

Babel

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Version 3.80.2869
2022/09/23

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	7
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	10
1.10	Shorthands	12
1.11	Package options	15
1.12	The base option	17
1.13	ini files	17
1.14	Selecting fonts	25
1.15	Modifying a language	27
1.16	Creating a language	28
1.17	Digits and counters	31
1.18	Dates	33
1.19	Accessing language info	33
1.20	Hyphenation and line breaking	35
1.21	Transforms	37
1.22	Selection based on BCP 47 tags	39
1.23	Selecting scripts	40
1.24	Selecting directions	41
1.25	Language attributes	45
1.26	Hooks	45
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	48
1.29	Tweaking some features	48
1.30	Tips, workarounds, known issues and notes	48
1.31	Current and future work	50
1.32	Tentative and experimental code	50
2	Loading languages with language.dat	50
2.1	Format	51
3	The interface between the core of babel and the language definition files	51
3.1	Guidelines for contributed languages	52
3.2	Basic macros	53
3.3	Skeleton	54
3.4	Support for active characters	55
3.5	Support for saving macro definitions	55
3.6	Support for extending macros	56
3.7	Macros common to a number of languages	56
3.8	Encoding-dependent strings	56
3.9	Executing code based on the selector	59
II	Source code	60
4	Identification and loading of required files	60
5	locale directory	60

6	Tools	61
6.1	Multiple languages	65
6.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	66
6.3	<code>base</code>	67
6.4	<code>key=value</code> options and other general option	67
6.5	Conditional loading of shorthands	69
6.6	Interlude for Plain	70
7	Multiple languages	70
7.1	Selecting the language	73
7.2	Errors	81
7.3	Hooks	83
7.4	Setting up language files	85
7.5	Shorthands	87
7.6	Language attributes	96
7.7	Support for saving macro definitions	97
7.8	Short tags	99
7.9	Hyphens	99
7.10	Multiencoding strings	100
7.11	Macros common to a number of languages	107
7.12	Making glyphs available	107
7.12.1	Quotation marks	107
7.12.2	Letters	109
7.12.3	Shorthands for quotation marks	109
7.12.4	Umlauts and tremas	110
7.13	Layout	111
7.14	Load engine specific macros	112
7.15	Creating and modifying languages	112
8	Adjusting the Babel bahavior	134
8.1	Cross referencing macros	136
8.2	Marks	139
8.3	Preventing clashes with other packages	140
8.3.1	<code>ifthen</code>	140
8.3.2	<code>varioref</code>	140
8.3.3	<code>hhline</code>	141
8.4	Encoding and fonts	141
8.5	Basic bidi support	143
8.6	Local Language Configuration	146
8.7	Language options	146
9	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	150
10	Loading hyphenation patterns	150
11	Font handling with <code>fontspec</code>	154
12	Hooks for XeTeX and LuaTeX	157
12.1	XeTeX	157
12.2	Layout	159
12.3	LuaTeX	161
12.4	Southeast Asian scripts	167
12.5	CJK line breaking	168
12.6	Arabic justification	170
12.7	Common stuff	174
12.8	Automatic fonts and ids switching	174
12.9	Bidi	179
12.10	Layout	181
12.11	Lua: transforms	186

12.12	Lua: Auto bidi with basic and basic-r	194
13	Data for CJK	204
14	The ‘nil’ language	204
15	Calendars	205
15.1	Islamic	205
16	Hebrew	207
17	Persian	211
18	Coptic and Ethiopic	212
19	Buddhist	212
20	Support for Plain T_EX (plain.def)	212
20.1	Not renaming hyphen.tex	212
20.2	Emulating some L ^A T _E X features	213
20.3	General tools	214
20.4	Encoding related macros	217
21	Acknowledgements	220

Troubleshoooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	8
Unknown language ‘LANG’	8
Argument of \language@active@arg” has an extra }	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	26
Package babel Info: The following fonts are not babel standard families	26

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

EXAMPLE Here is a simple full example for “traditional” \TeX engines (see below for xetex and luatex). The packages `fontenc` and `inputenc` do not belong to babel, but they are included in the example because typically you will need them. It assumes UTF-8, the default encoding:

PDF \TeX

```
\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 \TeX 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, T_EXLive, 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 L^AT_EX, the preamble of the document:

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

would tell L^AT_EX 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 is a real reason to do so:

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

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

EXAMPLE 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}
```

NOTE 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 pdf_{tex} follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDF_{TEX}

```
\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 x_{etex} and l_{uatex}, 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, because the default font supports both languages.

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:


```
\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. `lu` can be the locale name with tag `khh` or the tag for `lubakatanga`). 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 \LaTeX (ie, `\usepackage{<language>}`) is deprecated and you will get the error:²

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

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

```
! Package babel Error: Unknown language `#1'. Either you have
(babel)                misspelled its name, it has not been installed,
(babel)                or you requested it in a previous run. Fix its name,
(babel)                install it or just rerun the file, respectively. In
(babel)                some cases, you may need to remove the aux file
```

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.

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

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

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

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.

NOTE Bear in mind `\selectlanguage` can be automatically executed, in some cases, in the auxiliary files, at heads and foots, and after the environment `otherlanguage*`.

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 There are a couple of issues related to the way the language information is written to the auxiliary files:

- `\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 to be correctly synchronized. This rarely happens, but if it were the case, you must use `otherlanguage` instead.
- In addition, this macro inserts a `\write` in vertical mode, which may break the vertical spacing in some cases (for example, between lists). **New 3.64** The behavior can be adjusted with `\babeladjust{select.write=<mode>}`, where `<mode>` is `shift` (which shifts the skips down and adds a `\penalty`); `keep` (the default – with it the `\write` and the skips are kept in the order they are written), and `omit` (which may seem a too drastic solution, because nothing is written, but more often than not this command is applied to more or less short texts with no sectioning or similar commands and therefore no language synchronization is necessary).

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

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

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

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

Spaces after the environment are ignored.

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

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

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

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`. Furthermore, and because of this overloading, detecting the language of a chunk of text by external tools can become unfeasible. 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 `ruussian`, 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{ruussian}{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 of `\dag`). With `ini` files (see below), captions are ensured by default.

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things; for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionary and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\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:

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

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).

`\useshortands` `*{\langle char \rangle}`

The command `\useshortands` 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 `\useshortands*{\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 `\useshortands`. This restriction will be lifted in a future release.

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

The command `\defineshortand` 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 `\languageshortands{\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 dictionaries (languages do not define shorthands consistently, and “-”, “\”, “=” have different meanings). You can start with, say:

```
\useshortands*{"}  
\defineshortand{"*}{\babelhyphen{soft}}  
\defineshortand{"-}{\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:

```
\defineshortand[*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.

`\languageshortands` `{\langle language \rangle}`

The command `\languageshortands` 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{\languageshortands{ngerman}}
```

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

⁵ Actually, 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.

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]{\{\language shorthands{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:

```
\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

⁶Thanks to Enrico Gregorio

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

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.

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= `<char><char>... | 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 \TeX 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 \TeX 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= *<file>*

Load *<file>*.cfg instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).

main= *<language>*

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 T_EX, LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUpper case` and the like (this feature misuses some internal L^AT_EX tools, so use it only as a last resort).

hyphenmap= `off` | `first` | `select` | `other` | `other*`

New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.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 `other language`;

other* also sets it at `other language*` 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

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.

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.

provide= *

New 3.49 An alternative to `\babelprovide` for languages passed as options. See section 1.13, which describes also the variants `provide+=` and `provide*=`.

