exp{%
3442     \catcode\\%=14 \catcode\\=0
3443     \catcode\\={1 \catcode\\}=2
3444     \lowercase{\\InputIfFileExists{babel-#1.tex}{}}%
3445     \catcode\\%= \the\catcode\\% \relax
3446     \catcode\\= \the\catcode\\% \relax
3447     \catcode\\={ \the\catcode\\% \relax
3448     \catcode\\= \the\catcode\\% \relax}%
3449   \bbbl@esphack}

```

The following macros read and store ini files (but don't process them). For each line, there are 3 possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbbl@read@ini.

```

3450 \def\bbbl@inline#1\bbbl@inline{%
3451   \@ifnextchar[\bbbl@inisect{\@ifnextchar;\bbbl@iniskip\bbbl@inistore}#1\@@}% ]
3452 \def\bbbl@inisect[#1]#2\@@{\def\bbbl@section{#1}}
3453 \def\bbbl@iniskip#1\@@{%           if starts with ;
3454 \def\bbbl@inistore#1=#2\@@{%       full (default)
3455   \bbbl@trim@def\bbbl@tempa{#1}%
3456   \bbbl@trim\toks@{#2}%
3457   \bbbl@xin@{\bbbl@section/\bbbl@tempa}{\bbbl@key@list}%
3458   \ifin@else
3459     \bbbl@exp{%
3460       \\g@addto@macro\\bbbl@inidata{%
3461         \\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3462   \fi}
3463 \def\bbbl@inistore@min#1=#2\@@{%   minimal (maybe set in \bbbl@read@ini)
3464   \bbbl@trim@def\bbbl@tempa{#1}%
3465   \bbbl@trim\toks@{#2}%
3466   \bbbl@xin@{.identification.}{.\bbbl@section.}%
3467   \ifin@
3468     \bbbl@exp{\\g@addto@macro\\bbbl@inidata{%
3469       \\bbbl@elt{identification}{\bbbl@tempa}{\the\toks@}}}%
3470   \fi}

```

Now, the 'main loop', which ****must be executed inside a group****. At this point, \bbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

```

3471 \ifx\bbl@readstream\@undefined
3472 \csname newread\endcsname\bbl@readstream
3473 \fi
3474 \def\bbl@read@ini#1#2{%
3475   \global\let\bbl@extend@ini\@gobble
3476   \openin\bbl@readstream=babel-#1.ini
3477   \ifeof\bbl@readstream
3478     \bbl@error
3479     {There is no ini file for the requested language\%
3480      (#1). Perhaps you misspelled it or your installation\%
3481      is not complete.}%
3482     {Fix the name or reinstall babel.}%
3483   \else
3484     % == Store ini data in \bbl@inidata ==
3485     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3486     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3487     \bbl@info{Importing
3488               \ifcase#2font and identification \or basic \fi
3489               data for \language\%
3490               from babel-#1.ini. Reported}%
3491     \ifnum#2=\z@
3492       \global\let\bbl@inidata\@empty
3493       \let\bbl@inistore\bbl@inistore@min % Remember it's local
3494     \fi
3495     \def\bbl@section{identification}%
3496     \bbl@exp{\bbl@inistore tag.ini=#1\\@}%
3497     \bbl@inistore load.level=#2\\@
3498     \loop
3499     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3500       \endlinechar\m@ne
3501       \read\bbl@readstream to \bbl@line
3502       \endlinechar\^^M
3503       \ifx\bbl@line\@empty\else
3504         \expandafter\bbl@iniline\bbl@line\bbl@iniline
3505       \fi
3506     \repeat
3507     % == Process stored data ==
3508     \bbl@csarg\xdef{lini@\language}{#1}%
3509     \bbl@read@ini@aux
3510     % == 'Export' data ==
3511     \bbl@ini@exports{#2}%
3512     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
3513     \global\let\bbl@inidata\@empty
3514     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
3515     \bbl@tglobal\bbl@ini@loaded
3516   \fi}
3517 \def\bbl@read@ini@aux{%
3518   \let\bbl@savestrings\@empty
3519   \let\bbl@savetoday\@empty
3520   \let\bbl@savestate\@empty
3521   \def\bbl@elt##1##2##3{%
3522     \def\bbl@section{##1}%

```



```

3523 \in@{=date.}{=##1}% Find a better place
3524 \ifin@
3525 \bbl@ini@calendar{##1}%
3526 \fi
3527 \bbl@ifunset{bbl@inikv@##1}{}%
3528 {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
3529 \bbl@inidata}

```

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

```

3530 \def\bbl@extend@ini@aux#1{%
3531 \bbl@startcommands*{#1}{captions}%
3532 % Activate captions/... and modify exports
3533 \bbl@csarg\def{inikv@captions.licr}##1##2{%
3534 \setlocalecaption{#1}{##1}{##2}}%
3535 \def\bbl@inikv@captions##1##2{%
3536 \bbl@ini@captions@aux{##1}{##2}}%
3537 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3538 \def\bbl@exportkey##1##2##3{%
3539 \bbl@ifunset{bbl@kv@##2}{}%
3540 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
3541 \bbl@exp{\global\let<bbl@##1@languagename>\<bbl@kv@##2>}}%
3542 \fi}}%
3543 % As with \bbl@read@ini, but with some changes
3544 \bbl@read@ini@aux
3545 \bbl@ini@exports\tw@
3546 % Update inidata@lang by pretending the ini is read.
3547 \def\bbl@elt##1##2##3{%
3548 \def\bbl@section{##1}%
3549 \bbl@iniline##2=##3\bbl@iniline}%
3550 \csname bbl@inidata@#1\endcsname
3551 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
3552 \StartBabelCommands*{#1}{date}% And from the import stuff
3553 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3554 \bbl@savetoday
3555 \bbl@savestate
3556 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections. To be improved.

```

3557 \def\bbl@ini@calendar#1{%
3558 \lowercase{\def\bbl@tempa{=#1=}}%
3559 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3560 \bbl@replace\bbl@tempa{=date.}{}%
3561 \in@{.licr=}{#1=}%
3562 \ifin@
3563 \ifcase\bbl@engine
3564 \bbl@replace\bbl@tempa{.licr=}{}%
3565 \else
3566 \let\bbl@tempa\relax
3567 \fi
3568 \fi
3569 \ifx\bbl@tempa\relax\else
3570 \bbl@replace\bbl@tempa{=}{}%
3571 \bbl@exp{%
3572 \def<bbl@inikv@#1>####1####2{%
3573 \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
3574 \fi}

```

A key with a slash in \babelprovide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has

not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in `\bbl@inistore` above).

```

3575 \def\bbl@renewinikey#1/#2\@#3{%
3576   \edef\bbl@tempa{\zap@space #1 \@empty}%    section
3577   \edef\bbl@tempb{\zap@space #2 \@empty}%    key
3578   \bbl@trim\toks@{#3}%                        value
3579   \bbl@exp{%
3580     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
3581     \\g@addto@macro\\bbl@inidata{%
3582       \\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

3583 \def\bbl@exportkey#1#2#3{%
3584   \bbl@ifunset{bbl@kv@#2}%
3585     {\bbl@csarg\gdef{#1@\language}\@empty}%
3586     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3587       \bbl@csarg\gdef{#1@\language}\@empty}%
3588   \else
3589     \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3590   \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@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

```

3591 \def\bbl@iniwarning#1{%
3592   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3593   {\bbl@warning{%
3594     From babel-\bbl@cs{lini@\language}.ini:\\
3595     \bbl@cs{kv@identification.warning#1}\\
3596     Reported }}}
3597 %
3598 \let\bbl@release@transforms\@empty
3599 %
3600 \def\bbl@ini@exports#1{%
3601   % Identification always exported
3602   \bbl@iniwarning{%
3603     \ifcase\bbl@engine
3604       \bbl@iniwarning{.pdflatex}%
3605     \or
3606       \bbl@iniwarning{.lualatex}%
3607     \or
3608       \bbl@iniwarning{.xelatex}%
3609     \fi%
3610     \bbl@exportkey{llevel}{identification.load.level}{}%
3611     \bbl@exportkey{elname}{identification.name.english}{}%
3612     \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
3613       {\csname bbl@elname@\language\endcsname}}%
3614     \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
3615     \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
3616     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3617     \bbl@exportkey{esname}{identification.script.name}{}%
3618     \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
3619       {\csname bbl@esname@\language\endcsname}}%
3620     \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3621     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3622     % Also maps bcp47 -> language
3623     \ifbbl@bcptoname
3624       \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcpl}}{\language}%

```

```

3625 \fi
3626 % Conditional
3627 \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
3628   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3629   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3630   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3631   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3632   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3633   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3634   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3635   \bbl@exportkey{intsp}{typography.intraspaces}{}%
3636   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3637   \bbl@exportkey{chrng}{characters.ranges}{}%
3638   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3639   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3640   \ifnum#1=\tw@        % only (re)new
3641     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3642     \bbl@toGLOBAL\bbl@savetoday
3643     \bbl@toGLOBAL\bbl@savestate
3644     \bbl@savestrings
3645   \fi
3646 \fi}

```

A shared handler for key=val lines to be stored in \bbl@kv@<section>.<key>.

```

3647 \def\bbl@inikv#1#2{%      key=value
3648   \toks@{#2}%             This hides #'s from ini values
3649   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

By default, the following sections are just read. Actions are taken later.

```

3650 \let\bbl@inikv@identification\bbl@inikv
3651 \let\bbl@inikv@typography\bbl@inikv
3652 \let\bbl@inikv@characters\bbl@inikv
3653 \let\bbl@inikv@numbers\bbl@inikv

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

3654 \def\bbl@inikv@counters#1#2{%
3655   \bbl@ifsamestring{#1}{digits}%
3656   {\bbl@error{The counter name 'digits' is reserved for mapping\\
3657     decimal digits}%
3658     {Use another name.}}%
3659   }%
3660 \def\bbl@tempc{#1}%
3661 \bbl@trim@def{\bbl@tempb*}{#2}%
3662 \in@{.1$}{#1$}%
3663 \ifin@
3664   \bbl@replace\bbl@tempc{.1}{}%
3665   \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3666   \noexpand\bbl@alphanumeric{\bbl@tempc}%
3667 \fi
3668 \in@{.F.}{#1}%
3669 \ifin@else\in@{.S.}{#1}\fi
3670 \ifin@
3671   \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3672 \else
3673   \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3674   \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3675   \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3676 \fi}

```

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.

```

3677 \ifcase\bbl@engine
3678   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3679     \bbl@ini@captions@aux{#1}{#2}}
3680 \else
3681   \def\bbl@inikv@captions#1#2{%
3682     \bbl@ini@captions@aux{#1}{#2}}
3683 \fi

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

3684 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3685   \bbl@replace\bbl@tempa{.template}{}}%
3686   \def\bbl@toreplace{#1}{}}%
3687   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3688   \bbl@replace\bbl@toreplace{[{}]{\csname}%
3689   \bbl@replace\bbl@toreplace{[{}]{\csname the}%
3690   \bbl@replace\bbl@toreplace{[{}]{name\endcsname}}}%
3691   \bbl@replace\bbl@toreplace{[{}]{\endcsname}}}%
3692   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3693   \ifin@
3694     \@nameuse{bbl@patch\bbl@tempa}%
3695     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3696   \fi
3697   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3698   \ifin@
3699     \toks@{\expandafter{\bbl@toreplace}%
3700     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}}%
3701   \fi}
3702 \def\bbl@ini@captions@aux#1#2{%
3703   \bbl@trim@def\bbl@tempa{#1}%
3704   \bbl@xin@{.template}{\bbl@tempa}%
3705   \ifin@
3706     \bbl@ini@captions@template{#2}\languagename
3707   \else
3708     \bbl@ifblank{#2}%
3709     {\bbl@exp{%
3710       \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3711     {\bbl@trim\toks@{#2}}}%
3712     \bbl@exp{%
3713       \bbl@add\bbl@savestrings{%
3714         \SetString\<\bbl@tempa name>{\the\toks@}}}%
3715     \toks@\expandafter{\bbl@captionslist}%
3716     \bbl@exp{\in{\<\bbl@tempa name>}{\the\toks@}}}%
3717     \ifin@else
3718       \bbl@exp{%
3719         \bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3720         \bbl@toglobal\<bbl@extracaps@\languagename>}%
3721       \fi
3722     \fi}

```

Labels. Captions must contain just strings, no format at all, so there is new group in ini files.

```

3723 \def\bbl@list@the{%
3724   part,chapter,section,subsection,subsubsection,paragraph,%
3725   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3726   table,page,footnote,mpfootnote,mpfn}
3727 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3728   \bbl@ifunset{bbl@map@#1@\languagename}%

```

```

3729 {\@nameuse{#1}}%
3730 {\@nameuse{bbl@map@#1@\languagename}}%
3731 \def\bbl@inikv@labels#1#2{%
3732 \in@{.map}{#1}%
3733 \ifin@
3734 \ifx\bbl@KVP@labels\@nil\else
3735 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3736 \ifin@
3737 \def\bbl@tempc{#1}%
3738 \bbl@replace\bbl@tempc{.map}{}%
3739 \in@{, #2, }{, arabic, roman, Roman, alph, Alph, fnsymbol,}%
3740 \bbl@exp{%
3741 \gdef\<bbl@map@\bbl@tempc @\languagename>%
3742 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3743 \bbl@foreach\bbl@list@the{%
3744 \bbl@ifunset{the##1}{}%
3745 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3746 \bbl@exp{%
3747 \\bbl@sreplace\<the##1>%
3748 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3749 \\bbl@sreplace\<the##1>%
3750 {\<\@empty @\bbl@tempc>\<c@##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3751 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3752 \toks@ \expandafter\expandafter\expandafter{%
3753 \csname the##1\endcsname}%
3754 \expandafter\def\csname the##1\endcsname{{\the\toks@}}%
3755 \fi}}%
3756 \fi
3757 \fi
3758 %
3759 \else
3760 %
3761 % The following code is still under study. You can test it and make
3762 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3763 % language dependent.
3764 \in@{enumerate.}{#1}%
3765 \ifin@
3766 \def\bbl@tempa{#1}%
3767 \bbl@replace\bbl@tempa{enumerate.}{}%
3768 \def\bbl@toreplace{#2}%
3769 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3770 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3771 \bbl@replace\bbl@toreplace{}{\endcsname{}}%
3772 \toks@ \expandafter{\bbl@toreplace}%
3773 % TODO. Execute only once:
3774 \bbl@exp{%
3775 \\bbl@add\<extras\languagename>{%
3776 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3777 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3778 \\bbl@toggle\<extras\languagename>}%
3779 \fi
3780 \fi}

```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```

3781 \def\bbl@chapttype{chapter}
3782 \ifx\@makechapterhead\@undefined

```

```

3783 \let\bbl@patchchapter\relax
3784 \else\ifx\thechapter\@undefined
3785 \let\bbl@patchchapter\relax
3786 \else\ifx\ps@headings\@undefined
3787 \let\bbl@patchchapter\relax
3788 \else
3789 \def\bbl@patchchapter{%
3790 \global\let\bbl@patchchapter\relax
3791 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3792 \bbl@tglobal\appendix
3793 \bbl@sreplace\ps@headings
3794 {\@chapapp\thechapter}%
3795 {\bbl@chapterformat}%
3796 \bbl@tglobal\ps@headings
3797 \bbl@sreplace\chaptermark
3798 {\@chapapp\thechapter}%
3799 {\bbl@chapterformat}%
3800 \bbl@tglobal\chaptermark
3801 \bbl@sreplace\@makechapterhead
3802 {\@chapapp\space\thechapter}%
3803 {\bbl@chapterformat}%
3804 \bbl@tglobal\@makechapterhead
3805 \gdef\bbl@chapterformat{%
3806 \bbl@ifunset{\bbl@\bbl@chapttype fmt@\languagename}%
3807 {\@chapapp\space\thechapter}
3808 {\@nameuse{\bbl@\bbl@chapttype fmt@\languagename}}}}
3809 \let\bbl@patchappendix\bbl@patchchapter
3810 \fi\fi\fi
3811 \ifx\@part\@undefined
3812 \let\bbl@patchpart\relax
3813 \else
3814 \def\bbl@patchpart{%
3815 \global\let\bbl@patchpart\relax
3816 \bbl@sreplace\@part
3817 {\partname\nobreakspace\thepart}%
3818 {\bbl@partformat}%
3819 \bbl@tglobal\@part
3820 \gdef\bbl@partformat{%
3821 \bbl@ifunset{\bbl@partfmt@\languagename}%
3822 {\partname\nobreakspace\thepart}
3823 {\@nameuse{\bbl@partfmt@\languagename}}}}
3824 \fi

```

Date. TODO. Document

```

3825 % Arguments are _not_ protected.
3826 \let\bbl@calendar\@empty
3827 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3828 \def\bbl@localedate#1#2#3#4{%
3829 \begingroup
3830 \ifx\@empty#1\@empty\else
3831 \let\bbl@ld@calendar\@empty
3832 \let\bbl@ld@variant\@empty
3833 \edef\bbl@tempa{\zap@space#1 \@empty}%
3834 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3835 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3836 \edef\bbl@calendar{%
3837 \bbl@ld@calendar
3838 \ifx\bbl@ld@variant\@empty\else
3839 .\bbl@ld@variant

```