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 $\langle\textit{option-name}\rangle\{\langle\textit{code}\rangle\}$

This command is currently the only provided by `base`. Executes $\langle\textit{code}\rangle$ 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 $\langle\textit{option-name}\rangle$ 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}
```

NOTE With a recent version of \LaTeX , an alternative method to execute some code just after an `ldf` file is loaded is with `\AddToHook` and the hook `file/<language>.ldf/after`. `Babel` does not predeclare it, and you have to do it yourself with `\ActivateGenericHook`.

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 250 of these files containing the basic data required for a locale, plus basic templates for 500 about locales.

`ini` files are not meant only for `babel`, and they have been devised as a resource for other packages. To easy interoperability between \TeX 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{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო 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 math and 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).

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{lṇ lṃ lṁ lṅ lṇ lṅ} % 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 shorts texts the ini files should be fine, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX, etc.). This is what the class ltjbook does with luatex, which can be used in conjunction with the ldf for japanese, because the following piece of code loads luatexja:

```
\documentclass[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 hyphenations points are discarded (this bug is 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}	asa	Asu
agq	Aghem	ast	Asturian ^{ul}
ak	Akan	az-Cyrl	Azerbaijani
am	Amharic ^{ul}	az-Latn	Azerbaijani
ar	Arabic ^{ul}	az	Azerbaijani ^{ul}
ar-DZ	Arabic ^{ul}	bas	Basaa
ar-EG	Arabic ^{ul}	be	Belarusian ^{ul}
ar-IQ	Arabic ^{ul}	bem	Bemba
ar-JO	Arabic ^{ul}	bez	Bena
ar-LB	Arabic ^{ul}	bg	Bulgarian ^{ul}
ar-MA	Arabic ^{ul}	bm	Bambara
ar-PS	Arabic ^{ul}	bn	Bangla ^{ul}
ar-SA	Arabic ^{ul}	bo	Tibetan ^u
ar-SY	Arabic ^{ul}	brx	Bodo
ar-TN	Arabic ^{ul}	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian ^{ul}

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

mgo	Meta'	shi-Tfng	Tachelhit
mk	Macedonian ^{ul}	shi	Tachelhit
ml	Malayalam ^{ul}	si	Sinhala
mn	Mongolian	sk	Slovak ^{ul}
mr	Marathi ^{ul}	sl	Slovenian ^{ul}
ms-BN	Malay ^l	smn	Inari Sami
ms-SG	Malay ^l	sn	Shona
ms	Malay ^{ul}	so	Somali
mt	Maltese	sq	Albanian ^{ul}
mua	Mundang	sr-Cyrl-BA	Serbian ^{ul}
my	Burmese	sr-Cyrl-ME	Serbian ^{ul}
mzn	Mazanderani	sr-Cyrl-XK	Serbian ^{ul}
naq	Nama	sr-Cyrl	Serbian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Latn-BA	Serbian ^{ul}
nd	North Ndebele	sr-Latn-ME	Serbian ^{ul}
ne	Nepali	sr-Latn-XK	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Latn	Serbian ^{ul}
nmg	Kwasio	sr	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sv	Swedish ^{ul}
nnh	Ngiemboon	sw	Swahili
no	Norwegian	ta	Tamil ^u
nus	Nuer	te	Telugu ^{ul}
nyn	Nyankole	teo	Teso
om	Oromo	th	Thai ^{ul}
or	Odia	ti	Tigrinya
os	Ossetic	tk	Turkmen ^{ul}
pa-Arab	Punjabi	to	Tongan
pa-Guru	Punjabi	tr	Turkish ^{ul}
pa	Punjabi	twq	Tasawaq
pl	Polish ^{ul}	tzm	Central Atlas Tamazight
pms	Piedmontese ^{ul}	ug	Uyghur
ps	Pashto	uk	Ukrainian ^{ul}
pt-BR	Portuguese ^{ul}	ur	Urdu ^{ul}
pt-PT	Portuguese ^{ul}	uz-Arab	Uzbek
pt	Portuguese ^{ul}	uz-Cyrl	Uzbek
qu	Quechua	uz-Latn	Uzbek
rm	Romansh ^{ul}	uz	Uzbek
rn	Rundi	vai-Latn	Vai
ro	Romanian ^{ul}	vai-Vaii	Vai
ro-MD	Moldavian ^{ul}	vai	Vai
rof	Rombo	vi	Vietnamese ^{ul}
ru	Russian ^{ul}	vun	Vunjo
rw	Kinyarwanda	wae	Walser
rwk	Rwa	xog	Soga
sa-Beng	Sanskrit	yav	Yangben
sa-Deva	Sanskrit	yi	Yiddish
sa-Gujr	Sanskrit	yo	Yoruba
sa-Knda	Sanskrit	yue	Cantonese
sa-Mlym	Sanskrit	zgh	Standard Moroccan Tamazight
sa-Telu	Sanskrit		
sa	Sanskrit	zh-Hans-HK	Chinese ^u
sah	Sakha	zh-Hans-MO	Chinese ^u
saq	Samburu	zh-Hans-SG	Chinese ^u
sbp	Sangu	zh-Hans	Chinese ^u
se	Northern Sami ^{ul}	zh-Hant-HK	Chinese ^u
seh	Sena	zh-Hant-MO	Chinese ^u
ses	Koyraboro Senni	zh-Hant	Chinese ^u
sg	Sango	zh	Chinese ^u
shi-Latn	Tachelhit	zu	Zulu

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

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

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

filipino	kwasio
finnish	kyrgyz
french-be	lakota
french-belgium	langi
french-ca	lao
french-canada	latvian
french-ch	lingala
french-lu	lithuanian
french-luxembourg	lowersorbian
french-switzerland	lsorbian
french	lubakatanga
friulian	luo
fulah	luxembourgish
galician	luyia
ganda	macedonian
georgian	machame
german-at	makhuwameetto
german-austria	makonde
german-ch	malagasy
german-switzerland	malay-bn
german	malay-brunei
greek	malay-sg
gujarati	malay-singapore
gusii	malay
hausa-gh	malayalam
hausa-ghana	maltese
hausa-ne	manx
hausa-niger	marathi
hausa	masai
hawaiian	mazanderani
hebrew	meru
hindi	meta
hungarian	mexican
icelandic	mongolian
igbo	morisyen
inarisami	mundang
indonesian	nama
interlingua	nepali
irish	newzealand
italian	ngiemboon
japanese	ngomba
jolafonyi	norsk
kabuverdianu	northernluri
kabyle	northernsami
kako	northndebele
kalaallisut	norwegianbokmal
kalenjin	norwegiannynorsk
kamba	nswissgerman
kannada	nuer
kashmiri	nyankole
kazakh	nynorsk
khmer	occitan
kikuyu	oriya
kinyarwanda	oromo
konkani	ossetic
korean	pashto
koyraborosenni	persian
koyrachiini	piedmontese

polish	sinhala
polytonicgreek	slovak
portuguese-br	slovene
portuguese-brazil	slovenian
portuguese-portugal	soga
portuguese-pt	somali
portuguese	spanish-mexico
punjabi-arab	spanish-mx
punjabi-arabic	spanish
punjabi-gurmukhi	standardmoroccantamazight
punjabi-guru	swahili
punjabi	swedish
quechua	swissgerman
romanian	tachelhit-latin
romansh	tachelhit-latn
rombo	tachelhit-tfng
rundi	tachelhit-tifinagh
russian	tachelhit
rwa	taita
sakha	tamil
samburu	tasawaq
samin	telugu
sango	teso
sangu	thai
sanskrit-beng	tibetan
sanskrit-bengali	tigrinya
sanskrit-deva	tongan
sanskrit-devanagari	turkish
sanskrit-gujarati	turkmen
sanskrit-gujr	ukenglish
sanskrit-kannada	ukrainian
sanskrit-knda	uppersorbian
sanskrit-malayalam	urdu
sanskrit-mlym	usenglish
sanskrit-telu	usorbian
sanskrit-telugu	uyghur
sanskrit	uzbek-arab
scottishgaelic	uzbek-arabic
sena	uzbek-cyrillic
serbian-cyrillic-bosniaherzegovina	uzbek-cyrl
serbian-cyrillic-kosovo	uzbek-latin
serbian-cyrillic-montenegro	uzbek-latn
serbian-cyrillic	uzbek
serbian-cyrl-ba	vai-latin
serbian-cyrl-me	vai-latn
serbian-cyrl-xk	vai-vai
serbian-cyrl	vai-vaii
serbian-latin-bosniaherzegovina	vai
serbian-latin-kosovo	vietnam
serbian-latin-montenegro	vietnamese
serbian-latin	vunjo
serbian-latn-ba	walser
serbian-latn-me	welsh
serbian-latn-xk	westernfrisian
serbian-latn	yangben
serbian	yiddish
shambala	yoruba
shona	zarma
sichuanyi	zulu afrikaans

Modifying and adding values to ini files

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

1.14 Selecting fonts

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

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}{עִבְרִית} svenska.

\end{document}
```

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

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

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* an 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* an 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.

NOTE `\babelfont` is a high level interface to `fontspec`, and therefore in xetex you can apply Mappings. For example, there is a set of [transliterations for Brahmic scripts](#) by Davis M. Jones. After installing them in you distribution, just set the map as you would do with `fontspec`.

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

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 import’ed from ini files, you can modify the values of specific keys, like:

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

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

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

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

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do 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= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

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

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

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

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 (xetex or luatex) is mainly in Polytonic Greek with 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, polytonicgreek, provide=*]{babel}
```

Remember there is an alternative syntax for the latter:

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

Finally, also remember you might not need to load `italian` at all if there are only a few word in this language (see 1.3).

script= *<script-name>*

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

language= *<language-name>*

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

alph= *<counter-name>*

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

Alph= *<counter-name>*

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

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

intrapenalty= $\langle penalty \rangle$

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

transforms= $\langle transform-list \rangle$

See section 1.21.

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

Languages providing native digits in all or some variants are:

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

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

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

`\localnumeral` $\{\langle style \rangle\}\{\langle number \rangle\}$
`\localecounter` $\{\langle style \rangle\}\{\langle counter \rangle\}$

New 3.41 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):

- `\localnumeral{\langle style \rangle}\{\langle number \rangle\}`, like `\localnumeral{abjad}{15}`
- `\localecounter{\langle style \rangle}\{\langle counter \rangle\}`, 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`
Armenian `lower.letter`, `upper.letter`
Belarusan, Bulgarian, Church Slavic, Macedonian, Serbian `lower`, `upper`
Bangla `alphabetic`
Central Kurdish `alphabetic`
Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
Church Slavic (Glagolitic) `letters`
Coptic `epact`, `lower.letters`
French `date.day` (mainly for internal use).
Georgian `letters`
Greek `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with `keraia`)
Hebrew `letters` (neither `geresh` nor `gershayim yet`)
Hindi `alphabetic`
Italian `lower.legal`, `upper.legal`
Japanese `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`, `informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

Khmer consonant
Korean consonant, syllable, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, 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

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=.., convert>*]{*<year>*}{*<month>*}{*<day>*}

By default the calendar is the Gregorian, but an ini file 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 and calendar=coptic). However, with the option convert it's converted (using internally the following command).

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

\babelcalendar [*<date>*]{*<calendar>*}{*<year-macro>*}{*<month-macro>*}{*<day-macro>*}

New 3.76 Although calendars aren't the primary concern of babel, the package should be able to, at least, generate correctly the current date in the way users would expect in their own culture. Currently, \localedate can print dates in a few calendars (provided the ini locale file has been imported), but year, month and day had to be entered by hand, which is very inconvenient. With this macro, the current date is converted and stored in the three last arguments, which must be macros: allowed calendars are buddhist, coptic, hebrew, islamic-civil, islamic-umalqura, persian. The optional argument converts the given date, in the form '*<year>*-'*<month>*-'*<day>*'. Please, refer to the page on the news for 3.76 in the babel site for further details.

1.19 Accessing language info

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

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

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

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

\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). This is the value to be used for the ‘real’ provided tag (babel may fill other fields if they are considered necessary).

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. This is a required field for the fonts to be correctly set up, and therefore it should be always defined.

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

region.tag.bcp47 is the BCP 47 tag of the region or territory. Defined only if the locale loaded actually contains it (eg, es-MX does, but es doesn’t), which is how locales behave in the CLDR. **New 3.75**

variant.tag.bcp47 is the BCP 47 tag of the variant (in the BCP 47 sense, like 1901 for German). **New 3.75**

extension.<s>.tag.bcp47 is the BCP 47 value of the extension whose singleton is <s> (currently the recognized singletons are x, t and u). The internal syntax can be somewhat complex, and this feature is still somewhat tentative. An example is classiclatin which sets extension.x.tag.bcp47 to classic. **New 3.75**

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.

New 3.75 Sometimes, it comes in handy to be able to use \localeinfo in an expandable way even if something went wrong (for example, the locale currently active is undefined). For these cases, localeinfo* just returns an empty string instead of raising an error. Bear in mind that babel, following the CLDR, may leave the region unset, which means \getlanguageproperty*, described below, is the preferred command, so that the existence of a field can be checked before. This also means building a string with the language and the region with \localeinfo*{language.tab.bcp47}-\localeinfo*{region.tab.bcp47} is not usually a good idea (because of the hyphen).

\getlocaleproperty *{<macro>}{<locale>}{<property>}

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

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

the macro \hechap will contain the string פרק.

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.

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

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 (when it makes sense) as an attribute, too.

`\LocaleForEach {<code>}`

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.

`ensureinfo=off` **New 3.75** Previously, ini files were loaded only with `\babelprovide` and also when languages are selected if there is a `\babelfont` or they have not been explicitly declared. Now the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met (in previous versions you had to enable it with `\BabelEnsureInfo` in the preamble). Because of the way this feature works, problems are very unlikely, but there is switch as a package option to turn the new behavior off (`ensureinfo=off`).

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. With luatex there are also tools for non-standard hyphenation rules, explained in the next section.

`\babelhyphen *` {<type>}

`\babelhyphen *` {<text>}

New 3.9a It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in T_EX 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 T_EX 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 T_EX, `-` 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 L^AT_EX: (1) the character used is that set for the current font, while in L^AT_EX it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in L^AT_EX, but it can be changed to another value by redefining `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue `>0 pt` (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Multiple declarations work much like `\hyphenation` (last wins), but 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 Use `\babelhyphenation` instead of `\hyphenation` to set hyphenation exceptions in the preamble before any language is explicitly set with a selector. 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 `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`).

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.