```

3840     \fi}%
3841     \bbl@replace\bbl@calendar{gregorian}{}%
3842     \fi
3843     \bbl@cased
3844     {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3845 \endgroup}
3846 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3847 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3848     \bbl@trim@def\bbl@tempa{#1.#2}%
3849     \bbl@ifsamestring{\bbl@tempa}{months.wide}%         to savedate
3850     {\bbl@trim@def\bbl@tempa{#3}%
3851         \bbl@trim\toks@{#5}%
3852         \@temptokena\expandafter{\bbl@savestate}%
3853         \bbl@exp{% Reverse order - in ini last wins
3854             \def\\bbl@savestate{%
3855                 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3856                 \the\@temptokena}}}%
3857     {\bbl@ifsamestring{\bbl@tempa}{date.long}%         defined now
3858         {\lowercase{\def\bbl@tempb{#6}}%
3859             \bbl@trim@def\bbl@toreplace{#5}%
3860             \bbl@TG@@date
3861             \bbl@ifunset{bbl@date@\languagename @}%
3862             {\bbl@exp{% TODO. Move to a better place.
3863                 \gdef\<\languagename date>{\protect\<\languagename date >}%
3864                 \gdef\<\languagename date >####1####2####3{%
3865                     \\bbl@usedategroupttrue
3866                     \<bbl@ensure@\languagename>{%
3867                         \\localedate{####1}{####2}{####3}}}%
3868                     \\bbl@add\\bbl@savetoday{%
3869                         \\SetString\\today{%
3870                             \<\languagename date>%
3871                             {\the\year}{\the\month}{\the\day}}}%
3872                     }%
3873                     \global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3874                     \ifx\bbl@tempb\@empty\else
3875                         \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3876                     \fi}%
3877                 }}}

```

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. Note after `\bbl@replace \toks@` contains the resulting string, which is used by `\bbl@replace@finish@iii` (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3878 \let\bbl@calendar\@empty
3879 \newcommand\BabelDateSpace{\nobreakspace}
3880 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3881 \newcommand\BabelDated[1]{\number#1}
3882 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3883 \newcommand\BabelDateM[1]{\number#1}
3884 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3885 \newcommand\BabelDateMMMM[1]{%
3886     \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3887 \newcommand\BabelDatey[1]{\number#1}%
3888 \newcommand\BabelDateyy[1]{%
3889     \ifnum#1<10 0\number#1 %
3890     \else\ifnum#1<100 \number#1 %
3891     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3892     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %

```

```

3893 \else
3894 \bbl@error
3895 {Currently two-digit years are restricted to the\
3896 range 0-9999.}%
3897 {There is little you can do. Sorry.}%
3898 \fi\fi\fi\fi}
3899 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3900 \def\bbl@replace@finish@iii#1{%
3901 \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3902 \def\bbl@TG@date{%
3903 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3904 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3905 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3906 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3907 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3908 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3909 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3910 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3911 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3912 \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3913 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[###1|]}%
3914 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[###2|]}%
3915 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[###3|]}%
3916 \bbl@replace@finish@iii\bbl@toreplace}
3917 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3918 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3919 \let\bbl@release@transforms\@empty
3920 \namedef{bbl@inikv@transforms.prehyphenation}{%
3921 \bbl@transforms\babelprehyphenation}
3922 \namedef{bbl@inikv@transforms.posthyphenation}{%
3923 \bbl@transforms\babelposthyphenation}
3924 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3925 \begingroup
3926 \catcode\%=12
3927 \catcode\&=14
3928 \gdef\bbl@transforms#1#2#3{&%
3929 \ifx\bbl@KVP@transforms\@nil\else
3930 \directlua{
3931 str = [=[#2]=]
3932 str = str:gsub('%.%d+%.%d+$', '')
3933 tex.print([[ \def\string\babeltempa{]] .. str .. [ ]]])
3934 }&%
3935 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3936 \ifin@
3937 \in@{.0$}{#2$}&%
3938 \ifin@
3939 \g@addto@macro\bbl@release@transforms{&%
3940 \relax\bbl@transforms@aux#1{\languagename}{#3}}&%
3941 \else
3942 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3943 \fi
3944 \fi
3945 \fi}
3946 \endgroup

```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.


```

3947 \def\bbl@provide@lsys#1{%
3948   \bbl@ifunset{bbl@lname@#1}%
3949     {\bbl@load@info{#1}}%
3950     }%
3951   \bbl@csarg\let{lsys@#1}\@empty
3952   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3953   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3954   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3955   \bbl@ifunset{bbl@lname@#1}{}%
3956     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3957   \ifcase\bbl@engine\or\or
3958     \bbl@ifunset{bbl@prehc@#1}{}%
3959     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3960      }%
3961     {\ifx\bbl@xenoxyph\@undefined
3962      \let\bbl@xenoxyph\bbl@xenoxyph@d
3963      \ifx\AtBeginDocument\@notprerr
3964        \expandafter\@secondoftwo % to execute right now
3965        \fi
3966        \AtBeginDocument{%
3967          \bbl@patchfont{\bbl@xenoxyph}%
3968          \expandafter\selectlanguage\expandafter{\language}%
3969        \fi}%
3970   \fi
3971   \bbl@csarg\bbl@tglobal{lsys@#1}}
3972 \def\bbl@xenoxyph@d{%
3973   \bbl@ifset{bbl@prehc@language}%
3974     {\ifnum\hyphenchar\font=\defaultshyphenchar
3975      \iffontchar\font\bbl@c1{prehc}\relax
3976        \hyphenchar\font\bbl@c1{prehc}\relax
3977      \else\iffontchar\font"200B
3978        \hyphenchar\font"200B
3979      \else
3980        \bbl@warning
3981          {Neither 0 nor ZERO WIDTH SPACE are available\\%
3982           in the current font, and therefore the hyphen\\%
3983           will be printed. Try changing the fontspec's\\%
3984           'HyphenChar' to another value, but be aware\\%
3985           this setting is not safe (see the manual)}%
3986        \hyphenchar\font\defaultshyphenchar
3987      \fi\fi
3988     \fi}%
3989     {\hyphenchar\font\defaultshyphenchar}}
3990 % \fi}

```

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

```

3991 \def\bbl@load@info#1{%
3992   \def\BabelBeforeIni##1##2{%
3993     \begingroup
3994       \bbl@read@ini{##1}0%
3995       \endinput          % babel- .tex may contain onlypreamble's
3996       \endgroup}%        boxed, to avoid extra spaces:
3997   {\bbl@input@texini{#1}}

```

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.

[illegible]

```

4029 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
4030   \ifx\\#1%           % \\ before, in case #1 is multiletter
4031     \bbl@exp{%
4032       \def\\ \bbl@tempa####1{%
4033         \<ifcase>####1\space\the\toks@\<else>\\ \ctrerr\<fi>}}%
4034     \else
4035       \toks@\expandafter{\the\toks@\or #1}%
4036       \expandafter\bbl@buildifcase
4037   \fi}

```

```

4038 \newcommand\localexnumeral[2]{\bbl@cs{cntr@#1\@language}{#2}}
4039 \def\bbl@localexcntr#1#2{\localexnumeral{#2}{#1}}
4040 \newcommand\localecounter[2]{%
4041   \expandafter\bbl@localexcntr
4042   \expandafter{\number\csname c@#2\endcsname}{#1}}
4043 \def\bbl@alphnumeral#1#2{%
4044   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
4045 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
4046   \ifcase#9\car#8\@nilor % Currently 10000, but prepared for bigger
4047   \bbl@alphnumeral@i{#9}000000#1\or

```

```

4048 \bbl@alphanumeric@ii{#9}0000#1#2\or
4049 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
4050 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
4051 \bbl@alphnum@invalid{>9999}%
4052 \fi}
4053 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
4054 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
4055 {\bbl@cs{cntr@#1.4@\language}#5%
4056 \bbl@cs{cntr@#1.3@\language}#6%
4057 \bbl@cs{cntr@#1.2@\language}#7%
4058 \bbl@cs{cntr@#1.1@\language}#8%
4059 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4060 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
4061 {\bbl@cs{cntr@#1.S.321@\language}}%
4062 \fi}%
4063 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
4064 \def\bbl@alphnum@invalid#1{%
4065 \bbl@error{Alphabetic numeral too large (#1)}%
4066 {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.

```

4067 \newcommand\localeinfo[1]{%
4068 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
4069 {\bbl@error{I've found no info for the current locale.\%
4070 The corresponding ini file has not been loaded\%
4071 Perhaps it doesn't exist}%
4072 {See the manual for details.}}%
4073 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}%
4074 % \@namedef{bbl@info@name.locale}{lcname}
4075 \@namedef{bbl@info@tag.ini}{lini}
4076 \@namedef{bbl@info@name.english}{elname}
4077 \@namedef{bbl@info@name.opentype}{lname}
4078 \@namedef{bbl@info@tag.bcp47}{tbc}
4079 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
4080 \@namedef{bbl@info@tag.opentype}{lotf}
4081 \@namedef{bbl@info@script.name}{esname}
4082 \@namedef{bbl@info@script.name.opentype}{sname}
4083 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4084 \@namedef{bbl@info@script.tag.opentype}{sotf}
4085 \let\bbl@ensureinfo\@gobble
4086 \newcommand\BabelEnsureInfo{%
4087 \ifx\InputIfFileExists\undefined\else
4088 \def\bbl@ensureinfo##1{%
4089 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
4090 \fi
4091 \bbl@foreach\bbl@loaded{%
4092 \def\language{##1}%
4093 \bbl@ensureinfo{##1}}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

4094 \newcommand\getlocaleproperty{%
4095 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4096 \def\bbl@getproperty@s#1#2#3{%
4097 \let#1\relax
4098 \def\bbl@elt##1##2##3{%
4099 \bbl@ifsamestring{##1/##2}{#3}%

```

```

4100      {\providecommand#1{##3}%
4101      \def\bbl@elt####1####2####3{}}%
4102      {}}%
4103      \bbl@cs{inidata@#2}}%
4104 \def\bbl@getproperty@x#1#2#3{%
4105   \bbl@getproperty@s{#1}{#2}{#3}%
4106   \ifx#1\relax
4107     \bbl@error
4108     {Unknown key for locale '#2':\%
4109     #3\%
4110     \string#1 will be set to \relax}%
4111     {Perhaps you misspelled it.}%
4112   \fi}
4113 \let\bbl@ini@loaded\@empty
4114 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel bahavior

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

```

4115 \newcommand\babeladjust[1]{% TODO. Error handling.
4116   \bbl@forkv{#1}{%
4117     \bbl@ifunset{\bbl@ADJ@##1@##2}%
4118     {\bbl@cs{ADJ@##1}{##2}}%
4119     {\bbl@cs{ADJ@##1@##2}}}
4120 %
4121 \def\bbl@adjust@lua#1#2{%
4122   \ifvmode
4123     \ifnum\currentgrouplevel=\z@
4124       \directlua{ Babel.#2 }%
4125       \expandafter\expandafter\expandafter\@gobble
4126     \fi
4127   \fi
4128   {\bbl@error % The error is gobbled if everything went ok.
4129     {Currently, #1 related features can be adjusted only\%
4130     in the main vertical list.%
4131     {Maybe things change in the future, but this is what it is.}}}
4132 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
4133   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4134 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
4135   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4136 \@namedef{\bbl@ADJ@bidi.text@on}{%
4137   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4138 \@namedef{\bbl@ADJ@bidi.text@off}{%
4139   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4140 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
4141   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4142 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
4143   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4144 %
4145 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
4146   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4147 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
4148   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4149 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
4150   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4151 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
4152   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4153 \@namedef{\bbl@ADJ@justify.arabic@on}{%

```

```

4154 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4155 \@namedef{bbl@ADJ@justify.arabic@off}{%
4156 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4157 %
4158 \def\bbl@adjust@layout#1{%
4159 \ifvmode
4160 #1%
4161 \expandafter\@gobble
4162 \fi
4163 {\bbl@error % The error is gobbled if everything went ok.
4164 {Currently, layout related features can be adjusted only\%
4165 in vertical mode.}%
4166 {Maybe things change in the future, but this is what it is.}}}
4167 \@namedef{bbl@ADJ@layout.tabular@on}{%
4168 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4169 \@namedef{bbl@ADJ@layout.tabular@off}{%
4170 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4171 \@namedef{bbl@ADJ@layout.lists@on}{%
4172 \bbl@adjust@layout{\let\list\bbl@NL@list}}
4173 \@namedef{bbl@ADJ@layout.lists@off}{%
4174 \bbl@adjust@layout{\let\list\bbl@OL@list}}
4175 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4176 \bbl@activateposthyphen}
4177 %
4178 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4179 \bbl@bcppallowedtrue}
4180 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4181 \bbl@bcppallowedfalse}
4182 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4183 \def\bbl@bcp@prefix{#1}}
4184 \def\bbl@bcp@prefix{bcp47-}
4185 \@namedef{bbl@ADJ@autoload.options}#1{%
4186 \def\bbl@autoload@options{#1}}
4187 \let\bbl@autoload@bcptoptions\@empty
4188 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4189 \def\bbl@autoload@bcptoptions{#1}}
4190 \newif\ifbbl@bcptname
4191 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4192 \bbl@bcptnametrue}
4193 \BabelEnsureInfo}
4194 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4195 \bbl@bcptnamefalse}
4196 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4197 \directlua{ Babel.ignore_pre_char = function(node)
4198 return (node.lang == \the\csname l@nohyphenation\endcsname)
4199 end }}
4200 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4201 \directlua{ Babel.ignore_pre_char = function(node)
4202 return false
4203 end }}

As the final task, load the code for lua. TODO: use babel name, override

4204 \ifx\directlua\@undefined\else
4205 \ifx\bbl@luapatterns\@undefined
4206 \input luababel.def
4207 \fi
4208 \fi
4209 </core>

```

```

A proxy file for switch.def
4210 <*kernel>
4211 \let\bbl@onlyswitch\@empty
4212 \input babel.def
4213 \let\bbl@onlyswitch\@undefined
4214 </kernel>
4215 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by $\text{\texttt{initex}}$ because it should instruct $\text{\texttt{TeX}}$ to read hyphenation patterns. To this end the `docstrip` option `patterns` is used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

```

4216 <<Make sure ProvidesFile is defined>>
4217 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
4218 \xdef\bbl@format{\jobname}
4219 \def\bbl@version{\<version>}
4220 \def\bbl@date{\<date>}
4221 \ifx\AtBeginDocument\@undefined
4222   \def\@empty{}
4223 \fi
4224 <<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.

```

4225 \def\process@line#1#2 #3 #4 {%
4226   \ifx=#1%
4227     \process@synonym{#2}%
4228   \else
4229     \process@language{#1#2}{#3}{#4}%
4230   \fi
4231   \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.

```

4232 \toks@{}
4233 \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.

```

4234 \def\process@synonym#1{%
4235   \ifnum\last@language=\m@ne
4236     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4237   \else
4238     \expandafter\chardef\csname l@#1\endcsname\last@language
4239     \wlog{\string\l@#1=\string\language\the\last@language}%
4240     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4241       \csname\language\hyphenmins\endcsname
4242     \let\bbl@elt\relax
4243     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4244   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\langle lang \rangle hyphenmins` macro. When no assignments were made we provide a default setting. Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form `\bbl@elt{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

4245 \def\process@language#1#2#3{%
4246   \expandafter\addlanguage\csname l@#1\endcsname
4247   \expandafter\language\csname l@#1\endcsname
4248   \edef\language#1}%
4249   \bbl@hook@everylanguage{#1}%
4250   % > luatex
4251   \bbl@get@enc#1::@@@
4252   \begingroup
4253     \lefthyphenmin\m@ne
4254     \bbl@hook@loadpatterns{#2}%
4255     % > luatex
4256     \ifnum\lefthyphenmin=\m@ne
4257     \else
4258       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4259         \the\lefthyphenmin\the\righthyphenmin}%
4260     \fi
4261   \endgroup
4262   \def\bbl@tempa{#3}%
4263   \ifx\bbl@tempa\@empty\else
4264     \bbl@hook@loadexceptions{#3}%
4265     % > luatex
4266   \fi
4267   \let\bbl@elt\relax
4268   \edef\bbl@languages{%
4269     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4270   \ifnum\the\language=\z@
4271     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4272       \set@hyphenmins\tw@\thr@@\relax
4273     \else
4274       \expandafter\expandafter\expandafter\set@hyphenmins
4275       \csname #1hyphenmins\endcsname

```

```

4276 \fi
4277 \the\toks@
4278 \toks@{}}%
4279 \fi}

```

\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it in
\bbl@hyph@enc \bbl@hyph@enc. It uses delimited arguments to achieve this.

```

4280 \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.

```

4281 \def\bbl@hook@everylanguage#1{}
4282 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4283 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4284 \def\bbl@hook@loadkernel#1{%
4285 \def\addlanguage{\csname newlanguage\endcsname}%
4286 \def\adddialect##1##2{%
4287 \global\chardef##1##2\relax
4288 \wlog{\string##1 = a dialect from \string\language##2}}%
4289 \def\iflanguage#1{%
4290 \expandafter\ifx\csname l@##1\endcsname\relax
4291 \@nolanerr{##1}%
4292 \else
4293 \ifnum\csname l@##1\endcsname=\language
4294 \expandafter\expandafter\expandafter\@firstoftwo
4295 \else
4296 \expandafter\expandafter\expandafter\@secondoftwo
4297 \fi
4298 \fi}%
4299 \def\providehyphenmins##1##2{%
4300 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4301 \@namedef{##1hyphenmins}{##2}%
4302 \fi}%
4303 \def\set@hyphenmins##1##2{%
4304 \lefthyphenmin##1\relax
4305 \righthyphenmin##2\relax}%
4306 \def\selectlanguage{%
4307 \errhelp{Selecting a language requires a package supporting it}%
4308 \errmessage{Not loaded}}%
4309 \let\foreignlanguage\selectlanguage
4310 \let\otherlanguage\selectlanguage
4311 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4312 \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4313 \def\setlocale{%
4314 \errhelp{Find an armchair, sit down and wait}%
4315 \errmessage{Not yet available}}%
4316 \let\uselocale\setlocale
4317 \let\locale\setlocale
4318 \let\selectlocale\setlocale
4319 \let\localename\setlocale
4320 \let\textlocale\setlocale
4321 \let\textlanguage\setlocale
4322 \let\languagetext\setlocale}
4323 \begingroup
4324 \def\AddBabelHook#1#2{%
4325 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4326 \def\next{\toks1}%

```



```

4327 \else
4328 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4329 \fi
4330 \next}
4331 \ifx\directlua\@undefined
4332 \ifx\XeTeXinputencoding\@undefined\else
4333 \input xebabel.def
4334 \fi
4335 \else
4336 \input luababel.def
4337 \fi
4338 \openin1 = babel-\bbl@format.cfg
4339 \ifeof1
4340 \else
4341 \input babel-\bbl@format.cfg\relax
4342 \fi
4343 \closein1
4344 \endgroup
4345 \bbl@hook@loadkernel{switch.def}

```

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

```

4346 \openin1 = language.dat

```

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

```

4347 \def\language{english}%
4348 \ifeof1
4349 \message{I couldn't find the file language.dat,\space
4350 I will try the file hyphen.tex}
4351 \input hyphen.tex\relax
4352 \chardef\l@english\z@
4353 \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 .

```

4354 \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.