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}
```

New 3.67 Transforms predefined in the ini locale files can be made attribute-dependent, too. When an attribute between parenthesis is inserted subsequent transforms will be assigned to it (up to the list end or another attribute). For example, and provided an attribute called `\withsigmafinal` has been declared:

```
transforms = transliteration.omega (\withsigmafinal) sigma.final
```

This applies `transliteration.omega` always, but `sigma.final` only when `\withsigmafinal` is set.

Here are the transforms currently predefined. (A few may still require some fine-tuning. 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.
Finnish	<code>prehyphen.nobreak</code>	Line breaks just after hyphens prepended to words are prevented, like in “pakastekaapit ja -arkut”.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	<code>transliteration.omega</code>	Although the provided combinations are not the full set, this transform follows the syntax of Omega: = for the circumflex, v for digamma, and so on. For better compatibility with Levy’s system, ~ (as ‘string’) is an alternative to =. ' is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.

¹⁵They are similar in concept, but not the same, as those in Unicode. The main inspiration for this feature is the Omega transformation processes.

Greek	<code>sigma.final</code>	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transform does it. To prevent the conversion (an abbreviation, for example), write "s.
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: <code>!?:;</code> .
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs</i> , <i>ddz</i> , <i>ggy</i> , <i>lly</i> , <i>nny</i> , <i>ssz</i> , <i>tty</i> and <i>zzs</i> as <i>cs-cs</i> , <i>dz-dz</i> , etc.
Indic scripts	<code>danda.nobreak</code>	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.
Latin	<code>digraphs.ligatures</code>	Replaces the groups <i>ae</i> , <i>AE</i> , <i>oe</i> , <i>OE</i> with <i>æ</i> , <i>Æ</i> , <i>œ</i> , <i>Œ</i> .
Latin	<code>letters.noj</code>	Replaces <i>j</i> , <i>J</i> with <i>i</i> , <i>I</i> .
Latin	<code>letters.uv</code>	Replaces <i>v</i> , <i>U</i> with <i>u</i> , <i>V</i> .
Sanskrit	<code>transliteration.iast</code>	The IAST system to romanize Devanagari. ¹⁶
Serbian	<code>transliteration.gajica</code>	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	<code>kashida.plain</code>	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.

`\babelposthyphenation` [*options*]{*hyphenrules-name*}{*lua-pattern*}{*replacement*}

New 3.37-3.39 With *luatex* it is possible to define non-standard hyphenation rules, like *f-f* → *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}{([fmrtp]) | {1}}
{
  { no = {1}, pre = {1}{1}- }, % Replace first char with disc
  remove,                    % Remove automatic disc (2nd node)
  {}                          % Keep last char, untouched
}
```

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

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

New 3.67 With the optional argument you can associate a user defined transform to an attribute, so that it’s active only when it’s set (currently its attribute value is ignored). With this mechanism transforms can be set or unset even in the middle of paragraphs, and applied to single words. To define, set and unset the attribute, the LaTeX kernel provides the macros `\newattribute`, `\setattribute` and `\unsetattribute`. The following example shows how to use it, provided an attribute named `\latinnoj` has been declared:


```
\babelprehyphenation[attribute=\latinnoj]{\latin}{ J }{ string = I }
```

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 [*\options*]{*\locale-name*}{*\lua-pattern*}{*\replacement*}

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.

See the description above for the optional argument.

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: $\text{fr-Latn-FR} \rightarrow \text{fr-Latn} \rightarrow \text{fr-FR} \rightarrow \text{fr}$. Languages with the same resolved name are considered the same. Case is normalized before, so that $\text{fr-latn-fr} \rightarrow \text{fr-Latn-FR}$. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{
  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.¹⁷ 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.¹⁸

`\ensureascii` $\{\langle text \rangle\}$

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

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

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

1.24 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 are progresses in the latter, including `amsmath` and `mathtools` too, but for example gathered 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

¹⁷The 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.

¹⁸But still defined for backwards compatibility.

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 (الاعريقي) بـ
    Arabia أو Aravia (بالاعريقية Ἀραβία)، استخدم الرومان ثلاث
    بادئات بـ“Arabia” على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
    حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

    Most Arabic speakers consider the two varieties to be two registers
    of one language, although the two registers can be referred to in
    Arabic as 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`..`\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.¹⁹

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** .

¹⁹Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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 `\LaTeXe` **New 3.19** .

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr `{\lr-text}`

Digits in pdf_{tex} must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, `xetex`). This command is provided to set `{\lr-text}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `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 `{\section-name}`

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 `{\cmd}{\local-language}{\before}{\after}`

New 3.17 Something like:

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

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

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

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

```

\BabelFootnote{\footnote}{\language}\language}%
\BabelFootnote{\localfootnote}{\language}\language}%
\BabelFootnote{\mainfootnote}{\language}\language}%

```

(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}{\dot}.

```

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.

New 3.64 This is not the only way to inject code at those points. The events listed below can be used as a hook name in `\AddToHook` in the form `babel/⟨language-name⟩/⟨event-name⟩` (with `*` it's applied to all languages), but there is a limitation, because the parameters passed with the babel mechanism are not allowed. The `\AddToHook` mechanism does *not* replace the current one in 'babel'. Its main advantage is you can reconfigure 'babel' even before loading it. See the example below.

`\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 `\usesshortands*` to add a hook for the event `afterextras`).

New 3.33 They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE The generic unlocalized \TeX hooks are predefined, so that you can write:

```
\AddToHook{babel/*/afterextras}{\frenchspacing}
```

which is executed always after the extras for the language being selected (and just before the non-localized hooks defined with `\AddBabelHook`).

In addition, locale-specific hooks in the form `babel/<language-name>/<event-name>` are *recognized* (executed just before the localized babel hooks), but they are *not predefined*. You have to do it yourself. For example, to set `\frenchspacing` only in bengali:

```
\ActivateGenericHook{babel/bengali/afterextras}
\AddToHook{babel/bengali/afterextras}{\frenchspacing}
```


`\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)²⁰
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian upporsorbian
Welsh welsh

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

²⁰The two last name comes from the times when they had to be shortened to 8 characters

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

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

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

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{\z}{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 \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²¹ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is 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 \TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

²¹This explains why \LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

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.²²

But that is the easy part, because they don't require modifying the \LaTeX internals.

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

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

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 babel site for further details.

2 Loading languages with `language.dat`

\TeX and most engines based on it (pdf \TeX , xetex, $\epsilon\text{-}\TeX$, 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).²³ Until 3.9n, this task was delegated to the package `luatex-hyphen`, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).²⁴

²²See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to \TeX because their aim is just to display information and not fine typesetting.

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

²⁴The loader for lua(e)tex is slightly different as it's not based on babel but on `etex.src`. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the babel way, i.e., with `language.dat`.

2.1 Format

In that file the person who maintains a T_EX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored²⁵. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

```
% File      : language.dat
% Purpose   : tell iniTeX what files with patterns to load.
english    english.hyphenations
=british

dutch      hyphen.dutch exceptions.dutch % Nederlands
german     hyphen.ger
```

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.²⁶ For example:

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

With the previous settings, if the encoding when the language is selected is T1 then the patterns in hyphenT1.ger are used, but otherwise use those in hyphen.ger (note the encoding 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 T_EX users, so the files have to be coded so that they can be read by both L^AT_EX and plain T_EX. 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.

²⁵This is because different operating systems sometimes use very different file-naming conventions.

²⁶This is not a new feature, but in former versions it didn't work correctly.

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

Some recommendations:

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

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

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.

²⁷But 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>`.

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

\main@language To postpone the activation of the definitions needed for a language until the beginning of a

document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.

\ProvidesLanguage The macro `\ProvidesLanguage` should be used to identify the language definition files. Its syntax is similar to the syntax of the \TeX command `\ProvidesPackage`.

\LdfInit The macro `\LdfInit` 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 `@`-sign, preventing the `.ldf` file from being processed twice, etc.

\ldf@quit The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes resetting the category code of the `@`-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.

\ldf@finish The macro `\ldf@finish` does work needed at the end of each `.ldf` file. This includes resetting the category code of the `@`-sign, loading a local configuration file, and preparing the language to be activated at `\begin{document}` time.

\loadlocalcfg 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 `\captions{lang}` to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by `\ldf@finish`.

\substitutefontfamily (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 `.fd` 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\monthinname{<name of first month>}
% More strings

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

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



```
% More strings

\EndBabelCommands

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

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

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

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

3.4 Support for active characters

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

- `\initiate@active@char` The internal macro `\initiate@active@char` is used in language definition files to instruct \TeX to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.
- `\bbl@activate` 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 this²⁸.

- `\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,

²⁸This mechanism was introduced by Bernd Raichle.