```

4355 \loop
4356 \endlinechar\m@ne
4357 \read1 to \bbl@line
4358 \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.

```

4359 \if T\ifeof1F\fi T\relax
4360 \ifx\bbl@line\@empty\else
4361 \edef\bbl@line{\bbl@line\space\space\space}%
4362 \expandafter\process@line\bbl@line\relax
4363 \fi
4364 \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.

```

4365 \begingroup
4366   \def\bbl@elt#1#2#3#4{%
4367     \global\language=#2\relax
4368     \gdef\language#1}%
4369   \def\bbl@elt##1##2##3##4{}}%
4370   \bbl@languages
4371 \endgroup
4372 \fi
4373 \closein1

```

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

```

4374 \if/\the\toks@/\else
4375   \errhelp{language.dat loads no language, only synonyms}
4376   \errmessage{Orphan language synonym}
4377 \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.

```

4378 \let\bbl@line\@undefined
4379 \let\process@line\@undefined
4380 \let\process@synonym\@undefined
4381 \let\process@language\@undefined
4382 \let\bbl@get@enc\@undefined
4383 \let\bbl@hyph@enc\@undefined
4384 \let\bbl@tempa\@undefined
4385 \let\bbl@hook@loadkernel\@undefined
4386 \let\bbl@hook@everylanguage\@undefined
4387 \let\bbl@hook@loadpatterns\@undefined
4388 \let\bbl@hook@loadexceptions\@undefined
4389 \let\patterns\@undefined

```

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].

```

4390 <<*More package options>> ≡
4391 \chardef\bbl@bidimode\z@
4392 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4393 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4394 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4395 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4396 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4397 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4398 <</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`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading message, which is replaced by a more explanatory one.

```

4399 <<*Font selection>> ≡
4400 \bbl@trace{Font handling with fontspec}
4401 \ifx\ExplSyntaxOn\@undefined\else
4402   \ExplSyntaxOn
4403   \catcode\ =10

```

```

4404 \def\bbl@loadfontspec{%
4405   \usepackage{fontspec}% TODO. Apply patch always
4406   \expandafter
4407   \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4408     Font '\l_fontspec_fontname_tl' is using the\\%
4409     default features for language '##1'.\\%
4410     That's usually fine, because many languages\\%
4411     require no specific features, but if the output is\\%
4412     not as expected, consider selecting another font.}
4413   \expandafter
4414   \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4415     Font '\l_fontspec_fontname_tl' is using the\\%
4416     default features for script '##2'.\\%
4417     That's not always wrong, but if the output is\\%
4418     not as expected, consider selecting another font.}}
4419   \ExplSyntaxOff
4420 \fi
4421 \@onlypreamble\babelfont
4422 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4423   \bbl@foreach{#1}{%
4424     \expandafter\ifx\csname date##1\endcsname\relax
4425       \IfFileExists{babel-##1.tex}%
4426         {\babelprovide{##1}}%
4427       {}%
4428     \fi}%
4429   \edef\bbl@tempa{#1}%
4430   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4431   \ifx\fontspec@undefined
4432     \bbl@loadfontspec
4433   \fi
4434   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4435   \bbl@bblfont}
4436 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4437   \bbl@ifunset{\bbl@tempb family}%
4438     {\bbl@providedefam{\bbl@tempb}}%
4439     {}%
4440   % For the default font, just in case:
4441   \bbl@ifunset{\bbl@lsys@languagename}{\bbl@provide@lsys{languagename}}{}%
4442   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4443   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4444     \bbl@exp{%
4445       \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4446       \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4447       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4448   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4449     \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}}%

```

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

```

4450 \def\bbl@providedefam#1{%
4451   \bbl@exp{%
4452     \\\newcommand<#1default>{}% Just define it
4453     \\\bbl@add@list\\bbl@font@fams{#1}%
4454     \\\DeclareRobustCommand<#1family>{%
4455       \\\not@math@alphabet<#1family>\relax
4456       % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4457       \\\fontfamily<#1default>%
4458       \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4459       \\\selectfont}%
4460     \\\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.

```

4461 \def\bbl@nostdfont#1{%
4462   \bbl@ifunset{bbl@WFF@\f@family}%
4463   {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
4464   \bbl@infowarn{The current font is not a babel standard family:\%
4465     #1%
4466     \fontname\font\%
4467     There is nothing intrinsically wrong with this warning, and\%
4468     you can ignore it altogether if you do not need these\%
4469     families. But if they are used in the document, you should be\%
4470     aware 'babel' will no set Script and Language for them, so\%
4471     you may consider defining a new family with \string\babelfont.\%
4472     See the manual for further details about \string\babelfont.\%
4473     Reported}}
4474   {}}%
4475 \gdef\bbl@switchfont{%
4476   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4477   \bbl@exp{% eg Arabic -> arabic
4478   \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}}%
4479   \bbl@foreach\bbl@font@fams{%
4480     \bbl@ifunset{bbl@##1dflt@\languagename}% (1) language?
4481     {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4482     {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4483     {}}% 123=F - nothing!
4484     {\bbl@exp{% 3=T - from generic
4485       \global\let<bbl@##1dflt@\languagename>%
4486       \<bbl@##1dflt@>}}}%
4487     {\bbl@exp{% 2=T - from script
4488       \global\let<bbl@##1dflt@\languagename>%
4489       \<bbl@##1dflt@*\bbl@tempa>}}}%
4490     {}}% 1=T - language, already defined
4491   \def\bbl@tempa{\bbl@nostdfont{}}%
4492   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4493     \bbl@ifunset{bbl@##1dflt@\languagename}%
4494     {\bbl@cs{famrst@##1}%
4495     \global\bbl@csarg\let{famrst@##1}\relax}%
4496     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4497     \\\bbl@add\\originalTeX{%
4498     \\\bbl@font@rst{\bbl@cl{##1dflt}}}%
4499     \<##1default>\<##1family>{##1}}}%
4500     \\\bbl@font@set<bbl@##1dflt@\languagename>% the main part!
4501     \<##1default>\<##1family>}}}%
4502   \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.

```

4503 \ifx\f@family\undefined\else % if latex
4504 \ifcase\bbl@engine % if pdftex
4505 \let\bbl@ckeckstdfonts\relax
4506 \else
4507 \def\bbl@ckeckstdfonts{%
4508   \begingroup
4509   \global\let\bbl@ckeckstdfonts\relax
4510   \let\bbl@tempa\empty
4511   \bbl@foreach\bbl@font@fams{%
4512     \bbl@ifunset{bbl@##1dflt@}%
4513     {\nameuse{##1family}%
4514     \bbl@csarg\gdef{WFF@\f@family}}}% Flag

```

```

4515         \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \f@family\\}%
4516         \space\space\fontname\font\\}%
4517         \bbl@csarg\edef{##1dflt@}{\f@family}%
4518         \expandafter\edef\csname ##1default\endcsname{\f@family}%
4519         }%
4520     \ifx\bbl@tempa\@empty\else
4521         \bbl@infowarn{The following font families will use the default\\%
4522         settings for all or some languages:\\%
4523         \bbl@tempa
4524         There is nothing intrinsically wrong with it, but\\%
4525         'babel' will no set Script and Language, which could\\%
4526         be relevant in some languages. If your document uses\\%
4527         these families, consider redefining them with \string\babelfont.\\%
4528         Reported}%
4529     \fi
4530 \endgroup}
4531 \fi
4532 \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.

```

4533 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4534 \bbl@xin@{<>}{#1}%
4535 \ifin@
4536 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4537 \fi
4538 \bbl@exp{%
4539     \def\#2#1% eg, \rmdefault{\bbl@rmdflt@lang}
4540     \bbl@ifsamestring{#2}{\f@family}%
4541     {\#3%
4542     \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}%
4543     \let\bbl@tempa\relax}%
4544     {}}
4545 % TODO - next should be global?, but even local does its job. I'm
4546 % still not sure -- must investigate:
4547 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4548 \let\bbl@tempe\bbl@mapselect
4549 \let\bbl@mapselect\relax
4550 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4551 \let#4\@empty % Make sure \renewfontfamily is valid
4552 \bbl@exp{%
4553     \let\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4554     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4555     {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4556     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4557     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4558     \renewfontfamily\#4%
4559     [\bbl@cs{lsys@languagename},#2]{#3}% ie \bbl@exp{.}{#3}
4560 \begingroup
4561     #4%
4562     \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4563 \endgroup
4564 \let#4\bbl@temp@fam
4565 \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4566 \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.

```
4567 \def\bbbl@font@rst#1#2#3#4{%
4568   \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`.

```
4569 \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 is explained in the user guide, but essentially – that was not the way to go :-).

```
4570 \newcommand\babelFSstore[2][{%
4571   \bbbl@ifblank{#1}%
4572   {\bbbl@csarg\def{sname@#2}{Latin}}%
4573   {\bbbl@csarg\def{sname@#2}{#1}}%
4574   \bbbl@provide@dirs{#2}%
4575   \bbbl@csarg\ifnum{wdir@#2}>\z@
4576     \let\bbbl@beforeforeign\leavevmode
4577     \EnableBabelHook{babel-bidi}%
4578   \fi
4579   \bbbl@foreach{#2}{%
4580     \bbbl@FSstore{##1}{rm}\rmdefault\bbbl@save@rmdefault
4581     \bbbl@FSstore{##1}{sf}\sfdefault\bbbl@save@sfdefault
4582     \bbbl@FSstore{##1}{tt}\ttdefault\bbbl@save@ttdefault}}
4583 \def\bbbl@FSstore#1#2#3#4{%
4584   \bbbl@csarg\edef{#2default#1}{#3}%
4585   \expandafter\addto\csname extras#1\endcsname{%
4586     \let#4#3%
4587     \ifx#3\f@family
4588       \edef#3{\csname bbl@#2default#1\endcsname}%
4589       \fontfamily{#3}\selectfont
4590     \else
4591       \edef#3{\csname bbl@#2default#1\endcsname}%
4592       \fi}%
4593   \expandafter\addto\csname noextras#1\endcsname{%
4594     \ifx#3\f@family
4595       \fontfamily{#4}\selectfont
4596       \fi
4597     \let#3#4}}
4598 \let\bbbl@langfeatures\@empty
4599 \def\babelFSfeatures{% make sure \fontspec is redefined once
4600   \let\bbbl@ori@fontspec\fontspec
4601   \renewcommand\fontspec[1][{%
4602     \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
4603   \let\babelFSfeatures\bbbl@FSfeatures
4604   \babelFSfeatures}
4605 \def\bbbl@FSfeatures#1#2{%
4606   \expandafter\addto\csname extras#1\endcsname{%
4607     \babel@save\bbbl@langfeatures
4608     \edef\bbbl@langfeatures{#2,}}
4609 \<</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.

```
4610 \<<*Footnote changes>> ≡
```

```

4611 \bbl@trace{Bidi footnotes}
4612 \ifnum\bbl@bidimode>\z@
4613 \def\bbl@footnote#1#2#3{%
4614   \@ifnextchar[%
4615     {\bbl@footnote@o{#1}{#2}{#3}}%
4616     {\bbl@footnote@x{#1}{#2}{#3}}}
4617 \long\def\bbl@footnote@x#1#2#3#4{%
4618   \bgroup
4619     \select@language@x{\bbl@main@language}%
4620     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4621   \egroup}
4622 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4623   \bgroup
4624     \select@language@x{\bbl@main@language}%
4625     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4626   \egroup}
4627 \def\bbl@footnotetext#1#2#3{%
4628   \@ifnextchar[%
4629     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4630     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4631 \long\def\bbl@footnotetext@x#1#2#3#4{%
4632   \bgroup
4633     \select@language@x{\bbl@main@language}%
4634     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4635   \egroup}
4636 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4637   \bgroup
4638     \select@language@x{\bbl@main@language}%
4639     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4640   \egroup}
4641 \def\BabelFootnote#1#2#3#4{%
4642   \ifx\bbl@fn@footnote\undefined
4643     \let\bbl@fn@footnote\footnote
4644   \fi
4645   \ifx\bbl@fn@footnotetext\undefined
4646     \let\bbl@fn@footnotetext\footnotetext
4647   \fi
4648   \bbl@ifblank{#2}%
4649     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4650     \@namedef{\bbl@stripslash#1text}%
4651     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4652     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4653     \@namedef{\bbl@stripslash#1text}%
4654     {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4655 \fi
4656 <</Footnote changes>>

```

Now, the code.

```

4657 (*xetex)
4658 \def\BabelStringsDefault{unicode}
4659 \let\xebbl@stop\relax
4660 \AddBabelHook{xetex}{encodedcommands}{%
4661   \def\bbl@tempa{#1}%
4662   \ifx\bbl@tempa\empty
4663     \XeTeXinputencoding"bytes"%
4664   \else
4665     \XeTeXinputencoding"#1"%
4666   \fi
4667 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}

```

```

4668 \AddBabelHook{xetex}{stopcommands}{%
4669   \xebbl@stop
4670   \let\xebbl@stop\relax}
4671 \def\bbl@intraspace#1 #2 #3\@@{%
4672   \bbl@csarg\gdef{xeisp@\language}%
4673     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4674 \def\bbl@intrapenalty#1\@@{%
4675   \bbl@csarg\gdef{xeipn@\language}%
4676     {\XeTeXlinebreakpenalty #1\relax}}
4677 \def\bbl@provide@intraspace{%
4678   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4679   \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4680   \ifin@
4681     \bbl@ifunset{bbl@intsp@\language}{}%
4682     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4683       \ifx\bbl@KVP@intraspace\@nil
4684         \bbl@exp{%
4685           \\bbl@intraspace\bbl@cl{intsp}\\@@}%
4686         \fi
4687         \ifx\bbl@KVP@intrapenalty\@nil
4688           \bbl@intrapenalty0\@@
4689         \fi
4690       \fi
4691       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4692         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4693       \fi
4694       \ifx\bbl@KVP@intrapenalty\@nil\else
4695         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4696       \fi
4697       \bbl@exp{%
4698         % TODO. Execute only once (but redundant):
4699         \\bbl@add\<extras\language>{%
4700           \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4701           \<bbl@xeisp@\language>%
4702           \<bbl@xeipn@\language>%
4703           \\bbl@tglobal\<extras\language>%
4704           \\bbl@add\<noextras\language>{%
4705             \XeTeXlinebreaklocale "en"%
4706             \\bbl@tglobal\<noextras\language>}%
4707         \ifx\bbl@ispacesize\@undefined
4708           \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4709         \ifx\AtBeginDocument\@notprerr
4710           \expandafter\@secondoftwo % to execute right now
4711         \fi
4712         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4713       \fi}%
4714   \fi}
4715 \ifx\DisableBabelHook\@undefined\endinput\fi
4716 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4717 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4718 \DisableBabelHook{babel-fontspec}
4719 <<Font selection>>
4720 \input txtbabel.def
4721 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr,

typearea or titles, and geometry.

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

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdf_{tex} and xet_{ex}.

```

4722 (*texxet)
4723 \providecommand\bbl@provide@intraspace{}
4724 \bbl@trace{Redefinitions for bidi layout}
4725 \def\bbl@sspre@caption{%
4726   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}}
4727 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4728 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4729 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4730 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4731   \def\@hangfrom#1{%
4732     \setbox\@tempboxa\hbox{#1}}%
4733     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4734     \noindent\box\@tempboxa}
4735 \def\raggedright{%
4736   \let\@centercr
4737   \bbl@startskip\z@skip
4738   \@rightskip\@flushglue
4739   \bbl@endskip\@rightskip
4740   \parindent\z@
4741   \parfillskip\bbl@startskip}
4742 \def\raggedleft{%
4743   \let\@centercr
4744   \bbl@startskip\@flushglue
4745   \bbl@endskip\z@skip
4746   \parindent\z@
4747   \parfillskip\bbl@endskip}
4748 \fi
4749 \IfBabelLayout{lists}
4750   {\bbl@sreplace\list
4751     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4752     \def\bbl@listleftmargin{%
4753       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4754     \ifcase\bbl@engine
4755       \def\labelenumii{}\theenumii{}\% pdftex doesn't reverse ()
4756       \def\p@enumiii{\p@enumii}\theenumii{}\%
4757     \fi
4758     \bbl@sreplace\@verbatim
4759       {\leftskip\@totalleftmargin}%
4760       {\bbl@startskip\textwidth
4761         \advance\bbl@startskip-\linewidth}%
4762     \bbl@sreplace\@verbatim
4763       {\rightskip\z@skip}%
4764       {\bbl@endskip\z@skip}}%
4765   {}
4766 \IfBabelLayout{contents}
4767   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4768     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4769   {}
4770 \IfBabelLayout{columns}
4771   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outpuhbox}%
4772     \def\bbl@outpuhbox#1{%
4773       \hb@xt@\textwidth{%
4774         \hskip\columnwidth

```

```

4775 \hfil
4776 {\normalcolor\vrule \@width\columnseprule}%
4777 \hfil
4778 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4779 \hskip-\textwidth
4780 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4781 \hskip\columnsep
4782 \hskip\columnwidth}}}%
4783 {}
4784 <<Footnote changes>>
4785 \IfBabelLayout{footnotes}%
4786 {\BabelFootnote\footnote\language\language{}{}}%
4787 \BabelFootnote\localfootnote\language\language{}{}}%
4788 \BabelFootnote\mainfootnote{}{}}{}
4789 {}

```

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

```

4790 \IfBabelLayout{counters}%
4791 {\let\bbl@latinarabic=\@arabic
4792 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4793 \let\bbl@asciroman=\@roman
4794 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4795 \let\bbl@asciiRoman=\@Roman
4796 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4797 </texet>

```

13.3 LuaTeX

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

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

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, 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 file 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).