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

3.8 Encoding-dependent strings

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

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

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

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

\StartBabelCommands `{<language-list>}{<category>}[<selector>]`

The *<language-list>* specifies which languages the block is intended for. A block is taken into account only if the `\CurrentOption` is listed here. Alternatively, you can define `\BabelLanguages` to a comma-separated list of languages to be defined (if undefined,

`\StartBabelCommands` sets it to `\CurrentOption`). You may write `\CurrentOption` as the language, but this is discouraged – a explicit name (or names) is much better and clearer. A “selector” is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name `unicode` must be used for `xetex` and `luatex` (the key `strings` has also other two special values: `generic` and `encoded`). If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

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

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

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

The `<category>` is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.²⁹ It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiname{M\"a\"rz}
```

²⁹In future releases further categories may be added.

```

\SetString\monthivname{April}
\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
\SetString\monthviiname{Juli}
\SetString\monthviiiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

\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.³⁰

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands `{\langle code \rangle}`

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

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

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

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

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

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

```

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

```

#1 is replaced by the roman numeral.

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

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

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 `\@uc1clist` (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`I=\I\relax}
  {\lccode`i=\i\relax
   \lccode`I=\I\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 though the given uppercase codes, using the step, and assigns them the `\lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO` *{<uccode-from>}{<uccode-to>}{<step>}{<lccode>}* loops though the given uppercase codes, using the step, and assigns them the `\lccode`, which is fixed (MO stands for *many-to-one*).

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

```
\SetHyphenMap{\BabelLowerMM{"100}{\I1F}{2}{\I01}}
```

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.

3.9 Executing code based on the selector

`\IfBabelSelectorTF {<selectors>}{<true>}{<false>}`

New 3.67 Sometimes a different setup is desired depending on the selector used. Values allowed in `<selectors>` are `select`, `other`, `foreign`, `other*` (and also `foreign*` for the tentative starred version), and it can consist of a comma-separated list. For example:

```
\IfBabelSelectorTF{other, other*}{A}{B}
```

is true with these two environment selectors.
Its natural place of use is in hooks or in `\extras<language>`.

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>).

4 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

5 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, MMM for the month name) and anything outside is text. In addition, [] is a non breakable space and [.] is an abbreviation dot.

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

6 Tools

```
1 <<version=3.80.2869>>
2 <<date=2022/09/23>>
```

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

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

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

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

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

\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the \else and \fi parts of an \if-statement³¹. These macros will break if another \if... \fi statement appears in one of the arguments and it is not enclosed in braces.

```
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, \<.> for \noexpand applied to a built macro name (which does not define the macro if undefined to \relax, because it is created locally), and \[...] for

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

one-level expansion (where . . is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```

29 \def\bbl@exp#1{%
30   \begingroup
31   \let\<\bbl@exp@en
32   \let\[\bbl@exp@ue
33   \edef\bbl@exp@aux{\endgroup#1}%
34   \bbl@exp@aux}
35 \def\bbl@exp@en#1>{\expandafter\<\noexpand\csname#1\endcsname}%
36 \def\bbl@exp@ue#1]{%
37   \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

\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.

```

39 \def\bbl@tempa#1{%
40   \long\def\bbl@trim##1##2{%
41     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
42   \def\bbl@trim@c{%
43     \ifx\bbl@trim@a\@sptoken
44       \expandafter\bbl@trim@b
45     \else
46       \expandafter\bbl@trim@b\expandafter#1%
47     \fi}%
48   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
49 \bbl@tempa{ }
50 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
51 \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 does not waste memory.

```

52 \begingroup
53   \gdef\bbl@ifunset#1{%
54     \expandafter\ifx\csname#1\endcsname\relax
55       \expandafter\@firstoftwo
56     \else
57       \expandafter\@secondoftwo
58     \fi}
59 \bbl@ifunset{ifcsname}% TODO. A better test?
60 {}%
61 {\gdef\bbl@ifunset#1{%
62   \ifcsname#1\endcsname
63     \expandafter\ifx\csname#1\endcsname\relax
64       \bbl@afterelse\expandafter\@firstoftwo
65     \else
66       \bbl@afterfi\expandafter\@secondoftwo
67     \fi
68   \else
69     \expandafter\@firstoftwo
70   \fi}}
71 \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,

```

72 \def\bbl@ifblank#1{%
73   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
74 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
75 \def\bbl@ifset#1#2#3{%
76   \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).

```

77 \def\bbl@forkv#1#2{%
78   \def\bbl@kvcmd##1##2##3{#2}%
79   \bbl@kvnext#1,\@nil,}
80 \def\bbl@kvnext#1,{%
81   \ifx\@nil#1\relax\else
82     \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
83     \expandafter\bbl@kvnext
84   \fi}
85 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
86   \bbl@trim@def\bbl@forkv@a{#1}%
87   \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).

```

88 \def\bbl@vforeach#1#2{%
89   \def\bbl@forcmd##1{#2}%
90   \bbl@fornext#1,\@nil,}
91 \def\bbl@fornext#1,{%
92   \ifx\@nil#1\relax\else
93     \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
94     \expandafter\bbl@fornext
95   \fi}
96 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

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

```

97 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
98   \toks@{}%
99   \def\bbl@replace@aux##1##2##2{%
100     \ifx\bbl@nil##2%
101       \toks@\expandafter{\the\toks@##1}%
102     \else
103       \toks@\expandafter{\the\toks@##1#3}%
104       \bbl@afterfi
105       \bbl@replace@aux##2##2%
106     \fi}%
107   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
108   \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 ckecking the replacement is really necessary or just paranoia).

```

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

```



```

125         \\makeatletter % "internal" macros with @ are assumed
126         \\scantokens{%
127             \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
128         \catcode64=\the\catcode64\relax}% Restore @
129     \else
130         \let\bbl@tempc\@empty % Not \relax
131     \fi
132     \bbl@exp{% For the 'uplevel' assignments
133 \endgroup
134     \bbl@tempc}} % empty or expand to set #1 with changes
135 \fi

```

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

```

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

```

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

```

158 \def\bbl@bsphack{%
159   \ifhmode
160     \hskip\z@skip
161     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
162   \else
163     \let\bbl@esphack\@empty
164   \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`.

```

165 \def\bbl@cased{%
166   \ifx\oe\OE
167     \expandafter\in@\expandafter
168     {\expandafter\OE\expandafter}\expandafter{\oe}%
169   \ifin@
170     \bbl@afterelse\expandafter\MakeUppercase
171   \else
172     \bbl@afterfi\expandafter\MakeLowercase
173   \fi
174 \else
175   \expandafter\@firstofone
176 \fi}

```

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

```

177 \ifx\IfFormatAtLeastTF\@undefined
178   \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
179 \else
180   \let\bbl@ifformatlater\IfFormatAtLeastTF
181 \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. Used to deal with `alph`, `Alph` and `frenchspacing` when there are already changes (with `\babel@save`).

```

182 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
183   \toks@{\expandafter\expandafter\expandafter{%
184     \csname extras\language\endcsname}%
185     \bbl@exp{\in@{#1}{\the\toks@}}%
186     \ifin@ \else
187       \@temptokena{#2}%
188       \edef\bbl@tempc{\the\@temptokena\the\toks@}%
189       \toks@\expandafter{\bbl@tempc#3}%
190       \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
191     \fi}
192 \</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 .

```

193 \<{*Make sure ProvidesFile is defined}> \equiv
194 \ifx\ProvidesFile\@undefined
195   \def\ProvidesFile#1[#2 #3 #4]{%
196     \wlog{File: #1 #4 #3 <#2>}%
197     \let\ProvidesFile\@undefined}
198 \fi
199 \</Make sure ProvidesFile is defined>

```

6.1 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.

```

200 \<{*Define core switching macros}> \equiv
201 \ifx\language\@undefined
202   \csname newcount\endcsname\language
203 \fi
204 \</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.

```

205 \<{*Define core switching macros}> \equiv
206 \countdef\last@language=19
207 \def\addlanguage{\csname newlanguage\endcsname}
208 \</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.

6.2 The Package File (LaTeX, babel.sty)

```

209 <*package>
210 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
211 \ProvidesPackage{babel}[<date>] <version> The Babel package]

```

Start with some “private” debugging tool, and then define macros for errors.

```

212 \@ifpackagewith{babel}{debug}
213   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
214   \let\bbl@debug\@firstofone
215   \ifx\directlua\@undefined\else
216     \directlua{ Babel = Babel or {}
217       Babel.debug = true }%
218     \input{babel-debug.tex}%
219   \fi}
220 {\providecommand\bbl@trace[1]}%
221 \let\bbl@debug\@gobble
222 \ifx\directlua\@undefined\else
223   \directlua{ Babel = Babel or {}
224     Babel.debug = false }%
225   \fi}
226 \def\bbl@error#1#2{%
227   \begingroup
228     \def\{\MessageBreak}%
229     \PackageError{babel}{#1}{#2}%
230   \endgroup}
231 \def\bbl@warning#1{%
232   \begingroup
233     \def\{\MessageBreak}%
234     \PackageWarning{babel}{#1}%
235   \endgroup}
236 \def\bbl@infowarn#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \PackageNote{babel}{#1}%
240   \endgroup}
241 \def\bbl@info#1{%
242   \begingroup
243     \def\{\MessageBreak}%
244     \PackageInfo{babel}{#1}%
245   \endgroup}

```

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. But first, include here the *Basic macros* defined above.

```

246 <<Basic macros>>
247 \@ifpackagewith{babel}{silent}
248   {\let\bbl@info\@gobble
249     \let\bbl@infowarn\@gobble
250     \let\bbl@warning\@gobble}
251   {}
252 %
253 \def\AfterBabelLanguage#1{%
254   \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.

```

255 \ifx\bbl@languages\@undefined\else
256   \begingroup
257     \catcode\^^I=12
258     \@ifpackagewith{babel}{showlanguages}{%
259       \begingroup

```

```

260     \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}}%
261     \wlog{<*languages>}%
262     \bbl@languages
263     \wlog{</languages>}%
264     \endgroup{}}
265 \endgroup
266 \def\bbl@elt#1#2#3#4{%
267     \ifnum#2=\z@
268         \gdef\bbl@nulllanguage{#1}%
269         \def\bbl@elt##1##2##3##4{%
270             \fi}%
271     \bbl@languages
272 \fi%

```

6.3 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`.

```

273 \bbl@trace{Defining option 'base'}
274 \@ifpackagewith{babel}{base}{%
275     \let\bbl@onlyswitch\@empty
276     \let\bbl@provide@locale\relax
277     \input babel.def
278     \let\bbl@onlyswitch\@undefined
279     \ifx\directlua\@undefined
280         \DeclareOption*{\bbl@patterns{\CurrentOption}}%
281     \else
282         \input luababel.def
283         \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
284     \fi
285     \DeclareOption{base}{}%
286     \DeclareOption{showlanguages}{}%
287     \ProcessOptions
288     \global\expandafter\let\csname opt@babel.sty\endcsname\relax
289     \global\expandafter\let\csname ver@babel.sty\endcsname\relax
290     \global\let@ifl@ter@@\@ifl@ter
291     \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\@ifl@ter@@}%
292     \endinput{}%

```

6.4 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no modifiers have been given, the former is `\relax`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\@for` or load `keyval`, for example.

```

293 \bbl@trace{key=value and another general options}
294 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
295 \def\bbl@tempb#1.#2{% Remove trailing dot
296     #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
297 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
298     \ifx\@empty#2%
299         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
300     \else
301         \in@{,provide=}{, #1}%
302         \ifin@
303             \edef\bbl@tempc{%
304                 \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
305         \else
306             \in@{=}{#1}%
307         \ifin@

```

```

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

```

318 \DeclareOption{KeepShorthandsActive}{}
319 \DeclareOption{activeacute}{}
320 \DeclareOption{activegrave}{}
321 \DeclareOption{debug}{}
322 \DeclareOption{noconfigs}{}
323 \DeclareOption{showlanguages}{}
324 \DeclareOption{silent}{}
325 % \DeclareOption{mono}{}
326 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
327 \chardef\bbl@iniflag\z@
328 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}    % main -> +1
329 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\@tw@}  % add = 2
330 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\@thr@} % add + main
331 % A separate option
332 \let\bbl@autoload@options\@empty
333 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
334 % Don't use. Experimental. TODO.
335 \newif\ifbbl@single
336 \DeclareOption{selectors=off}{\bbl@singletrue}
337 <(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.

```

338 \let\bbl@opt@shorthands\@nnil
339 \let\bbl@opt@config\@nnil
340 \let\bbl@opt@main\@nnil
341 \let\bbl@opt@headfoot\@nnil
342 \let\bbl@opt@layout\@nnil
343 \let\bbl@opt@provide\@nnil

```

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

```

344 \def\bbl@tempa#1=#2\bbl@tempa{%
345   \bbl@csarg\ifx{opt@#1}\@nnil
346   \bbl@csarg\edef{opt@#1}{#2}%
347   \else
348     \bbl@error
349     {Bad option '#1=#2'. Either you have misspelled the\\%
350     key or there is a previous setting of '#1'. Valid\\%
351     keys are, among others, 'shorthands', 'main', 'bidi',\\%
352     'strings', 'config', 'headfoot', 'safe', 'math'.}%
353     {See the manual for further details.}
354   \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.

```

355 \let\bbl@language@opts\@empty
356 \DeclareOption*{%

```

```

357 \bbl@xin@{\string=}{\CurrentOption}%
358 \ifin@
359 \expandafter\bbl@tempa\CurrentOption\bbl@tempa
360 \else
361 \bbl@add@list\bbl@language@opts{\CurrentOption}%
362 \fi}

```

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

```

363 \ProcessOptions*
364 \ifx\bbl@opt@provide\@nnil
365 \let\bbl@opt@provide\@empty %%%% MOVE above
366 \else
367 \chardef\bbl@iniflag\@ne
368 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
369 \in@{,provide,}{, #1,}%
370 \ifin@
371 \def\bbl@opt@provide{#2}%
372 \bbl@replace\bbl@opt@provide{;}{,}%
373 \fi}
374 \fi
375 %

```

6.5 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=...

```

376 \bbl@trace{Conditional loading of shorthands}
377 \def\bbl@sh@string#1{%
378 \ifx#1\@empty\else
379 \ifx#1t\string~%
380 \else\ifx#1c\string,%
381 \else\string#1%
382 \fi\fi
383 \expandafter\bbl@sh@string
384 \fi}
385 \ifx\bbl@opt@shorthands\@nnil
386 \def\bbl@ifshorthand#1#2#3{#2}%
387 \else\ifx\bbl@opt@shorthands\@empty
388 \def\bbl@ifshorthand#1#2#3{#3}%
389 \else

```

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

```

390 \def\bbl@ifshorthand#1{%
391 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
392 \ifin@
393 \expandafter\@firstoftwo
394 \else
395 \expandafter\@secondoftwo
396 \fi}

```

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

```

397 \edef\bbl@opt@shorthands{%
398 \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.

```

399 \bbl@ifshorthand{'}%
400 {\PassOptionsToPackage{activeacute}{babel}}{}
401 \bbl@ifshorthand{`}%
402 {\PassOptionsToPackage{activegrave}{babel}}{}
403 \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.

```
404 \ifx\bbl@opt@headfoot\@nnil\else
405   \g@addto@macro\@resetactivechars{%
406     \set@typeset@protect
407     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
408     \let\protect\noexpand}
409 \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 currently set, but in a future release it will be set to none.