```

4798 (*luatex)
4799 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4800 \bbl@trace{Read language.dat}
4801 \ifx\bbl@readstream\undefined
4802   \csname newread\endcsname\bbl@readstream
4803 \fi
4804 \begingroup
4805   \toks@{}
4806   \count@\z@ % 0=start, 1=0th, 2=normal
4807   \def\bbl@process@line#1#2 #3 #4 {%
4808     \ifx=#1%
4809       \bbl@process@synonym{#2}%
4810     \else
4811       \bbl@process@language{#1#2}{#3}{#4}%
4812     \fi
4813     \ignorespaces}
4814   \def\bbl@manylang{%
4815     \ifnum\bbl@last>\@ne
4816       \bbl@info{Non-standard hyphenation setup}%
4817     \fi
4818     \let\bbl@manylang\relax}
4819   \def\bbl@process@language#1#2#3{%
4820     \ifcase\count@
4821       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4822     \or
4823       \count@\tw@
4824     \fi
4825     \ifnum\count@=\tw@
4826       \expandafter\addlanguage\csname l@#1\endcsname
4827       \language\allocationnumber
4828       \chardef\bbl@last\allocationnumber
4829       \bbl@manylang
4830       \let\bbl@elt\relax
4831       \xdef\bbl@languages{%
4832         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4833     \fi
4834     \the\toks@
4835     \toks@{}}
4836   \def\bbl@process@synonym@aux#1#2{%
4837     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4838     \let\bbl@elt\relax
4839     \xdef\bbl@languages{%
4840       \bbl@languages\bbl@elt{#1}{#2}{}}}%
4841   \def\bbl@process@synonym#1{%
4842     \ifcase\count@
4843       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4844     \or
4845       \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4846     \else
4847       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4848     \fi}
4849   \ifx\bbl@languages\undefined % Just a (sensible?) guess
4850     \chardef\l@english\z@
4851     \chardef\l@USenglish\z@
4852     \chardef\bbl@last\z@
4853     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}

```

```

4854 \gdef\bbl@languages{%
4855     \bbl@elt{english}{0}{hyphen.tex}}}%
4856     \bbl@elt{USenglish}{0}{}{}}
4857 \else
4858     \global\let\bbl@languages@format\bbl@languages
4859     \def\bbl@elt#1#2#3#4{% Remove all except language 0
4860         \ifnum#2>\z@\else
4861             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4862         \fi}%
4863     \xdef\bbl@languages{\bbl@languages}%
4864 \fi
4865 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{} } % Define flags
4866 \bbl@languages
4867 \openin\bbl@readstream=language.dat
4868 \ifeof\bbl@readstream
4869     \bbl@warning{I couldn't find language.dat. No additional\\%
4870                 patterns loaded. Reported}%
4871 \else
4872     \loop
4873         \endlinechar\m@ne
4874         \read\bbl@readstream to \bbl@line
4875         \endlinechar\^^M
4876         \if T\ifeof\bbl@readstream F\fi T\relax
4877         \ifx\bbl@line\empty\else
4878             \edef\bbl@line{\bbl@line\space\space\space}%
4879             \expandafter\bbl@process@line\bbl@line\relax
4880         \fi
4881     \repeat
4882 \fi
4883 \endgroup
4884 \bbl@trace{Macros for reading patterns files}
4885 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
4886 \ifx\babelcatcodetablenum\undefined
4887     \ifx\newcatcodetable\undefined
4888         \def\babelcatcodetablenum{5211}
4889         \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4890     \else
4891         \newcatcodetable\babelcatcodetablenum
4892         \newcatcodetable\bbl@pattcodes
4893     \fi
4894 \else
4895     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4896 \fi
4897 \def\bbl@luapatterns#1#2{%
4898     \bbl@get@enc#1::@@@
4899     \setbox\z@\hbox\bgroup
4900         \begingroup
4901             \savecatcodetable\babelcatcodetablenum\relax
4902             \initcatcodetable\bbl@pattcodes\relax
4903             \catcodetable\bbl@pattcodes\relax
4904             \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4905             \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\-=13
4906             \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4907             \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4908             \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4909             \catcode\`=12 \catcode\`=12 \catcode\`=12
4910             \input #1\relax
4911             \catcodetable\babelcatcodetablenum\relax
4912         \endgroup

```

```

4913 \def\bbl@tempa{#2}%
4914 \ifx\bbl@tempa@empty\else
4915 \input #2\relax
4916 \fi
4917 \egroup}%
4918 \def\bbl@patterns@lua#1{%
4919 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4920 \csname l@#1\endcsname
4921 \edef\bbl@tempa{#1}%
4922 \else
4923 \csname l@#1:\f@encoding\endcsname
4924 \edef\bbl@tempa{#1:\f@encoding}%
4925 \fi\relax
4926 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4927 \@ifundefined{bbl@hyphendata@the\language}%
4928 {\def\bbl@elt##1##2##3##4{%
4929 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4930 \def\bbl@tempb{##3}%
4931 \ifx\bbl@tempb@empty\else % if not a synonymous
4932 \def\bbl@tempc{{##3}{##4}}%
4933 \fi
4934 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4935 \fi}%
4936 \bbl@languages
4937 \@ifundefined{bbl@hyphendata@the\language}%
4938 {\bbl@info{No hyphenation patterns were set for\%
4939 language '\bbl@tempa'. Reported}}%
4940 {\expandafter\expandafter\expandafter\bbl@luapatterns
4941 \csname bbl@hyphendata@the\language\endcsname}}}%
4942 \endinput\fi
4943 % Here ends \ifx\AddBabelHook\@undefined
4944 % A few lines are only read by hyphen.cfg
4945 \ifx\DisableBabelHook\@undefined
4946 \AddBabelHook{luatex}{everylanguage}{%
4947 \def\process@language##1##2##3{%
4948 \def\process@line####1####2 ####3 ####4 {}}}
4949 \AddBabelHook{luatex}{loadpatterns}{%
4950 \input #1\relax
4951 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4952 {{#1}}}%
4953 \AddBabelHook{luatex}{loadexceptions}{%
4954 \input #1\relax
4955 \def\bbl@tempb##1##2{{##1}{##2}}%
4956 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4957 {\expandafter\expandafter\expandafter\bbl@tempb
4958 \csname bbl@hyphendata@the\language\endcsname}}
4959 \endinput\fi
4960 % Here stops reading code for hyphen.cfg
4961 % The following is read the 2nd time it's loaded
4962 \begingroup % TODO - to a lua file
4963 \catcode`\%=12
4964 \catcode`\'=12
4965 \catcode`\%=12
4966 \catcode`\:=12
4967 \directlua{
4968 Babel = Babel or {}
4969 function Babel.bytes(line)
4970 return line:gsub(".",
4971 function (chr) return unicode.utf8.char(string.byte(chr)) end)

```

```

4972 end
4973 function Babel.begin_process_input()
4974   if luatexbase and luatexbase.add_to_callback then
4975     luatexbase.add_to_callback('process_input_buffer',
4976                               Babel.bytes, 'Babel.bytes')
4977   else
4978     Babel.callback = callback.find('process_input_buffer')
4979     callback.register('process_input_buffer', Babel.bytes)
4980   end
4981 end
4982 function Babel.end_process_input ()
4983   if luatexbase and luatexbase.remove_from_callback then
4984     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4985   else
4986     callback.register('process_input_buffer', Babel.callback)
4987   end
4988 end
4989 function Babel.addpatterns(pp, lg)
4990   local lg = lang.new(lg)
4991   local pats = lang.patterns(lg) or ''
4992   lang.clear_patterns(lg)
4993   for p in pp:gmatch('[^%s]+') do
4994     ss = ''
4995     for i in string.utfcharacters(p:gsub('%d', '')) do
4996       ss = ss .. '%d?' .. i
4997     end
4998     ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4999     ss = ss:gsub('%.%%d%?$', '%%.')
5000     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5001     if n == 0 then
5002       tex.sprint(
5003         [[\string\csname\space bbl@info\endcsname{New pattern: }
5004         .. p .. [{}]]])
5005       pats = pats .. ' ' .. p
5006     else
5007       tex.sprint(
5008         [[\string\csname\space bbl@info\endcsname{Renew pattern: }
5009         .. p .. [{}]]])
5010     end
5011   end
5012   lang.patterns(lg, pats)
5013 end
5014 }
5015 \endgroup
5016 \ifx\newattribute\@undefined\else
5017   \newattribute\bbl@attr@locale
5018   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5019   \AddBabelHook{luatex}{beforeextras}{%
5020     \setattribute\bbl@attr@locale\localeid}
5021 \fi
5022 \def\BabelStringsDefault{unicode}
5023 \let\luabbl@stop\relax
5024 \AddBabelHook{luatex}{encodedcommands}{%
5025   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5026   \ifx\bbl@tempa\bbl@tempb\else
5027     \directlua{Babel.begin_process_input()}%
5028     \def\luabbl@stop{%
5029       \directlua{Babel.end_process_input()}}%
5030   \fi}%

```

```

5031 \AddBabelHook{luatex}{stopcommands}{%
5032   \luabb1@stop
5033   \let\luabb1@stop\relax}
5034 \AddBabelHook{luatex}{patterns}{%
5035   \@ifundefined{bbl@hyphendata@the\language}%
5036     {\def\bbl@elt##1##2##3##4{%
5037       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5038       \def\bbl@tempb{##3}%
5039       \ifx\bbl@tempb\@empty\else % if not a synonymous
5040         \def\bbl@tempc{##3}{##4}}%
5041       \fi
5042       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5043     \fi}%
5044   \bbl@languages
5045   \@ifundefined{bbl@hyphendata@the\language}%
5046     {\bbl@info{No hyphenation patterns were set for\%
5047       language '#2'. Reported}}%
5048     {\expandafter\expandafter\expandafter\bbl@luapatterns
5049       \csname bbl@hyphendata@the\language\endcsname}}}%
5050 \@ifundefined{bbl@patterns@}{}%
5051 \begingroup
5052   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5053   \ifin@else
5054     \ifx\bbl@patterns@\@empty\else
5055       \directlua{ Babel.addpatterns(
5056         [[\bbl@patterns@]], \number\language) }%
5057       \fi
5058       \@ifundefined{bbl@patterns@#1}%
5059         \@empty
5060         {\directlua{ Babel.addpatterns(
5061           [[\space\csname bbl@patterns@#1\endcsname]],
5062           \number\language) }}%
5063       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5064     \fi
5065   \endgroup}%
5066 \bbl@exp{%
5067   \bbl@ifunset{bbl@prehc@\language}{}%
5068   {\bbl@ifblank{\bbl@cs{prehc@\language}}}%
5069   {\prehyphenchar=\bbl@c1{prehc}\relax}}}%

```

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

```

5070 \@onlypreamble\babelpatterns
5071 \AtEndOfPackage{%
5072   \newcommand\babelpatterns[2][\@empty]{%
5073     \ifx\bbl@patterns@\relax
5074       \let\bbl@patterns@\@empty
5075     \fi
5076     \ifx\bbl@pttnlist\@empty\else
5077       \bbl@warning{%
5078         You must not intermingle \string\selectlanguage\space and\%
5079         \string\babelpatterns\space or some patterns will not\%
5080         be taken into account. Reported}%
5081       \fi
5082     \ifx\@empty#1%
5083       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5084     \else
5085       \edef\bbl@tempb{\zap@space#1 \@empty}%

```

```

5086 \bbl@for\bbl@tempa\bbl@tempb{%
5087 \bbl@fixname\bbl@tempa
5088 \bbl@iflanguage\bbl@tempa{%
5089 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5090 \ifundefined{bbl@patterns@\bbl@tempa}%
5091 \@empty
5092 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5093 #2}}}%
5094 \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. 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. See Unicode UAX 14.

```

5095% TODO - to a lua file
5096 \directlua{
5097   Babel = Babel or {}
5098   Babel.linebreaking = Babel.linebreaking or {}
5099   Babel.linebreaking.before = {}
5100   Babel.linebreaking.after = {}
5101   Babel.locale = {} % Free to use, indexed by \localeid
5102   function Babel.linebreaking.add_before(func)
5103     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5104     table.insert(Babel.linebreaking.before, func)
5105   end
5106   function Babel.linebreaking.add_after(func)
5107     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5108     table.insert(Babel.linebreaking.after, func)
5109   end
5110 }
5111 \def\bbl@intraspace#1 #2 #3\@@{%
5112   \directlua{
5113     Babel = Babel or {}
5114     Babel.intraspaces = Babel.intraspaces or {}
5115     Babel.intraspaces['\csname bbl@sbcpr@languagename\endcsname'] = %
5116       {b = #1, p = #2, m = #3}
5117     Babel.locale_props[\the\localeid].intraspace = %
5118       {b = #1, p = #2, m = #3}
5119   }}
5120 \def\bbl@intrapenalty#1\@@{%
5121   \directlua{
5122     Babel = Babel or {}
5123     Babel.intrapenalties = Babel.intrapenalties or {}
5124     Babel.intrapenalties['\csname bbl@sbcpr@languagename\endcsname'] = #1
5125     Babel.locale_props[\the\localeid].intrapenalty = #1
5126   }}
5127 \begingroup
5128 \catcode`\%=12
5129 \catcode`\^=14
5130 \catcode`\'=12
5131 \catcode`\~=12
5132 \gdef\bbl@seaintraspace{^
5133   \let\bbl@seaintraspace\relax
5134   \directlua{
5135     Babel = Babel or {}
5136     Babel.sea_enabled = true

```



```

5137 Babel.sea_ranges = Babel.sea_ranges or {}
5138 function Babel.set_chranges (script, chrng)
5139     local c = 0
5140     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5141         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5142         c = c + 1
5143     end
5144 end
5145 function Babel.sea_disc_to_space (head)
5146     local sea_ranges = Babel.sea_ranges
5147     local last_char = nil
5148     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5149     for item in node.traverse(head) do
5150         local i = item.id
5151         if i == node.id'glyph' then
5152             last_char = item
5153         elseif i == 7 and item.subtype == 3 and last_char
5154             and last_char.char > 0x0C99 then
5155             quad = font.getfont(last_char.font).size
5156             for lg, rg in pairs(sea_ranges) do
5157                 if last_char.char > rg[1] and last_char.char < rg[2] then
5158                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyr11
5159                     local intraspace = Babel.intraspaces[lg]
5160                     local intrapenalty = Babel.intrapenalties[lg]
5161                     local n
5162                     if intrapenalty ~= 0 then
5163                         n = node.new(14, 0)      ^% penalty
5164                         n.penalty = intrapenalty
5165                         node.insert_before(head, item, n)
5166                     end
5167                     n = node.new(12, 13)      ^% (glue, spaceskip)
5168                     node.setglue(n, intraspace.b * quad,
5169                                 intraspace.p * quad,
5170                                 intraspace.m * quad)
5171                     node.insert_before(head, item, n)
5172                     node.remove(head, item)
5173                 end
5174             end
5175         end
5176     end
5177 end
5178 }^^
5179 \bbl@luahyphenate}

```

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.

```

5180 \catcode`\%=14
5181 \gdef\bbl@cjkintraspaces{%
5182     \let\bbl@cjkintraspaces\relax
5183     \directlua{
5184         Babel = Babel or {}
5185         require('babel-data-cjk.lua')

```

```

5186 Babel.cjk_enabled = true
5187 function Babel.cjk_linebreak(head)
5188     local GLYPH = node.id'glyph'
5189     local last_char = nil
5190     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5191     local last_class = nil
5192     local last_lang = nil
5193
5194     for item in node.traverse(head) do
5195         if item.id == GLYPH then
5196
5197             local lang = item.lang
5198
5199             local LOCALE = node.get_attribute(item,
5200                 Babel.attr_locale)
5201             local props = Babel.locale_props[LOCALE]
5202
5203             local class = Babel.cjk_class[item.char].c
5204
5205             if props.cjk_quotes and props.cjk_quotes[item.char] then
5206                 class = props.cjk_quotes[item.char]
5207             end
5208
5209             if class == 'cp' then class = 'cl' end % ]] as CL
5210             if class == 'id' then class = 'I' end
5211
5212             local br = 0
5213             if class and last_class and Babel.cjk_breaks[last_class][class] then
5214                 br = Babel.cjk_breaks[last_class][class]
5215             end
5216
5217             if br == 1 and props.linebreak == 'c' and
5218                 lang ~= \the\l@nohyphenation\space and
5219                 last_lang ~= \the\l@nohyphenation then
5220                 local intrapenalty = props.intrapenalty
5221                 if intrapenalty ~= 0 then
5222                     local n = node.new(14, 0)      % penalty
5223                     n.penalty = intrapenalty
5224                     node.insert_before(head, item, n)
5225                 end
5226                 local intraspace = props.intraspace
5227                 local n = node.new(12, 13)         % (glue, spaceskip)
5228                 node.setglue(n, intraspace.b * quad,
5229                     intraspace.p * quad,
5230                     intraspace.m * quad)
5231                 node.insert_before(head, item, n)
5232             end
5233
5234             if font.getfont(item.font) then
5235                 quad = font.getfont(item.font).size
5236             end
5237             last_class = class
5238             last_lang = lang
5239             else % if penalty, glue or anything else
5240                 last_class = nil
5241             end
5242         end
5243         lang.hyphenate(head)
5244     end

```

```

5245 }%
5246 \bbl@luahyphenate}
5247 \gdef\bbl@luahyphenate{%
5248 \let\bbl@luahyphenate\relax
5249 \directlua{
5250   luatexbase.add_to_callback('hyphenate',
5251     function (head, tail)
5252       if Babel.linebreaking.before then
5253         for k, func in ipairs(Babel.linebreaking.before) do
5254           func(head)
5255         end
5256       end
5257       if Babel.cjk_enabled then
5258         Babel.cjk_linebreak(head)
5259       end
5260       lang.hyphenate(head)
5261       if Babel.linebreaking.after then
5262         for k, func in ipairs(Babel.linebreaking.after) do
5263           func(head)
5264         end
5265       end
5266       if Babel.sea_enabled then
5267         Babel.sea_disc_to_space(head)
5268       end
5269     end,
5270     'Babel.hyphenate')
5271 }
5272 }
5273 \endgroup
5274 \def\bbl@provide@intraspace{%
5275   \bbl@ifunset{\bbl@intsp@language}{}%
5276   {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5277     \bbl@xin@{/c}{/\bbl@cl{lbrk}}}%
5278     \ifin@           % cjk
5279     \bbl@cjk_intraspace
5280     \directlua{
5281       Babel = Babel or {}
5282       Babel.locale_props = Babel.locale_props or {}
5283       Babel.locale_props[\the\localeid].linebreak = 'c'
5284     }%
5285     \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@@}%
5286     \ifx\bbl@KVP@intrapenalty\@nil
5287       \bbl@intrapenalty0\@@
5288     \fi
5289   \else           % sea
5290     \bbl@sea_intraspace
5291     \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@@}%
5292     \directlua{
5293       Babel = Babel or {}
5294       Babel.sea_ranges = Babel.sea_ranges or {}
5295       Babel.set_chranges('\bbl@cl{sbc}',
5296         '\bbl@cl{chrng}')
5297     }%
5298     \ifx\bbl@KVP@intrapenalty\@nil
5299       \bbl@intrapenalty0\@@
5300     \fi
5301   \fi
5302 \fi
5303 \ifx\bbl@KVP@intrapenalty\@nil\else