```
410 \ifx\bbl@opt@safe\@undefined
411   \def\bbl@opt@safe{BR}
412   % \let\bbl@opt@safe\empty % Pending of \cite
413 \fi
```

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

```
414 \bbl@trace{Defining IfBabelLayout}
415 \ifx\bbl@opt@layout\@nnil
416   \newcommand\IfBabelLayout[3]{#3}%
417 \else
418   \newcommand\IfBabelLayout[1]{%
419     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
420     \ifin@
421       \expandafter\@firstoftwo
422     \else
423       \expandafter\@secondoftwo
424     \fi}
425 \fi
426 \</package>
427 \<core>
```

6.6 Interlude for Plain

Because of the way `docstrip` works, we need to insert some code for Plain here. However, the tools provided by the babel installer for literate programming makes this section a short interlude, because the actual code is below, tagged as *Emulate LaTeX*.

```
428 \ifx\ldf@quit\@undefined\else
429 \endinput\fi % Same line!
430 \<Make sure ProvidesFile is defined>
431 \ProvidesFile{babel.def}[\<date>] [\<version>] Babel common definitions]
432 \ifx\AtBeginDocument\@undefined % TODO. change test.
433   \<Emulate LaTeX>
434 \fi
```

That is all for the moment. Now follows some common stuff, for both Plain and \LaTeX . After it, we will resume the \LaTeX -only stuff.

```
435 \</core>
436 \<package | core>
```

7 Multiple languages

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

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.

```
437 \def\bbl@version{\<version>}
438 \def\bbl@date{\<date>}
439 \<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.

```

440 \def\adddialect#1#2{%
441   \global\chardef#1#2\relax
442   \bbl@usehooks{adddialect}{#1}{#2}}%
443   \begingroup
444     \count@#1\relax
445     \def\bbl@elt##1##2##3##4{%
446       \ifnum\count@=##2\relax
447         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
448         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
449           set to \expandafter\string\csname l@##1\endcsname\%
450           (\string\language\the\count@). Reported}%
451         \def\bbl@elt####1####2####3####4{%
452           \fi}%
453         \bbl@cs{languages}%
454       \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.

```

455 \def\bbl@fixname#1{%
456   \begingroup
457   \def\bbl@tempe{l@}%
458   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
459   \bbl@tempd
460     {\lowercase\expandafter{\bbl@tempd}%
461     {\uppercase\expandafter{\bbl@tempd}%
462     \@empty
463     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
464     {\uppercase\expandafter{\bbl@tempd}}}%
465     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
466     {\lowercase\expandafter{\bbl@tempd}}}%
467     \@empty
468     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
469   \bbl@tempd
470   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
471 \def\bbl@iflanguage#1{%
472   \@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`.

```

473 \def\bbl@bcpcase#1#2#3#4\@#5{%
474   \ifx\@empty#3%
475     \uppercase{\def#5{#1#2}}%
476   \else
477     \uppercase{\def#5{#1}}%
478     \lowercase{\edef#5{#5#2#3#4}}%
479   \fi}
480 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
481   \let\bbl@bcp\relax
482   \lowercase{\def\bbl@tempa{#1}}%
483   \ifx\@empty#2%
484     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
485   \else\ifx\@empty#3%
486     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
487     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%

```



```

488     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
489     }%
490     \ifx\bbl@bcp\relax
491       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
492     \fi
493   \else
494     \bbl@bcpcase#2\@empty\@empty\@empty\bbl@tempb
495     \bbl@bcpcase#3\@empty\@empty\@empty\bbl@tempc
496     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
497       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
498     }%
499     \ifx\bbl@bcp\relax
500       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
501       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
502     }%
503   \fi
504   \ifx\bbl@bcp\relax
505     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
506     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
507   }%
508 \fi
509 \ifx\bbl@bcp\relax
510   \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
511 \fi
512 \fi\fi}
513 \let\bbl@initoload\relax
514 \def\bbl@provide@locale{%
515   \ifx\babelprovide\undefined
516     \bbl@error{For a language to be defined on the fly 'base'\\%
517               is not enough, and the whole package must be\\%
518               loaded. Either delete the 'base' option or\\%
519               request the languages explicitly}%
520     {See the manual for further details.}%
521   \fi
522 % TODO. Option to search if loaded, with \LocaleForEach
523 \let\bbl@auxname\language % Still necessary. TODO
524 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
525 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
526 \ifbbl@bcpallowed
527   \expandafter\ifx\csname date\language\endcsname\relax
528     \expandafter
529     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@
530     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
531       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
532       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
533       \expandafter\ifx\csname date\language\endcsname\relax
534         \let\bbl@initoload\bbl@bcp
535         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
536         \let\bbl@initoload\relax
537       \fi
538       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
539     \fi
540   \fi
541 \fi
542 \expandafter\ifx\csname date\language\endcsname\relax
543   \IfFileExists{babel-\language.tex}%
544   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
545   {}%
546 \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.

```

547 \def\iflanguage#1{%
548   \bbl@iflanguage{#1}{%
549     \ifnum\csname l@#1\endcsname=\language
550       \expandafter\@firstoftwo
551     \else
552       \expandafter\@secondoftwo
553     \fi}}

```

7.1 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.

```

554 \let\bbl@select@type\z@
555 \edef\selectlanguage{%
556   \noexpand\protect
557   \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`.

```
558 \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.

```
559 \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.

```
560 \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`

```

561 \def\bbl@push@language{%
562   \ifx\languagename\undefined\else
563     \ifx\currentgrouplevel\undefined
564       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
565     \else
566       \ifnum\currentgrouplevel=\z@
567         \xdef\bbl@language@stack{\languagename+}%
568       \else
569         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
570       \fi
571     \fi
572   \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 `\languagename`. For this we first define a helper function.

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

```
573 \def\bbl@pop@lang#1+#2\@@{%
574   \edef\language{#1}%
575   \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).

```
576 \let\bbl@ifrestoring\@secondoftwo
577 \def\bbl@pop@language{%
578   \expandafter\bbl@pop@lang\bbl@language@stack\@@
579   \let\bbl@ifrestoring\@firstoftwo
580   \expandafter\bbl@set@language\expandafter{\language}%
581   \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).

```
582 \chardef\localeid\z@
583 \def\bbl@id@last{0} % No real need for a new counter
584 \def\bbl@id@assign{%
585   \bbl@ifunset\bbl@id@@\language}%
586   {\count@\bbl@id@last\relax
587    \advance\count@\@ne
588    \bbl@csarg\chardef{id@@\language}\count@
589    \edef\bbl@id@last{\the\count@}%
590    \ifcase\bbl@engine\or
591      \directlua{
592        Babel = Babel or {}
593        Babel.locale_props = Babel.locale_props or {}
594        Babel.locale_props[\bbl@id@last] = {}
595        Babel.locale_props[\bbl@id@last].name = '\language'
596      }%
597    \fi}%
598  {}%
599  \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of `\selectlanguage`.

```
600 \expandafter\def\csname selectlanguage \endcsname#1{%
601   \ifnum\bbl@hymapset=\@ccclv\let\bbl@hymapset\tw@\fi
602   \bbl@push@language
603   \aftergroup\bbl@pop@language
604   \bbl@set@language{#1}}
```

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

We also write a command to change the current language in the auxiliary files.

`\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).