```

```

5304 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
5305 \fi}}

```

13.6 Arabic justification

```

5306 \ifnum\bb1@bidimode>100 \ifnum\bb1@bidimode<200
5307 \def\bb1ar@chars{%
5308 0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5309 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5310 0640,0641,0642,0643,0644,0645,0646,0647,0649}
5311 \def\bb1ar@elongated{%
5312 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5313 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5314 0649,064A}
5315 \begingroup
5316 \catcode\_ =11 \catcode`:=11
5317 \gdef\bb1ar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5318 \endgroup
5319 \gdef\bb1@arabicjust{%
5320 \let\bb1ar@arabicjust\relax
5321 \newattribute\bb1ar@kashida
5322 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bb1ar@kashida' }%
5323 \bb1ar@kashida=\z@
5324 \bb1@patchfont{{\bb1@parsejalt}}%
5325 \directlua{
5326 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5327 Babel.arabic.elong_map[\the\localeid] = {}
5328 luatexbase.add_to_callback('post_linebreak_filter',
5329 Babel.arabic.justify, 'Babel.arabic.justify')
5330 luatexbase.add_to_callback('hpack_filter',
5331 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5332 }}%
5333 % Save both node lists to make replacement. TODO. Save also widths to
5334 % make computations
5335 \def\bb1ar@fetchjalt#1#2#3#4{%
5336 \bb1@exp{\bb1@foreach{#1}}{%
5337 \bb1@ifunset{bb1ar@JE@##1}%
5338 {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5339 {\setbox\z@\hbox{^^^200d\char"@nameuse{bb1ar@JE@##1}#2}}%
5340 \directlua{%
5341 local last = nil
5342 for item in node.traverse(tex.box[0].head) do
5343 if item.id == node.id'glyph' and item.char > 0x600 and
5344 not (item.char == 0x200D) then
5345 last = item
5346 end
5347 end
5348 Babel.arabic.#3['##1#4'] = last.char
5349 }}}
5350 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5351 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5352 % positioning?
5353 \gdef\bb1@parsejalt{%
5354 \ifx\addfontfeature\undefined\else
5355 \bb1@xin@{/e}{/\bb1@cl{lnbrk}}%
5356 \fin@
5357 \directlua{%
5358 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5359 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}

```

```

5360         tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5361     end
5362 }%
5363 \fi
5364 \fi}
5365 \gdef\bbl@parsejalti{%
5366 \begingroup
5367 \let\bbl@parsejalt\relax % To avoid infinite loop
5368 \edef\bbl@tempb{\fontid\font}%
5369 \bblar@nofswarn
5370 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5371 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5372 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5373 \addfontfeature{RawFeature+=jalt}%
5374 % \namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5375 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5376 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5377 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5378 \directlua{%
5379     for k, v in pairs(Babel.arabic.from) do
5380         if Babel.arabic.dest[k] and
5381             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5382             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5383                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5384         end
5385     end
5386 }%
5387 \endgroup}
5388 %
5389 \begingroup
5390 \catcode`#=11
5391 \catcode`~=11
5392 \directlua{
5393
5394 Babel.arabic = Babel.arabic or {}
5395 Babel.arabic.from = {}
5396 Babel.arabic.dest = {}
5397 Babel.arabic.justify_factor = 0.95
5398 Babel.arabic.justify_enabled = true
5399
5400 function Babel.arabic.justify(head)
5401     if not Babel.arabic.justify_enabled then return head end
5402     for line in node.traverse_id(node.id'hlist', head) do
5403         Babel.arabic.justify_hlist(head, line)
5404     end
5405     return head
5406 end
5407
5408 function Babel.arabic.justify_hbox(head, gc, size, pack)
5409     local has_inf = false
5410     if Babel.arabic.justify_enabled and pack == 'exactly' then
5411         for n in node.traverse_id(12, head) do
5412             if n.stretch_order > 0 then has_inf = true end
5413         end
5414         if not has_inf then
5415             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5416         end
5417     end
5418     return head

```

```

5419 end
5420
5421 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5422     local d, new
5423     local k_list, k_item, pos_inline
5424     local width, width_new, full, k_curr, wt_pos, goal, shift
5425     local subst_done = false
5426     local elong_map = Babel.arabic.elong_map
5427     local last_line
5428     local GLYPH = node.id'glyph'
5429     local KASHIDA = Babel.attr_kashida
5430     local LOCALE = Babel.attr_locale
5431
5432     if line == nil then
5433         line = {}
5434         line.glue_sign = 1
5435         line.glue_order = 0
5436         line.head = head
5437         line.shift = 0
5438         line.width = size
5439     end
5440
5441     % Exclude last line. todo. But-- it discards one-word lines, too!
5442     % ? Look for glue = 12:15
5443     if (line.glue_sign == 1 and line.glue_order == 0) then
5444         elongs = {} % Stores elongated candidates of each line
5445         k_list = {} % And all letters with kashida
5446         pos_inline = 0 % Not yet used
5447
5448         for n in node.traverse_id(GLYPH, line.head) do
5449             pos_inline = pos_inline + 1 % To find where it is. Not used.
5450
5451             % Elongated glyphs
5452             if elong_map then
5453                 local locale = node.get_attribute(n, LOCALE)
5454                 if elong_map[locale] and elong_map[locale][n.font] and
5455                     elong_map[locale][n.font][n.char] then
5456                     table.insert(elongs, {node = n, locale = locale} )
5457                     node.set_attribute(n.prev, KASHIDA, 0)
5458                 end
5459             end
5460
5461             % Tatwil
5462             if Babel.kashida_wts then
5463                 local k_wt = node.get_attribute(n, KASHIDA)
5464                 if k_wt > 0 then % todo. parameter for multi inserts
5465                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5466                 end
5467             end
5468
5469         end % of node.traverse_id
5470
5471         if #elongs == 0 and #k_list == 0 then goto next_line end
5472         full = line.width
5473         shift = line.shift
5474         goal = full * Babel.arabic.justify_factor % A bit crude
5475         width = node.dimensions(line.head) % The 'natural' width
5476
5477         % == Elongated ==

```

```

5478 % Original idea taken from 'chickenize'
5479 while (#elongs > 0 and width < goal) do
5480     subst_done = true
5481     local x = #elongs
5482     local curr = elongs[x].node
5483     local oldchar = curr.char
5484     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5485     width = node.dimensions(line.head) % Check if the line is too wide
5486     % Substitute back if the line would be too wide and break:
5487     if width > goal then
5488         curr.char = oldchar
5489         break
5490     end
5491     % If continue, pop the just substituted node from the list:
5492     table.remove(elongs, x)
5493 end
5494
5495 % == Tatwil ==
5496 if #k_list == 0 then goto next_line end
5497
5498 width = node.dimensions(line.head) % The 'natural' width
5499 k_curr = #k_list
5500 wt_pos = 1
5501
5502 while width < goal do
5503     subst_done = true
5504     k_item = k_list[k_curr].node
5505     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5506         d = node.copy(k_item)
5507         d.char = 0x0640
5508         line.head, new = node.insert_after(line.head, k_item, d)
5509         width_new = node.dimensions(line.head)
5510         if width > goal or width == width_new then
5511             node.remove(line.head, new) % Better compute before
5512             break
5513         end
5514         width = width_new
5515     end
5516     if k_curr == 1 then
5517         k_curr = #k_list
5518         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5519     else
5520         k_curr = k_curr - 1
5521     end
5522 end
5523
5524 ::next_line::
5525
5526 % Must take into account marks and ins, see luatex manual.
5527 % Have to be executed only if there are changes. Investigate
5528 % what's going on exactly.
5529 if subst_done and not gc then
5530     d = node.hpack(line.head, full, 'exactly')
5531     d.shift = shift
5532     node.insert_before(head, line, d)
5533     node.remove(head, line)
5534 end
5535 end % if process line
5536 end

```

```

5537 }
5538 \endgroup
5539 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5540 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5541 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5542 \DisableBabelHook{babel-fontspec}
5543 <<Font selection>>

```

13.8 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.

```

5544 % TODO - to a lua file
5545 \directlua{
5546 Babel.script_blocks = {
5547   ['dflt'] = {},
5548   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5549             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5550   ['Armn'] = {{0x0530, 0x058F}},
5551   ['Beng'] = {{0x0980, 0x09FF}},
5552   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5553   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5554   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5555             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5556   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5557   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5558             {0xAB00, 0xAB2F}},
5559   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5560   % Don't follow strictly Unicode, which places some Coptic letters in
5561   % the 'Greek and Coptic' block
5562   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5563   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5564             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5565             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5566             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5567             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5568             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5569   ['Hebr'] = {{0x0590, 0x05FF}},
5570   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5571             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5572   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5573   ['Knda'] = {{0x0C80, 0x0CFF}},
5574   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5575             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5576             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5577   ['Lao0'] = {{0x0E80, 0x0EFF}},
5578   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5579             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5580             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5581   ['Mahj'] = {{0x11150, 0x1117F}},
5582   ['Mlym'] = {{0x0D00, 0x0D7F}},
5583   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},

```



```

5584 ['Orya'] = {{0x0B00, 0x0B7F}},
5585 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5586 ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5587 ['Taml'] = {{0x0B80, 0x0BFF}},
5588 ['Telu'] = {{0x0C00, 0x0C7F}},
5589 ['Tfng'] = {{0x2D30, 0x2D7F}},
5590 ['Thai'] = {{0x0E00, 0x0E7F}},
5591 ['Tibt'] = {{0x0F00, 0x0FFF}},
5592 ['Vaii'] = {{0xA500, 0xA63F}},
5593 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5594 }
5595
5596 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5597 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5598 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5599
5600 function Babel.locale_map(head)
5601   if not Babel.locale_mapped then return head end
5602
5603   local LOCALE = Babel.attr_locale
5604   local GLYPH = node.id('glyph')
5605   local inmath = false
5606   local toloc_save
5607   for item in node.traverse(head) do
5608     local toloc
5609     if not inmath and item.id == GLYPH then
5610       % Optimization: build a table with the chars found
5611       if Babel.chr_to_loc[item.char] then
5612         toloc = Babel.chr_to_loc[item.char]
5613       else
5614         for lc, maps in pairs(Babel.loc_to_scr) do
5615           for _, rg in pairs(maps) do
5616             if item.char >= rg[1] and item.char <= rg[2] then
5617               Babel.chr_to_loc[item.char] = lc
5618               toloc = lc
5619               break
5620             end
5621           end
5622         end
5623       end
5624       % Now, take action, but treat composite chars in a different
5625       % fashion, because they 'inherit' the previous locale. Not yet
5626       % optimized.
5627       if not toloc and
5628         (item.char >= 0x0300 and item.char <= 0x036F) or
5629         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5630         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5631         toloc = toloc_save
5632       end
5633       if toloc and toloc > -1 then
5634         if Babel.locale_props[toloc].lg then
5635           item.lang = Babel.locale_props[toloc].lg
5636           node.set_attribute(item, LOCALE, toloc)
5637         end
5638         if Babel.locale_props[toloc]['/'..item.font] then
5639           item.font = Babel.locale_props[toloc]['/'..item.font]
5640         end
5641         toloc_save = toloc
5642       end

```

```

5643 elseif not inmath and item.id == 7 then
5644   item.replace = item.replace and Babel.locale_map(item.replace)
5645   item.pre      = item.pre and Babel.locale_map(item.pre)
5646   item.post     = item.post and Babel.locale_map(item.post)
5647   elseif item.id == node.id'math' then
5648     inmath = (item.subtype == 0)
5649   end
5650 end
5651 return head
5652 end
5653 }

```

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

```

5654 \newcommand\babelcharproperty[1]{%
5655   \count@=#1\relax
5656   \ifvmode
5657     \expandafter\bbl@chprop
5658   \else
5659     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5660               vertical mode (preamble or between paragraphs)}%
5661     {See the manual for futher info}%
5662   \fi}
5663 \newcommand\bbl@chprop[3][\the\count@]{%
5664   \@tempcnta=#1\relax
5665   \bbl@ifunset{\bbl@chprop@#2}%
5666   {\bbl@error{No property named '#2'. Allowed values are\\%
5667               direction (bc), mirror (bmg), and linebreak (lb)}%
5668    {See the manual for futher info}}%
5669   }%
5670   \loop
5671     \bbl@cs{chprop@#2}{#3}%
5672   \ifnum\count@<\@tempcnta
5673     \advance\count@\@ne
5674   \repeat}
5675 \def\bbl@chprop@direction#1{%
5676   \directlua{
5677     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5678     Babel.characters[\the\count@]['d'] = '#1'
5679   }}
5680 \let\bbl@chprop@bc\bbl@chprop@direction
5681 \def\bbl@chprop@mirror#1{%
5682   \directlua{
5683     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5684     Babel.characters[\the\count@]['m'] = '\number#1'
5685   }}
5686 \let\bbl@chprop@bmg\bbl@chprop@mirror
5687 \def\bbl@chprop@linebreak#1{%
5688   \directlua{
5689     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5690     Babel.cjk_characters[\the\count@]['c'] = '#1'
5691   }}
5692 \let\bbl@chprop@lb\bbl@chprop@linebreak
5693 \def\bbl@chprop@locale#1{%
5694   \directlua{
5695     Babel.chr_to_loc = Babel.chr_to_loc or {}
5696     Babel.chr_to_loc[\the\count@] =
5697       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5698   }}

```

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). The Lua code is below.

```
5699 \directlua{
5700   Babel.nohyphenation = \the\l@nohyphenation
5701 }
```

Now the T_EX 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).

```
5702 \begingroup
5703 \catcode`\~ = 12
5704 \catcode`\% = 12
5705 \catcode`\& = 14
5706 \gdef\babelposthyphenation#1#2#3{&%
5707   \bbl@activateposthyphen
5708   \begingroup
5709     \def\babeltempa{\bbl@add@list\babeltempb}&%
5710     \let\babeltempb\@empty
5711     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5712     \bbl@replace\bbl@tempa{,}{ ,}&%
5713     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5714       \bbl@ifsamestring{##1}{remove}&%
5715       {\bbl@add@list\babeltempb{nil}}&%
5716       {\directlua{
5717         local rep = {[#1]=]
5718         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5719         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5720         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5721         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5722         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5723         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5724         tex.print([[\\string\babeltempa{[]] .. rep .. [[]]])
5725       }}}&%
5726   \directlua{
5727     local lbkr = Babel.linebreaking.replacements[1]
5728     local u = unicode.utf8
5729     local id = \the\csname l@#1\endcsname
5730     &% Convert pattern:
5731     local patt = string.gsub(=[#2]=, '%s', '')
5732     if not u.find(patt, '()', nil, true) then
5733       patt = '()' .. patt .. '()'
5734     end
5735     patt = string.gsub(patt, '%(%)%', '^()')
5736     patt = string.gsub(patt, '%$(%)', '()$')
5737     patt = u.gsub(patt, '{(.)}',
5738       function (n)
5739         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5740       end)
5741     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5742       function (n)
5743         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5744       end)
5745     lbkr[id] = lbkr[id] or {}
```

```

5746     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5747 }&%
5748 \endgroup}
5749 % TODO. Copy paste pattern.
5750 \gdef\babelprehyphenation#1#2#3{&%
5751   \bbl@activateprehyphen
5752   \begin{group}
5753     \def\babeltempa{\bbl@add@list\babeltempb}&%
5754     \let\babeltempb\@empty
5755     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5756     \bbl@replace\bbl@tempa{,}{ ,}&%
5757     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5758       \bbl@ifsamestring{##1}{remove}&%
5759       {\bbl@add@list\babeltempb{nil}}&%
5760       {\directlua{
5761         local rep = {[##1]=}
5762         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5763         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5764         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5765         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5766           'space = { ' .. '%2, %3, %4' .. ' }')
5767         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5768           'spacefactor = { ' .. '%2, %3, %4' .. ' }')
5769         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5770         tex.print([[ \string\babeltempa{}}] .. rep .. [[}}]])
5771       }}&%
5772     \directlua{
5773       local lbkr = Babel.linebreaking.replacements[0]
5774       local u = unicode.utf8
5775       local id = \the\csname bbl@id@#1\endcsname
5776       &% Convert pattern:
5777       local patt = string.gsub([=[#2]=], '%s', ' ')
5778       local patt = string.gsub(patt, '|', ' ')
5779       if not u.find(patt, '()', nil, true) then
5780         patt = '()' .. patt .. '()'
5781       end
5782       &% patt = string.gsub(patt, '%(%)^', '^()')
5783       &% patt = string.gsub(patt, '([^\%])%$%$', '%1()$')
5784       patt = u.gsub(patt, '{(.)}',
5785         function (n)
5786           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5787         end)
5788       patt = u.gsub(patt, '{(%x%x%x%x+)}',
5789         function (n)
5790           return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5791         end)
5792       lbkr[id] = lbkr[id] or {}
5793       table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5794     }&%
5795   \endgroup}
5796 \endgroup
5797 \def\bbl@activateposthyphen{%
5798   \let\bbl@activateposthyphen\relax
5799   \directlua{
5800     require('babel-transforms.lua')
5801     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5802   }}
5803 \def\bbl@activateprehyphen{%
5804   \let\bbl@activateprehyphen\relax

```

```

5805 \directlua{
5806   require('babel-transforms.lua')
5807   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5808 }}

```

13.9 Bidi

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 \TeX . Just in case, consider the possibility it has not been loaded.

```

5809 \def\bbl@activate@preotf{%
5810   \let\bbl@activate@preotf\relax % only once
5811   \directlua{
5812     Babel = Babel or {}
5813     %
5814     function Babel.pre_otfload_v(head)
5815       if Babel.numbers and Babel.digits_mapped then
5816         head = Babel.numbers(head)
5817       end
5818       if Babel.bidi_enabled then
5819         head = Babel.bidi(head, false, dir)
5820       end
5821       return head
5822     end
5823     %
5824     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5825       if Babel.numbers and Babel.digits_mapped then
5826         head = Babel.numbers(head)
5827       end
5828       if Babel.bidi_enabled then
5829         head = Babel.bidi(head, false, dir)
5830       end
5831       return head
5832     end
5833     %
5834     luatexbase.add_to_callback('pre_linebreak_filter',
5835       Babel.pre_otfload_v,
5836       'Babel.pre_otfload_v',
5837     luatexbase.priority_in_callback('pre_linebreak_filter',
5838       'luaotfload.node_processor') or nil)
5839     %
5840     luatexbase.add_to_callback('hpack_filter',
5841       Babel.pre_otfload_h,
5842       'Babel.pre_otfload_h',
5843     luatexbase.priority_in_callback('hpack_filter',
5844       'luaotfload.node_processor') or nil)
5845   }}

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option `bidi=`.

```

5846 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5847   \let\bbl@beforeforeign\leavevmode
5848   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5849   \RequirePackage{luatexbase}
5850   \bbl@activate@preotf
5851   \directlua{
5852     require('babel-data-bidi.lua')

```

```

5853 \ifcase\expandafter\@gobbletwo\the\bb1@bidimode\or
5854 require('babel-bidi-basic.lua')
5855 \or
5856 require('babel-bidi-basic-r.lua')
5857 \fi}
5858 % TODO - to locale_props, not as separate attribute
5859 \newattribute\bb1@attr@dir
5860 \directlua{ Babel.attr_dir = luatexbase.registernumber'bb1@attr@dir' }
5861 % TODO. I don't like it, hackish:
5862 \bb1@exp{\output{\bodydir\pagedir\the\output}}
5863 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5864 \fi\fi
5865 \chardef\bb1@thetextdir\z@
5866 \chardef\bb1@thepardir\z@
5867 \def\bb1@getluadir#1{%
5868 \directlua{
5869 if tex.#1dir == 'TLT' then
5870 tex.sprint('0')
5871 elseif tex.#1dir == 'TRT' then
5872 tex.sprint('1')
5873 end}}
5874 \def\bb1@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5875 \ifcase#3\relax
5876 \ifcase\bb1@getluadir{#1}\relax\else
5877 #2 TLT\relax
5878 \fi
5879 \else
5880 \ifcase\bb1@getluadir{#1}\relax
5881 #2 TRT\relax
5882 \fi
5883 \fi}
5884 \def\bb1@textdir#1{%
5885 \bb1@setluadir{text}\textdir{#1}%
5886 \chardef\bb1@thetextdir#1\relax
5887 \setattribute\bb1@attr@dir{\numexpr\bb1@thepardir*3+#1}}
5888 \def\bb1@pardir#1{%
5889 \bb1@setluadir{par}\pardir{#1}%
5890 \chardef\bb1@thepardir#1\relax}
5891 \def\bb1@bodydir{\bb1@setluadir{body}\bodydir}
5892 \def\bb1@pagedir{\bb1@setluadir{page}\pagedir}
5893 \def\bb1@dirparastext{\pardir\the\textdir\relax}% %%%
5894 %
5895 \ifnum\bb1@bidimode>\z@
5896 \def\bb1@mathboxdir{%
5897 \ifcase\bb1@thetextdir\relax
5898 \everyhbox{\bb1@mathboxdir@aux L}%
5899 \else
5900 \everyhbox{\bb1@mathboxdir@aux R}%
5901 \fi}
5902 \def\bb1@mathboxdir@aux#1{%
5903 \@ifnextchar\egroup{}\{\textdir T#1T\relax}}
5904 \frozen@everymath\expandafter{%
5905 \expandafter\bb1@mathboxdir\the\frozen@everymath}
5906 \frozen@everydisplay\expandafter{%
5907 \expandafter\bb1@mathboxdir\the\frozen@everydisplay}
5908 \fi

```

13.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

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

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```
5909 \bbl@trace{Redefinitions for bidi layout}
5910 \ifx\@eqnnum\undefined\else
5911   \ifx\bbl@attr@dir\undefined\else
5912     \edef\@eqnnum{%
5913       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5914       \unexpanded\expandafter{\@eqnnum}}
5915   \fi
5916 \fi
5917 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5918 \ifnum\bbl@bidimode>\z@
5919   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5920     \bbl@exp{%
5921       \mathdir\the\bodydir
5922       #1%           Once entered in math, set boxes to restore values
5923       \<ifmmode>%
5924         \everyvbox{%
5925           \the\everyvbox
5926           \bodydir\the\bodydir
5927           \mathdir\the\mathdir
5928           \everyhbox{\the\everyhbox}%
5929           \everyvbox{\the\everyvbox}}%
5930         \everyhbox{%
5931           \the\everyhbox
5932           \bodydir\the\bodydir
5933           \mathdir\the\mathdir
5934           \everyhbox{\the\everyhbox}%
5935           \everyvbox{\the\everyvbox}}%
5936       \<fi>}}%
5937   \def\@hangfrom#1{%
5938     \setbox\@tempboxa\hbox{{#1}}%
5939     \hangindent\wd\@tempboxa
5940     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5941       \shapemode\@ne
5942     \fi
5943     \noindent\box\@tempboxa}
5944 \fi
5945 \IfBabelLayout{tabular}
5946   {\let\bbl@OL@@tabular\@tabular
5947     \bbl@replace\@tabular{$}\{\bbl@nextfake$}%
5948     \let\bbl@NL@@tabular\@tabular
5949     \AtBeginDocument{%
5950       \ifx\bbl@NL@@tabular\@tabular\else
5951         \bbl@replace\@tabular{$}\{\bbl@nextfake$}%
5952         \let\bbl@NL@@tabular\@tabular
5953       \fi}}
```

```

5954 {}
5955 \IfBabelLayout{lists}
5956 {\let\bbl@OL@list\list
5957 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5958 \let\bbl@NL@list\list
5959 \def\bbl@listparshape#1#2#3{%
5960 \parshape #1 #2 #3 %
5961 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5962 \shapemode\tw@
5963 \fi}}
5964 {}
5965 \IfBabelLayout{graphics}
5966 {\let\bbl@pictresetdir\relax
5967 \def\bbl@pictsetdir#1{%
5968 \ifcase\bbl@thetextdir
5969 \let\bbl@pictresetdir\relax
5970 \else
5971 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5972 \or\textdir TLT
5973 \else\bodydir TLT \textdir TLT
5974 \fi
5975 % \(\text|par)dir required in pgf:
5976 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5977 \fi}%
5978 \ifx\AddToHook\undefined\else
5979 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
5980 \directlua{
5981 Babel.get_picture_dir = true
5982 Babel.picture_has_bidi = 0
5983 function Babel.picture_dir (head)
5984 if not Babel.get_picture_dir then return head end
5985 for item in node.traverse(head) do
5986 if item.id == node.id'glyph' then
5987 local itemchar = item.char
5988 % TODO. Copypaste pattern from Babel.bidi (-r)
5989 local chardata = Babel.characters[itemchar]
5990 local dir = chardata and chardata.d or nil
5991 if not dir then
5992 for nn, et in ipairs(Babel.ranges) do
5993 if itemchar < et[1] then
5994 break
5995 elseif itemchar <= et[2] then
5996 dir = et[3]
5997 break
5998 end
5999 end
6000 end
6001 if dir and (dir == 'al' or dir == 'r') then
6002 Babel.picture_has_bidi = 1
6003 end
6004 end
6005 end
6006 return head
6007 end
6008 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6009 "Babel.picture_dir")
6010 }%
6011 \AtBeginDocument{%
6012 \long\def\put(#1,#2)#3{%

```



```

6013 \killglue
6014 % Try:
6015 \ifx\bbbl@pictresetdir\relax
6016 \def\bbbl@tempc{0}%
6017 \else
6018 \directlua{
6019     Babel.get_picture_dir = true
6020     Babel.picture_has_bidi = 0
6021 }%
6022 \setbox\z@\hb@xt@\z@{%
6023     \@defaultunitsset\@tempdimc{#1}\unitlength
6024     \kern\@tempdimc
6025     #3\hss}%
6026 \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6027 \fi
6028 % Do:
6029 \@defaultunitsset\@tempdimc{#2}\unitlength
6030 \raise\@tempdimc\hb@xt@\z@{%
6031     \@defaultunitsset\@tempdimc{#1}\unitlength
6032     \kern\@tempdimc
6033     {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6034 \ignorespaces}%
6035 \MakeRobust\put}%
6036 \fi
6037 \AtBeginDocument
6038 {\ifx\tikz@atbegin@node\undefined\else
6039     \ifx\AddToHook\undefined\else % TODO. Still tentative.
6040     \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6041     \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6042     \fi
6043     \let\bbbl@OL@pgfpicture\pgfpicture
6044     \bbbl@sreplace\pgfpicture{\pgfpicturetrue}%
6045     {\bbbl@pictsetdir\z@\pgfpicturetrue}%
6046     \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6047     \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6048     \bbbl@sreplace\tikz{\beginpgroup}%
6049     {\beginpgroup\bbbl@pictsetdir\tw@}%
6050     \fi
6051     \ifx\AddToHook\undefined\else
6052     \AddToHook{env/tcolorbox/begin}{\bbbl@pictsetdir\@ne}%
6053     \fi
6054 }}
6055 {}

```

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.

```

6056 \IfBabelLayout{counters}%
6057 {\let\bbbl@OL@@textsuperscript\textsuperscript
6058 \bbbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6059 \let\bbbl@latinarabic=\@arabic
6060 \let\bbbl@OL@@arabic\@arabic
6061 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
6062 \@ifpackagewith{babel}{bidi=default}%
6063 {\let\bbbl@asciroman=\@roman
6064 \let\bbbl@OL@@roman\@roman
6065 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
6066 \let\bbbl@asciiRoman=\@Roman
6067 \let\bbbl@OL@@roman\@Roman

```



```

6109     end
6110     last = n
6111 end
6112 return head
6113 end
6114
6115 Babel.fetch_subtext = {}
6116
6117 Babel.ignore_pre_char = function(node)
6118     return (node.lang == Babel.nohyphenation)
6119 end
6120
6121 -- Merging both functions doesn't seem feasible, because there are too
6122 -- many differences.
6123 Babel.fetch_subtext[0] = function(head)
6124     local word_string = ''
6125     local word_nodes = {}
6126     local lang
6127     local item = head
6128     local inmath = false
6129
6130     while item do
6131
6132         if item.id == 11 then
6133             inmath = (item.subtype == 0)
6134         end
6135
6136         if inmath then
6137             -- pass
6138
6139         elseif item.id == 29 then
6140             local locale = node.get_attribute(item, Babel.attr_locale)
6141
6142             if lang == locale or lang == nil then
6143                 lang = lang or locale
6144                 if Babel.ignore_pre_char(item) then
6145                     word_string = word_string .. Babel.us_char
6146                 else
6147                     word_string = word_string .. unicode.utf8.char(item.char)
6148                 end
6149                 word_nodes[#word_nodes+1] = item
6150             else
6151                 break
6152             end
6153
6154         elseif item.id == 12 and item.subtype == 13 then
6155             word_string = word_string .. ' '
6156             word_nodes[#word_nodes+1] = item
6157
6158             -- Ignore leading unrecognized nodes, too.
6159             elseif word_string ~= '' then
6160                 word_string = word_string .. Babel.us_char
6161                 word_nodes[#word_nodes+1] = item -- Will be ignored
6162             end
6163
6164             item = item.next
6165         end
6166
6167         -- Here and above we remove some trailing chars but not the

```

```

6168 -- corresponding nodes. But they aren't accessed.
6169 if word_string:sub(-1) == ' ' then
6170     word_string = word_string:sub(1,-2)
6171 end
6172 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6173 return word_string, word_nodes, item, lang
6174 end
6175
6176 Babel.fetch_subtext[1] = function(head)
6177     local word_string = ''
6178     local word_nodes = {}
6179     local lang
6180     local item = head
6181     local inmath = false
6182
6183     while item do
6184
6185         if item.id == 11 then
6186             inmath = (item.subtype == 0)
6187         end
6188
6189         if inmath then
6190             -- pass
6191
6192         elseif item.id == 29 then
6193             if item.lang == lang or lang == nil then
6194                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6195                     lang = lang or item.lang
6196                     word_string = word_string .. unicode.utf8.char(item.char)
6197                     word_nodes[#word_nodes+1] = item
6198                 end
6199             else
6200                 break
6201             end
6202
6203         elseif item.id == 7 and item.subtype == 2 then
6204             word_string = word_string .. '='
6205             word_nodes[#word_nodes+1] = item
6206
6207         elseif item.id == 7 and item.subtype == 3 then
6208             word_string = word_string .. '|'
6209             word_nodes[#word_nodes+1] = item
6210
6211         -- (1) Go to next word if nothing was found, and (2) implicitly
6212         -- remove leading USs.
6213         elseif word_string == '' then
6214             -- pass
6215
6216         -- This is the responsible for splitting by words.
6217         elseif (item.id == 12 and item.subtype == 13) then
6218             break
6219
6220         else
6221             word_string = word_string .. Babel.us_char
6222             word_nodes[#word_nodes+1] = item -- Will be ignored
6223         end
6224
6225         item = item.next
6226     end

```

```

6227
6228 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6229 return word_string, word_nodes, item, lang
6230 end
6231
6232 function Babel.pre_hyphenate_replace(head)
6233   Babel.hyphenate_replace(head, 0)
6234 end
6235
6236 function Babel.post_hyphenate_replace(head)
6237   Babel.hyphenate_replace(head, 1)
6238 end
6239
6240 Babel.us_char = string.char(31)
6241
6242 function Babel.hyphenate_replace(head, mode)
6243   local u = unicode.utf8
6244   local lbkr = Babel.linebreaking.replacements[mode]
6245
6246   local word_head = head
6247
6248   while true do -- for each subtext block
6249
6250     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6251
6252     if Babel.debug then
6253       print()
6254       print((mode == 0) and '@@@<' or '@@@>', w)
6255     end
6256
6257     if nw == nil and w == '' then break end
6258
6259     if not lang then goto next end
6260     if not lbkr[lang] then goto next end
6261
6262     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6263     -- loops are nested.
6264     for k=1, #lbkr[lang] do
6265       local p = lbkr[lang][k].pattern
6266       local r = lbkr[lang][k].replace
6267
6268       if Babel.debug then
6269         print('*****', p, mode)
6270       end
6271
6272       -- This variable is set in some cases below to the first *byte*
6273       -- after the match, either as found by u.match (faster) or the
6274       -- computed position based on sc if w has changed.
6275       local last_match = 0
6276       local step = 0
6277
6278       -- For every match.
6279       while true do
6280         if Babel.debug then
6281           print('====')
6282         end
6283         local new -- used when inserting and removing nodes
6284
6285         local matches = { u.match(w, p, last_match) }

```

```

6286
6287     if #matches < 2 then break end
6288
6289     -- Get and remove empty captures (with ()'s, which return a
6290     -- number with the position), and keep actual captures
6291     -- (from (...)), if any, in matches.
6292     local first = table.remove(matches, 1)
6293     local last = table.remove(matches, #matches)
6294     -- Non re-fetched substrings may contain \31, which separates
6295     -- subsubstrings.
6296     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6297
6298     local save_last = last -- with A()BC()D, points to D
6299
6300     -- Fix offsets, from bytes to unicode. Explained above.
6301     first = u.len(w:sub(1, first-1)) + 1
6302     last = u.len(w:sub(1, last-1)) -- now last points to C
6303
6304     -- This loop stores in n small table the nodes
6305     -- corresponding to the pattern. Used by 'data' to provide a
6306     -- predictable behavior with 'insert' (now w_nodes is modified on
6307     -- the fly), and also access to 'remove'd nodes.
6308     local sc = first-1 -- Used below, too
6309     local data_nodes = {}
6310
6311     for q = 1, last-first+1 do
6312         data_nodes[q] = w_nodes[sc+q]
6313     end
6314
6315     -- This loop traverses the matched substring and takes the
6316     -- corresponding action stored in the replacement list.
6317     -- sc = the position in substr nodes / string
6318     -- rc = the replacement table index
6319     local rc = 0
6320
6321     while rc < last-first+1 do -- for each replacement
6322         if Babel.debug then
6323             print('.....', rc + 1)
6324         end
6325         sc = sc + 1
6326         rc = rc + 1
6327
6328         if Babel.debug then
6329             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6330             local ss = ''
6331             for itt in node.traverse(head) do
6332                 if itt.id == 29 then
6333                     ss = ss .. unicode.utf8.char(itt.char)
6334                 else
6335                     ss = ss .. '{' .. itt.id .. '}'
6336                 end
6337             end
6338             print('*****', ss)
6339         end
6340
6341         local crep = r[rc]
6342         local item = w_nodes[sc]
6343         local item_base = item

```

```

6345     local placeholder = Babel.us_char
6346     local d
6347
6348     if crep and crep.data then
6349         item_base = data_nodes[crep.data]
6350     end
6351
6352     if crep then
6353         step = crep.step or 0
6354     end
6355
6356     if crep and next(crep) == nil then -- = {}
6357         last_match = save_last    -- Optimization
6358         goto next
6359
6360     elseif crep == nil or crep.remove then
6361         node.remove(head, item)
6362         table.remove(w_nodes, sc)
6363         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6364         sc = sc - 1 -- Nothing has been inserted.
6365         last_match = utf8.offset(w, sc+1+step)
6366         goto next
6367
6368     elseif crep and crep.kashida then -- Experimental
6369         node.set_attribute(item,
6370             Babel.attr_kashida,
6371             crep.kashida)
6372         last_match = utf8.offset(w, sc+1+step)
6373         goto next
6374
6375     elseif crep and crep.string then
6376         local str = crep.string(matches)
6377         if str == '' then -- Gather with nil
6378             node.remove(head, item)
6379             table.remove(w_nodes, sc)
6380             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6381             sc = sc - 1 -- Nothing has been inserted.
6382         else
6383             local loop_first = true
6384             for s in string.utfvalues(str) do
6385                 d = node.copy(item_base)
6386                 d.char = s
6387                 if loop_first then
6388                     loop_first = false
6389                     head, new = node.insert_before(head, item, d)
6390                     if sc == 1 then
6391                         word_head = head
6392                     end
6393                     w_nodes[sc] = d
6394                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6395                 else
6396                     sc = sc + 1
6397                     head, new = node.insert_before(head, item, d)
6398                     table.insert(w_nodes, sc, new)
6399                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6400                 end
6401             end
6402             if Babel.debug then
6403                 print('.....', 'str')
6404                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)

```

```

6404         end
6405     end -- for
6406     node.remove(head, item)
6407 end -- if ''
6408 last_match = utf8.offset(w, sc+1+step)
6409 goto next
6410
6411 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6412     d = node.new(7, 0) -- (disc, discretionary)
6413     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6414     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6415     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6416     d.attr = item_base.attr
6417     if crep.pre == nil then -- TeXbook p96
6418         d.penalty = crep.penalty or tex.hyphenpenalty
6419     else
6420         d.penalty = crep.penalty or tex.exhyphenpenalty
6421     end
6422     placeholder = '|'
6423     head, new = node.insert_before(head, item, d)
6424
6425 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6426     -- ERROR
6427
6428 elseif crep and crep.penalty then
6429     d = node.new(14, 0) -- (penalty, userpenalty)
6430     d.attr = item_base.attr
6431     d.penalty = crep.penalty
6432     head, new = node.insert_before(head, item, d)
6433
6434 elseif crep and crep.space then
6435     -- 655360 = 10 pt = 10 * 65536 sp
6436     d = node.new(12, 13) -- (glue, spaceskip)
6437     local quad = font.getfont(item_base.font).size or 655360
6438     node.setglue(d, crep.space[1] * quad,
6439                 crep.space[2] * quad,
6440                 crep.space[3] * quad)
6441     if mode == 0 then
6442         placeholder = ' '
6443     end
6444     head, new = node.insert_before(head, item, d)
6445
6446 elseif crep and crep.spacefactor then
6447     d = node.new(12, 13) -- (glue, spaceskip)
6448     local base_font = font.getfont(item_base.font)
6449     node.setglue(d,
6450                 crep.spacefactor[1] * base_font.parameters['space'],
6451                 crep.spacefactor[2] * base_font.parameters['space_stretch'],
6452                 crep.spacefactor[3] * base_font.parameters['space_shrink'])
6453     if mode == 0 then
6454         placeholder = ' '
6455     end
6456     head, new = node.insert_before(head, item, d)
6457
6458 elseif mode == 0 and crep and crep.space then
6459     -- ERROR
6460
6461 end -- ie replacement cases
6462

```



```

6463         -- Shared by disc, space and penalty.
6464         if sc == 1 then
6465             word_head = head
6466         end
6467         if crep.insert then
6468             w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6469             table.insert(w_nodes, sc, new)
6470             last = last + 1
6471         else
6472             w_nodes[sc] = d
6473             node.remove(head, item)
6474             w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6475         end
6476
6477         last_match = utf8.offset(w, sc+1+step)
6478
6479         ::next::
6480
6481     end -- for each replacement
6482
6483     if Babel.debug then
6484         print('.....', '/')
6485         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6486     end
6487
6488     end -- for match
6489
6490     end -- for patterns
6491
6492     ::next::
6493     word_head = nw
6494 end -- for substring
6495 return head
6496 end
6497
6498 -- This table stores capture maps, numbered consecutively
6499 Babel.capture_maps = {}
6500
6501 -- The following functions belong to the next macro
6502 function Babel.capture_func(key, cap)
6503     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
6504     local cnt
6505     local u = unicode.utf8
6506     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(.-)}}', Babel.capture_func_map)
6507     if cnt == 0 then
6508         ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6509             function (n)
6510                 return u.char(tonumber(n, 16))
6511             end)
6512     end
6513     ret = ret:gsub("%[%[%]]%.%", '')
6514     ret = ret:gsub("%.%.%[%[%]]%", '')
6515     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6516 end
6517
6518 function Babel.capt_map(from, mapno)
6519     return Babel.capture_maps[mapno][from] or from
6520 end
6521

```

```

6522 -- Handle the {n|abc|ABC} syntax in captures
6523 function Babel.capture_func_map(capno, from, to)
6524     local u = unicode.utf8
6525     from = u.gsub(from, '{(%x%x%x%x+)}',
6526         function (n)
6527             return u.char(tonumber(n, 16))
6528         end)
6529     to = u.gsub(to, '{(%x%x%x%x+)}',
6530         function (n)
6531             return u.char(tonumber(n, 16))
6532         end)
6533     local froms = {}
6534     for s in string.utfcharacters(from) do
6535         table.insert(froms, s)
6536     end
6537     local cnt = 1
6538     table.insert(Babel.capture_maps, {})
6539     local mlen = table.getn(Babel.capture_maps)
6540     for s in string.utfcharacters(to) do
6541         Babel.capture_maps[mlen][froms[cnt]] = s
6542         cnt = cnt + 1
6543     end
6544     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6545         (mlen) .. ").. " .. "[["
6546 end
6547
6548 -- Create/Extend reversed sorted list of kashida weights:
6549 function Babel.capture_kashida(key, wt)
6550     wt = tonumber(wt)
6551     if Babel.kashida_wts then
6552         for p, q in ipairs(Babel.kashida_wts) do
6553             if wt == q then
6554                 break
6555             elseif wt > q then
6556                 table.insert(Babel.kashida_wts, p, wt)
6557                 break
6558             elseif table.getn(Babel.kashida_wts) == p then
6559                 table.insert(Babel.kashida_wts, wt)
6560             end
6561         end
6562     else
6563         Babel.kashida_wts = { wt }
6564     end
6565     return 'kashida = ' .. wt
6566 end
6567 </transforms>

```

13.12 Lua: 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.

```
6568 (*basic-r)
6569 Babel = Babel or {}
6570
6571 Babel.bidi_enabled = true
6572
6573 require('babel-data-bidi.lua')
6574
6575 local characters = Babel.characters
6576 local ranges = Babel.ranges
6577
6578 local DIR = node.id("dir")
6579
6580 local function dir_mark(head, from, to, outer)
6581   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6582   local d = node.new(DIR)
6583   d.dir = '+' .. dir
6584   node.insert_before(head, from, d)
6585   d = node.new(DIR)
6586   d.dir = '-' .. dir
6587   node.insert_after(head, to, d)
6588 end
6589
6590 function Babel.bidi(head, ispar)
6591   local first_n, last_n          -- first and last char with nums
6592   local last_es                  -- an auxiliary 'last' used with nums
6593   local first_d, last_d          -- first and last char in L/R block
6594   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):

```

6595 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6596 local strong_lr = (strong == 'l') and 'l' or 'r'
6597 local outer = strong
6598
6599 local new_dir = false
6600 local first_dir = false
6601 local inmath = false
6602
6603 local last_lr
6604
6605 local type_n = ''
6606
6607 for item in node.traverse(head) do
6608
6609   -- three cases: glyph, dir, otherwise
6610   if item.id == node.id'glyph'
6611     or (item.id == 7 and item.subtype == 2) then
6612
6613     local itemchar
6614     if item.id == 7 and item.subtype == 2 then
6615       itemchar = item.replace.char
6616     else
6617       itemchar = item.char
6618     end
6619     local chardata = characters[itemchar]
6620     dir = chardata and chardata.d or nil
6621     if not dir then
6622       for nn, et in ipairs(ranges) do
6623         if itemchar < et[1] then
6624           break
6625         elseif itemchar <= et[2] then
6626           dir = et[3]
6627           break
6628         end
6629       end
6630     end
6631     dir = dir or 'l'
6632     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).

```

6633   if new_dir then
6634     attr_dir = 0
6635     for at in node.traverse(item.attr) do
6636       if at.number == Babel.attr_dir then
6637         attr_dir = at.value % 3
6638       end
6639     end
6640     if attr_dir == 1 then
6641       strong = 'r'
6642     elseif attr_dir == 2 then
6643       strong = 'al'
6644     else
6645       strong = 'l'
6646     end
6647     strong_lr = (strong == 'l') and 'l' or 'r'

```

```

6648         outer = strong_lr
6649         new_dir = false
6650     end
6651
6652     if dir == 'nsm' then dir = strong end          -- W1

```

Numbers. The dual $\langle al \rangle / \langle r \rangle$ system for R is somewhat cumbersome.

```

6653         dir_real = dir          -- We need dir_real to set strong below
6654         if dir == 'al' then dir = 'r' end -- W3

```

By W2, there are no $\langle en \rangle$ $\langle et \rangle$ $\langle es \rangle$ if strong == $\langle al \rangle$, only $\langle an \rangle$. Therefore, there are not $\langle et en \rangle$ nor $\langle en et \rangle$, W5 can be ignored, and W6 applied:

```

6655         if strong == 'al' then
6656             if dir == 'en' then dir = 'an' end          -- W2
6657             if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6658             strong_lr = 'r'          -- W3
6659         end

```

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

```

6660     elseif item.id == node.id'dir' and not inmath then
6661         new_dir = true
6662         dir = nil
6663     elseif item.id == node.id'math' then
6664         inmath = (item.subtype == 0)
6665     else
6666         dir = nil          -- Not a char
6667     end

```

Numbers in R mode. A sequence of $\langle en \rangle$, $\langle et \rangle$, $\langle an \rangle$, $\langle es \rangle$ and $\langle cs \rangle$ 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 $\langle an \rangle$ is relevant if $\langle al \rangle$.

```

6668         if dir == 'en' or dir == 'an' or dir == 'et' then
6669             if dir ~= 'et' then
6670                 type_n = dir
6671             end
6672             first_n = first_n or item
6673             last_n = last_es or item
6674             last_es = nil
6675         elseif dir == 'es' and last_n then -- W3+W6
6676             last_es = item
6677         elseif dir == 'cs' then          -- it's right - do nothing
6678         elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6679             if strong_lr == 'r' and type_n ~= '' then
6680                 dir_mark(head, first_n, last_n, 'r')
6681             elseif strong_lr == 'l' and first_d and type_n == 'an' then
6682                 dir_mark(head, first_n, last_n, 'r')
6683                 dir_mark(head, first_d, last_d, outer)
6684                 first_d, last_d = nil, nil
6685             elseif strong_lr == 'l' and type_n ~= '' then
6686                 last_d = last_n
6687             end
6688             type_n = ''
6689             first_n, last_n = nil, nil
6690         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:

```
6691   if dir == 'l' or dir == 'r' then
6692     if dir ~= outer then
6693       first_d = first_d or item
6694       last_d = item
6695     elseif first_d and dir ~= strong_lr then
6696       dir_mark(head, first_d, last_d, outer)
6697       first_d, last_d = nil, nil
6698     end
6699   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.

```
6700   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6701     item.char = characters[item.char] and
6702               characters[item.char].m or item.char
6703   elseif (dir or new_dir) and last_lr ~= item then
6704     local mir = outer .. strong_lr .. (dir or outer)
6705     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6706       for ch in node.traverse(node.next(last_lr)) do
6707         if ch == item then break end
6708         if ch.id == node.id'glyph' and characters[ch.char] then
6709           ch.char = characters[ch.char].m or ch.char
6710         end
6711       end
6712     end
6713   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).

```
6714   if dir == 'l' or dir == 'r' then
6715     last_lr = item
6716     strong = dir_real          -- Don't search back - best save now
6717     strong_lr = (strong == 'l') and 'l' or 'r'
6718   elseif new_dir then
6719     last_lr = nil
6720   end
6721 end
```

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

```
6722   if last_lr and outer == 'r' then
6723     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6724       if characters[ch.char] then
6725         ch.char = characters[ch.char].m or ch.char
6726       end
6727     end
6728   end
6729   if first_n then
6730     dir_mark(head, first_n, last_n, outer)
6731   end
6732   if first_d then
6733     dir_mark(head, first_d, last_d, outer)
6734   end
```

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

```

6735 return node.prev(head) or head
6736 end
6737 </basic-r>

And here the Lua code for bidi=basic:

6738 (*basic)
6739 Babel = Babel or {}
6740
6741 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6742
6743 Babel.fontmap = Babel.fontmap or {}
6744 Babel.fontmap[0] = {}      -- l
6745 Babel.fontmap[1] = {}      -- r
6746 Babel.fontmap[2] = {}      -- al/an
6747
6748 Babel.bidi_enabled = true
6749 Babel.mirroring_enabled = true
6750
6751 require('babel-data-bidi.lua')
6752
6753 local characters = Babel.characters
6754 local ranges = Babel.ranges
6755
6756 local DIR = node.id('dir')
6757 local GLYPH = node.id('glyph')
6758
6759 local function insert_implicit(head, state, outer)
6760     local new_state = state
6761     if state.sim and state.eim and state.sim ~= state.eim then
6762         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6763         local d = node.new(DIR)
6764         d.dir = '+' .. dir
6765         node.insert_before(head, state.sim, d)
6766         local d = node.new(DIR)
6767         d.dir = '-' .. dir
6768         node.insert_after(head, state.eim, d)
6769     end
6770     new_state.sim, new_state.eim = nil, nil
6771     return head, new_state
6772 end
6773
6774 local function insert_numeric(head, state)
6775     local new
6776     local new_state = state
6777     if state.san and state.ean and state.san ~= state.ean then
6778         local d = node.new(DIR)
6779         d.dir = '+TLT'
6780         _, new = node.insert_before(head, state.san, d)
6781         if state.san == state.sim then state.sim = new end
6782         local d = node.new(DIR)
6783         d.dir = '-TLT'
6784         _, new = node.insert_after(head, state.ean, d)
6785         if state.ean == state.eim then state.eim = new end
6786     end
6787     new_state.san, new_state.ean = nil, nil
6788     return head, new_state
6789 end
6790
6791 -- TODO - \hbox with an explicit dir can lead to wrong results

```

```

6792 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6793 -- was s made to improve the situation, but the problem is the 3-dir
6794 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6795 -- well.
6796
6797 function Babel.bidi(head, ispar, hdir)
6798   local d    -- d is used mainly for computations in a loop
6799   local prev_d = ''
6800   local new_d = false
6801
6802   local nodes = {}
6803   local outer_first = nil
6804   local inmath = false
6805
6806   local glue_d = nil
6807   local glue_i = nil
6808
6809   local has_en = false
6810   local first_et = nil
6811
6812   local ATDIR = Babel.attr_dir
6813
6814   local save_outer
6815   local temp = node.get_attribute(head, ATDIR)
6816   if temp then
6817     temp = temp % 3
6818     save_outer = (temp == 0 and 'l') or
6819                  (temp == 1 and 'r') or
6820                  (temp == 2 and 'al')
6821   elseif ispar then -- Or error? Shouldn't happen
6822     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6823   else -- Or error? Shouldn't happen
6824     save_outer = ('TRT' == hdir) and 'r' or 'l'
6825   end
6826   -- when the callback is called, we are just _after_ the box,
6827   -- and the textdir is that of the surrounding text
6828   -- if not ispar and hdir ~= tex.textdir then
6829   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6830   -- end
6831   local outer = save_outer
6832   local last = outer
6833   -- 'al' is only taken into account in the first, current loop
6834   if save_outer == 'al' then save_outer = 'r' end
6835
6836   local fontmap = Babel.fontmap
6837
6838   for item in node.traverse(head) do
6839
6840     -- In what follows, #node is the last (previous) node, because the
6841     -- current one is not added until we start processing the neutrals.
6842
6843     -- three cases: glyph, dir, otherwise
6844     if item.id == GLYPH
6845        or (item.id == 7 and item.subtype == 2) then
6846
6847       local d_font = nil
6848       local item_r
6849       if item.id == 7 and item.subtype == 2 then
6850         item_r = item.replace -- automatic discs have just 1 glyph

```



```

6851     else
6852         item_r = item
6853     end
6854     local chardata = characters[item_r.char]
6855     d = chardata and chardata.d or nil
6856     if not d or d == 'nsm' then
6857         for nn, et in ipairs(ranges) do
6858             if item_r.char < et[1] then
6859                 break
6860             elseif item_r.char <= et[2] then
6861                 if not d then d = et[3]
6862                 elseif d == 'nsm' then d_font = et[3]
6863                 end
6864                 break
6865             end
6866         end
6867     end
6868     d = d or 'l'
6869
6870     -- A short 'pause' in bidi for mapfont
6871     d_font = d_font or d
6872     d_font = (d_font == 'l' and 0) or
6873             (d_font == 'nsm' and 0) or
6874             (d_font == 'r' and 1) or
6875             (d_font == 'al' and 2) or
6876             (d_font == 'an' and 2) or nil
6877     if d_font and fontmap and fontmap[d_font][item_r.font] then
6878         item_r.font = fontmap[d_font][item_r.font]
6879     end
6880
6881     if new_d then
6882         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6883         if inmath then
6884             attr_d = 0
6885         else
6886             attr_d = node.get_attribute(item, ATDIR)
6887             attr_d = attr_d % 3
6888         end
6889         if attr_d == 1 then
6890             outer_first = 'r'
6891             last = 'r'
6892         elseif attr_d == 2 then
6893             outer_first = 'r'
6894             last = 'al'
6895         else
6896             outer_first = 'l'
6897             last = 'l'
6898         end
6899         outer = last
6900         has_en = false
6901         first_et = nil
6902         new_d = false
6903     end
6904
6905     if glue_d then
6906         if (d == 'l' and 'l' or 'r') ~= glue_d then
6907             table.insert(nodes, {glue_i, 'on', nil})
6908         end
6909         glue_d = nil

```

```

6910         glue_i = nil
6911     end
6912
6913     elseif item.id == DIR then
6914         d = nil
6915         new_d = true
6916
6917     elseif item.id == node.id'glue' and item.subtype == 13 then
6918         glue_d = d
6919         glue_i = item
6920         d = nil
6921
6922     elseif item.id == node.id'math' then
6923         inmath = (item.subtype == 0)
6924
6925     else
6926         d = nil
6927     end
6928
6929     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6930     if last == 'al' and d == 'en' then
6931         d = 'an'          -- W3
6932     elseif last == 'al' and (d == 'et' or d == 'es') then
6933         d = 'on'          -- W6
6934     end
6935
6936     -- EN + CS/ES + EN      -- W4
6937     if d == 'en' and #nodes >= 2 then
6938         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6939             and nodes[#nodes-1][2] == 'en' then
6940             nodes[#nodes][2] = 'en'
6941         end
6942     end
6943
6944     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6945     if d == 'an' and #nodes >= 2 then
6946         if (nodes[#nodes][2] == 'cs')
6947             and nodes[#nodes-1][2] == 'an' then
6948             nodes[#nodes][2] = 'an'
6949         end
6950     end
6951
6952     -- ET/EN                -- W5 + W7->l / W6->on
6953     if d == 'et' then
6954         first_et = first_et or (#nodes + 1)
6955     elseif d == 'en' then
6956         has_en = true
6957         first_et = first_et or (#nodes + 1)
6958     elseif first_et then    -- d may be nil here !
6959         if has_en then
6960             if last == 'l' then
6961                 temp = 'l'    -- W7
6962             else
6963                 temp = 'en'   -- W5
6964             end
6965         else
6966             temp = 'on'       -- W6
6967         end
6968         for e = first_et, #nodes do

```

```

6969         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6970     end
6971     first_et = nil
6972     has_en = false
6973 end
6974
6975 -- Force mathdir in math if ON (currently works as expected only
6976 -- with 'l')
6977 if inmath and d == 'on' then
6978     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6979 end
6980
6981 if d then
6982     if d == 'al' then
6983         d = 'r'
6984         last = 'al'
6985     elseif d == 'l' or d == 'r' then
6986         last = d
6987     end
6988     prev_d = d
6989     table.insert(nodes, {item, d, outer_first})
6990 end
6991
6992 outer_first = nil
6993
6994 end
6995
6996 -- TODO -- repeated here in case EN/ET is the last node. Find a
6997 -- better way of doing things:
6998 if first_et then -- dir may be nil here !
6999     if has_en then
7000         if last == 'l' then
7001             temp = 'l' -- W7
7002         else
7003             temp = 'en' -- W5
7004         end
7005     else
7006         temp = 'on' -- W6
7007     end
7008     for e = first_et, #nodes do
7009         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7010     end
7011 end
7012
7013 -- dummy node, to close things
7014 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7015
7016 ----- NEUTRAL -----
7017
7018 outer = save_outer
7019 last = outer
7020
7021 local first_on = nil
7022
7023 for q = 1, #nodes do
7024     local item
7025
7026     local outer_first = nodes[q][3]
7027     outer = outer_first or outer

```

```

7028     last = outer_first or last
7029
7030     local d = nodes[q][2]
7031     if d == 'an' or d == 'en' then d = 'r' end
7032     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7033
7034     if d == 'on' then
7035         first_on = first_on or q
7036     elseif first_on then
7037         if last == d then
7038             temp = d
7039         else
7040             temp = outer
7041         end
7042         for r = first_on, q - 1 do
7043             nodes[r][2] = temp
7044             item = nodes[r][1] -- MIRRORING
7045             if Babel.mirroring_enabled and item.id == GLYPH
7046                 and temp == 'r' and characters[item.char] then
7047                 local font_mode = font.fonts[item.font].properties.mode
7048                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7049                     item.char = characters[item.char].m or item.char
7050                 end
7051             end
7052         end
7053         first_on = nil
7054     end
7055
7056     if d == 'r' or d == 'l' then last = d end
7057 end
7058
7059 ----- IMPLICIT, REORDER -----
7060
7061 outer = save_outer
7062 last = outer
7063
7064 local state = {}
7065 state.has_r = false
7066
7067 for q = 1, #nodes do
7068     local item = nodes[q][1]
7069
7070     outer = nodes[q][3] or outer
7071
7072     local d = nodes[q][2]
7073
7074     if d == 'nsm' then d = last end -- W1
7075     if d == 'en' then d = 'an' end
7076     local isdir = (d == 'r' or d == 'l')
7077
7078     if outer == 'l' and d == 'an' then
7079         state.san = state.san or item
7080         state.ean = item
7081     elseif state.san then
7082         head, state = insert_numeric(head, state)
7083     end
7084
7085     if outer == 'l' then

```

```

7087     if d == 'an' or d == 'r' then      -- im -> implicit
7088         if d == 'r' then state.has_r = true end
7089         state.sim = state.sim or item
7090         state.eim = item
7091     elseif d == 'l' and state.sim and state.has_r then
7092         head, state = insert_implicit(head, state, outer)
7093     elseif d == 'l' then
7094         state.sim, state.eim, state.has_r = nil, nil, false
7095     end
7096 else
7097     if d == 'an' or d == 'l' then
7098         if nodes[q][3] then -- nil except after an explicit dir
7099             state.sim = item -- so we move sim 'inside' the group
7100         else
7101             state.sim = state.sim or item
7102         end
7103         state.eim = item
7104     elseif d == 'r' and state.sim then
7105         head, state = insert_implicit(head, state, outer)
7106     elseif d == 'r' then
7107         state.sim, state.eim = nil, nil
7108     end
7109 end
7110
7111 if isdir then
7112     last = d      -- Don't search back - best save now
7113 elseif d == 'on' and state.san then
7114     state.san = state.san or item
7115     state.ean = item
7116 end
7117
7118 end
7119
7120 return node.prev(head) or head
7121 end
7122 </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.

```

7123 ⟨*nil⟩
7124 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7125 \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.

```

7126 \ifx\l@nil\undefined
7127   \newlanguage\l@nil
7128   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7129   \let\bbl@elt\relax
7130   \edef\bbl@languages{% Add it to the list of languages
7131     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
7132 \fi

```

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

```

7133 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
7134 \let\captionnil\@empty
7135 \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.

```

7136 \ldf@finish{nil}
7137 ⟨/nil⟩

```

16 Support for Plain T_EX (plain.def)

16.1 Not renaming hyphen.tex

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

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

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```

7138 ⟨*bplain | blplain⟩
7139 \catcode`\{=1 % left brace is begin-group character
7140 \catcode`\}=2 % right brace is end-group character
7141 \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).

```

7142 \openin 0 hyphen.cfg
7143 \ifeof0
7144 \else
7145   \let\ainput

```

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.

```
7146 \def\input #1 {%
7147   \let\input\a
7148   \a hyphen.cfg
7149   \let\a\undefined
7150 }
7151 \fi
7152 </bplain | bplain>
```

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

```
7153 <bplain>\a plain.tex
7154 <bplain>\a 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.

```
7155 <bplain>\def\fmtname{babel-plain}
7156 <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`.

```
7157 <<*Emulate LaTeX>> \equiv
7158 % == Code for plain ==
7159 \def\@empty{}
7160 \def\loadlocalcfg#1{%
7161   \openin0#1.cfg
7162   \ifeof0
7163     \closein0
7164   \else
7165     \closein0
7166     {\immediate\write16{*****}%
7167      \immediate\write16{* Local config file #1.cfg used}%
7168      \immediate\write16{*}%
7169     }
7170   \input #1.cfg\relax
7171   \fi
7172   \@endofldf}
```

16.3 General tools

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

```
7173 \long\def\@firstofone#1{#1}
7174 \long\def\@firstoftwo#1#2{#1}
7175 \long\def\@secondoftwo#1#2{#2}
7176 \def\@nnil{\@nil}
7177 \def\@gobbletwo#1#2{}
7178 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7179 \def\@star@or@long#1{%
7180   \@ifstar
7181   {\let\l@ngrel@x\relax#1}%
7182   {\let\l@ngrel@x\long#1}}
7183 \let\l@ngrel@x\relax
7184 \def\@car#1#2\@nil{#1}
```

```

7185 \def\@cdr#1#2\@nil{#2}
7186 \let\@typeset@protect\relax
7187 \let\protected@edef\edef
7188 \long\def\@gobble#1{}
7189 \edef\@backslashchar{\expandafter\@gobble\string\}
7190 \def\strip@prefix#1>{}
7191 \def\g@addto@macro#1#2{%
7192     \toks@\expandafter{#1#2}%
7193     \xdef#1{\the\toks@}}
7194 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7195 \def\@nameuse#1{\csname #1\endcsname}
7196 \def\@ifundefined#1{%
7197     \expandafter\ifx\csname#1\endcsname\relax
7198         \expandafter\@firstoftwo
7199     \else
7200         \expandafter\@secondoftwo
7201     \fi}
7202 \def\@expandtwoargs#1#2#3{%
7203     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7204 \def\zap@space#1 #2{%
7205     #1%
7206     \ifx#2\@empty\else\expandafter\zap@space\fi
7207     #2}
7208 \let\bbl@trace\@gobble

```

\LaTeX 2_ε has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

7209 \ifx\@preamblecmds\@undefined
7210     \def\@preamblecmds{}
7211 \fi
7212 \def\@onlypreamble#1{%
7213     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7214         \@preamblecmds\do#1}}
7215 \@onlypreamble\@onlypreamble

```

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

```

7216 \def\begin{document}{%
7217     \@begin{document}hook
7218     \global\let\@begin{document}hook\@undefined
7219     \def\do##1{\global\let##1\@undefined}%
7220     \@preamblecmds
7221     \global\let\do\noexpand}
7222 \ifx\@begin{document}hook\@undefined
7223     \def\@begin{document}hook{}
7224 \fi
7225 \@onlypreamble\@begin{document}hook
7226 \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`.

```

7227 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7228 \@onlypreamble\AtEndOfPackage
7229 \def\@endofldf{}
7230 \@onlypreamble\@endofldf
7231 \let\bbl@afterlang\@empty
7232 \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.


```

7233 \catcode`\&=\z@
7234 \ifx&\if@files\@undefined
7235   \expandafter\let\csname if@files\expandafter\endcsname
7236     \csname iffalse\endcsname
7237 \fi
7238 \catcode`\&=4

Mimick LATEX's commands to define control sequences.

7239 \def\newcommand{\@star@or@long\new@command}
7240 \def\new@command#1{%
7241   \@testopt{\@newcommand#1}0}
7242 \def\@newcommand#1[#2]{%
7243   \@ifnextchar [{\@xargdef#1[#2]}%
7244     {\@argdef#1[#2]}}
7245 \long\def\@argdef#1[#2]#3{%
7246   \@yargdef#1\@ne{#2}{#3}}
7247 \long\def\@xargdef#1[#2][#3]#4{%
7248   \expandafter\def\expandafter#1\expandafter{%
7249     \expandafter\@protected@testopt\expandafter #1%
7250     \csname\string#1\expandafter\endcsname{#3}}%
7251   \expandafter\@yargdef \csname\string#1\endcsname
7252   \tw@{#2}{#4}}
7253 \long\def\@yargdef#1#2#3{%
7254   \@tempcnta#3\relax
7255   \advance \@tempcnta \@ne
7256   \let\@hash@\relax
7257   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7258   \@tempcntb #2%
7259   \@whilenum\@tempcntb <\@tempcnta
7260   \do{%
7261     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7262     \advance\@tempcntb \@ne}%
7263   \let\@hash@###
7264   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7265 \def\providecommand{\@star@or@long\provide@command}
7266 \def\provide@command#1{%
7267   \begingroup
7268     \escapechar\m@ne\xdef\@tempa{\string#1}%
7269   \endgroup
7270   \expandafter\@ifundefined\@tempa
7271     {\def\reserved@a{\new@command#1}}%
7272     {\let\reserved@a\relax
7273     \def\reserved@a{\new@command\reserved@a}}%
7274   \reserved@a}%

7275 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7276 \def\declare@robustcommand#1{%
7277   \edef\reserved@a{\string#1}%
7278   \def\reserved@b{#1}%
7279   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7280   \edef#1{%
7281     \ifx\reserved@a\reserved@b
7282       \noexpand\x@protect
7283       \noexpand#1%
7284     \fi
7285     \noexpand\protect
7286     \expandafter\noexpand\csname
7287       \expandafter\@gobble\string#1 \endcsname
7288   }%
7289   \expandafter\new@command\csname

```

```

7290 \expandafter\@gobble\string#1 \endcsname
7291 }
7292 \def\x@protect#1{%
7293 \ifx\protect\@typeset@protect\else
7294 \x@protect#1%
7295 \fi
7296 }
7297 \catcode`\&=\z@ % Trick to hide conditionals
7298 \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`.

```

7299 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7300 \catcode`\&=4
7301 \ifx\in@\@undefined
7302 \def\in@#1#2{%
7303 \def\in@@##1#1##2##3\in@@{%
7304 \ifx\in@@##2\in@false\else\in@true\fi}%
7305 \in@@##1\in@\in@@}
7306 \else
7307 \let\bbl@tempa\@empty
7308 \fi
7309 \bbl@tempa

```

\TeX 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).

```

7310 \def\@ifpackagewith#1#2#3#4{#3}

```

The \TeX 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.

```

7311 \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 $\text{\TeX}2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

7312 \ifx\@tempcnta\@undefined
7313 \csname newcount\endcsname\@tempcnta\relax
7314 \fi
7315 \ifx\@tempcntb\@undefined
7316 \csname newcount\endcsname\@tempcntb\relax
7317 \fi

```

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

```

7318 \ifx\bye\@undefined
7319 \advance\count10 by -2\relax
7320 \fi
7321 \ifx\@ifnextchar\@undefined
7322 \def\@ifnextchar#1#2#3{%
7323 \let\reserved@d=#1%
7324 \def\reserved@a{#2}\def\reserved@b{#3}%
7325 \futurelet\@let@token\@ifnch}
7326 \def\@ifnch{%
7327 \ifx\@let@token\@sptoken
7328 \let\reserved@c\@ifnch
7329 \else

```

```

7330 \ifx\@let@token\reserved@
7331 \let\reserved@c\reserved@a
7332 \else
7333 \let\reserved@c\reserved@b
7334 \fi
7335 \fi
7336 \reserved@c}
7337 \def\:{\let@sptoken= } \: % this makes \@sptoken a space token
7338 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7339 \fi
7340 \def\@testopt#1#2{%
7341 \@ifnextchar[{#1}{#1[#2]}}
7342 \def\@protected@testopt#1{%
7343 \ifx\protect\@typeset@protect
7344 \expandafter\@testopt
7345 \else
7346 \@x@protect#1%
7347 \fi}
7348 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7349 #2\relax}\fi}
7350 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7351 \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

7352 \def\DeclareTextCommand{%
7353 \@dec@text@cmd\providecommand
7354 }
7355 \def\ProvideTextCommand{%
7356 \@dec@text@cmd\providecommand
7357 }
7358 \def\DeclareTextSymbol#1#2#3{%
7359 \@dec@text@cmd\chardef#1{#2}#3\relax
7360 }
7361 \def\@dec@text@cmd#1#2#3{%
7362 \expandafter\def\expandafter#2%
7363 \expandafter{%
7364 \csname#3-cmd\expandafter\endcsname
7365 \expandafter#2%
7366 \csname#3\string#2\endcsname
7367 }%
7368 % \let\@ifdefinable\@rc@ifdefinable
7369 \expandafter#1\csname#3\string#2\endcsname
7370 }
7371 \def\@current@cmd#1{%
7372 \ifx\protect\@typeset@protect\else
7373 \noexpand#1\expandafter\@gobble
7374 \fi
7375 }
7376 \def\@changed@cmd#1#2{%
7377 \ifx\protect\@typeset@protect
7378 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7379 \expandafter\ifx\csname ?\string#1\endcsname\relax
7380 \expandafter\def\csname ?\string#1\endcsname{%
7381 \@changed@x@err{#1}%
7382 }%
7383 \fi

```

```

7384      \global\expandafter\let
7385      \csname\cf@encoding \string#1\expandafter\endcsname
7386      \csname ?\string#1\endcsname
7387      \fi
7388      \csname\cf@encoding\string#1%
7389      \expandafter\endcsname
7390  \else
7391      \noexpand#1%
7392  \fi
7393 }
7394 \def\@changed@x@err#1{%
7395     \errhelp{Your command will be ignored, type <return> to proceed}%
7396     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7397 \def\DeclareTextCommandDefault#1{%
7398     \DeclareTextCommand#1?%
7399 }
7400 \def\ProvideTextCommandDefault#1{%
7401     \ProvideTextCommand#1?%
7402 }
7403 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7404 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7405 \def\DeclareTextAccent#1#2#3{%
7406     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7407 }
7408 \def\DeclareTextCompositeCommand#1#2#3#4{%
7409     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7410     \edef\reserved@b{\string##1}%
7411     \edef\reserved@c{%
7412         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7413     \ifx\reserved@b\reserved@c
7414         \expandafter\expandafter\expandafter\ifx
7415             \expandafter\@car\reserved@a\relax\relax\@nil
7416             \@text@composite
7417     \else
7418         \edef\reserved@b##1{%
7419             \def\expandafter\noexpand
7420                 \csname#2\string#1\endcsname####1{%
7421                 \noexpand\@text@composite
7422                 \expandafter\noexpand\csname#2\string#1\endcsname
7423                 #####1\noexpand\@empty\noexpand\@text@composite
7424                 {##1}%
7425             }%
7426         }%
7427         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7428     \fi
7429     \expandafter\def\csname\expandafter\string\csname
7430         #2\endcsname\string#1-\string#3\endcsname{#4}
7431 \else
7432     \errhelp{Your command will be ignored, type <return> to proceed}%
7433     \errmessage{\string\DeclareTextCompositeCommand\space used on
7434         inappropriate command \protect#1}
7435 \fi
7436 }
7437 \def\@text@composite#1#2#3\@text@composite{%
7438     \expandafter\@text@composite@x
7439     \csname\string#1-\string#2\endcsname
7440 }
7441 \def\@text@composite@x#1#2{%
7442     \ifx#1\relax

```

```

7443     #2%
7444 \else
7445     #1%
7446 \fi
7447 }
7448 %
7449 \def\@strip@args#1:#2-#3\@strip@args{#2}
7450 \def\DeclareTextComposite#1#2#3#4{%
7451     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7452     \bgroup
7453         \lcode` \@=#4%
7454         \lowercase{%
7455     \egroup
7456     \reserved@a @%
7457 }%
7458 }
7459 %
7460 \def\UseTextSymbol#1#2{#2}
7461 \def\UseTextAccent#1#2#3{}
7462 \def\@use@text@encoding#1{}
7463 \def\DeclareTextSymbolDefault#1#2{%
7464     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7465 }
7466 \def\DeclareTextAccentDefault#1#2{%
7467     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7468 }
7469 \def\cf@encoding{OT1}

```

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

```

7470 \DeclareTextAccent{"}{OT1}{127}
7471 \DeclareTextAccent{'}{OT1}{19}
7472 \DeclareTextAccent{^}{OT1}{94}
7473 \DeclareTextAccent{`}{OT1}{18}
7474 \DeclareTextAccent{~}{OT1}{126}

```

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

```

7475 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7476 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7477 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
7478 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
7479 \DeclareTextSymbol{\i}{OT1}{16}
7480 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain $T_{\text{E}}X$ doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```

7481 \ifx\scriptsize@undefined
7482     \let\scriptsize\sevenrm
7483 \fi
7484 % End of code for plain
7485 <</Emulate LaTeX>>

```

A proxy file:

```

7486 < *plain>
7487 \input babel.def
7488 </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 L^AT_EX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The T_EXbook*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *L^AT_EX, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: T_EXhax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German T_EX*, *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).