```

605 \def\BabelContentsFiles{toc,lof,lot}
606 \def\bbl@set@language#1{% from selectlanguage, pop@
607 % The old buggy way. Preserved for compatibility.
608 \edef\language{%
609 \ifnum\escapechar=\expandafter`\string#1\@empty
610 \else\string#1\@empty\fi}%
611 \ifcat\relax\noexpand#1%
612 \expandafter\ifx\csname date\language\endcsname\relax
613 \edef\language{#1}%
614 \let\locale\language
615 \else
616 \bbl@info{Using '\string\language' instead of 'language' is\\%
617 deprecated. If what you want is to use a\\%
618 macro containing the actual locale, make\\%
619 sure it does not not match any language.\\%
620 Reported}%
621 \ifx\scantokens\@undefined
622 \def\locale{??}%
623 \else
624 \scantokens\expandafter{\expandafter
625 \def\expandafter\locale\expandafter{\language}}%
626 \fi
627 \fi
628 \else
629 \def\locale{#1}% This one has the correct catcodes
630 \fi
631 \select@language{\language}%
632 % write to aux
633 \expandafter\ifx\csname date\language\endcsname\relax\else
634 \if@files
635 \ifx\babel@aux\@gobbles\else % Set if single in the first, redundant
636 \bbl@savelastskip
637 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
638 \bbl@restorelastskip
639 \fi
640 \bbl@usehooks{write}{}%
641 \fi
642 \fi}
643 %
644 \let\bbl@restorelastskip\relax
645 \let\bbl@savelastskip\relax
646 %
647 \newif\ifbbl@bcpallowed
648 \bbl@bcpallowedfalse
649 \def\select@language#1{% from set@, babel@aux
650 \ifx\bbl@selectname\@empty
651 \def\bbl@selectname{select}%
652 % set hymap
653 \fi
654 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
655 % set name
656 \edef\language{#1}%
657 \bbl@fixname\language
658 % TODO. name@map must be here?
659 \bbl@provide@locale
660 \bbl@iflanguage\language{%
661 \expandafter\ifx\csname date\language\endcsname\relax
662 \bbl@error
663 {Unknown language '\language'. Either you have\\%
664 misspelled its name, it has not been installed,\\%
665 or you requested it in a previous run. Fix its name,\\%
666 install it or just rerun the file, respectively. In\\%
667 some cases, you may need to remove the aux file}%

```

```

668         {You may proceed, but expect wrong results}%
669     \else
670         % set type
671         \let\bbl@select@type\z@
672         \expandafter\bbl@switch\expandafter{\language}%
673     \fi}}
674 \def\babel@aux#1#2{%
675     \select@language{#1}%
676     \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
677         \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
678 \def\babel@toc#1#2{%
679     \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.

```

680 \newif\ifbbl@usedategroup
681 \def\bbl@switch#1{% from select@, foreign@
682     % make sure there is info for the language if so requested
683     \bbl@ensureinfo{#1}%
684     % restore
685     \originalTeX
686     \expandafter\def\expandafter\originalTeX\expandafter{%
687         \csname noextras#1\endcsname
688         \let\originalTeX\@empty
689         \babel@beginsave}%
690     \bbl@usehooks{afterreset}}}%
691     \languageshorthands{none}%
692     % set the locale id
693     \bbl@id@assign
694     % switch captions, date
695     % No text is supposed to be added here, so we remove any
696     % spurious spaces.
697     \bbl@bsphack
698     \ifcase\bbl@select@type
699         \csname captions#1\endcsname\relax
700         \csname date#1\endcsname\relax
701     \else
702         \bbl@xin@{,captions,}{, \bbl@select@opts,}%
703         \ifin@
704             \csname captions#1\endcsname\relax
705         \fi
706         \bbl@xin@{,date,}{, \bbl@select@opts,}%
707         \ifin@ % if \foreign... within \<lang>date
708             \csname date#1\endcsname\relax
709         \fi
710     \fi
711     \bbl@esphack
712     % switch extras
713     \bbl@usehooks{beforeextras}}}%
714     \csname extras#1\endcsname\relax
715     \bbl@usehooks{afterextras}}}%
716     % > babel-ensure
717     % > babel-sh-<short>

```

```

718 % > babel-bidi
719 % > babel-fontspec
720 % hyphenation - case mapping
721 \ifcase\bbbl@opt@hyphenmap\or
722   \def\BabelLower##1##2{\lccode##1=##2\relax}%
723   \ifnum\bbbl@hymapsel>4\else
724     \csname\language\name @bbbl@hyphenmap\endcsname
725   \fi
726   \chardef\bbbl@opt@hyphenmap\z@
727 \else
728   \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
729     \csname\language\name @bbbl@hyphenmap\endcsname
730   \fi
731 \fi
732 \let\bbbl@hymapsel\@cclv
733 % hyphenation - select rules
734 \ifnum\csname l@language\endcsname=\l@unhyphenated
735   \edef\bbbl@tempa{u}%
736 \else
737   \edef\bbbl@tempa{\bbbl@cl{lbrk}}%
738 \fi
739 % linebreaking - handle u, e, k (v in the future)
740 \bbbl@xin@{/u}{/\bbbl@tempa}%
741 \ifin@{\else\bbbl@xin@{/e}}{\bbbl@tempa}\fi % elongated forms
742 \ifin@{\else\bbbl@xin@{/k}}{\bbbl@tempa}\fi % only kashida
743 \ifin@{\else\bbbl@xin@{/v}}{\bbbl@tempa}\fi % variable font
744 \ifin@
745   % unhyphenated/kashida/elongated = allow stretching
746   \language\l@unhyphenated
747   \babel@savevariable\emergencystretch
748   \emergencystretch\maxdimen
749   \babel@savevariable\hbadness
750   \hbadness\@M
751 \else
752   % other = select patterns
753   \bbbl@patterns{#1}%
754 \fi
755 % hyphenation - mins
756 \babel@savevariable\lefthyphenmin
757 \babel@savevariable\righthyphenmin
758 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
759   \set@hyphenmins\tw@\thr@@\relax
760 \else
761   \expandafter\expandafter\expandafter\set@hyphenmins
762     \csname #1hyphenmins\endcsname\relax
763 \fi
764 \let\bbbl@selectorname\@empty}

```

`otherlanguage (env.)` 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.

```

765 \long\def\otherlanguage#1{%
766   \def\bbbl@selectorname{other}%
767   \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@@\fi
768   \csname selectlanguage \endcsname{#1}%
769   \ignorespaces}

```

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

```

770 \long\def\endotherlanguage{%

```

```
771 \global\@ignoretrue\ignorespaces}
```

`otherlanguage*` (*env.*) 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`.

```
772 \expandafter\def\csname otherlanguage*\endcsname{%
773   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
774 \def\bbl@otherlanguage@s[#1]#2{%
775   \def\bbl@selectorname{other*}%
776   \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
777   \def\bbl@select@opts{#1}%
778   \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”.

```
779 \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`.

```
780 \providecommand\bbl@beforeforeign{}
781 \edef\foreignlanguage{%
782   \noexpand\protect
783   \expandafter\noexpand\csname foreignlanguage \endcsname}
784 \expandafter\def\csname foreignlanguage \endcsname{%
785   \@ifstar\bbl@foreign@s\bbl@foreign@x}
786 \providecommand\bbl@foreign@x[3][{}]{%
787   \begingroup
788     \def\bbl@selectorname{foreign}%
789     \def\bbl@select@opts{#1}%
790     \let\BabelText\@firstofone
791     \bbl@beforeforeign
792     \foreign@language{#2}%
793     \bbl@usehooks{foreign}{}%
794     \BabelText{#3}% Now in horizontal mode!
795   \endgroup}
796 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
797   \begingroup
798     {\par}%
799     \def\bbl@selectorname{foreign*}%
800     \let\bbl@select@opts\@empty
801     \let\BabelText\@firstofone
802     \foreign@language{#1}%
803     \bbl@usehooks{foreign*}{}%
804     \bbl@dirparastext
```

```

805 \BabelText{#2}% Still in vertical mode!
806 {\par}%
807 \endgroup}

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the other `language*` 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`.

```

808 \def\foreign@language#1{%
809 % set name
810 \edef\language#1}%
811 \ifbbl@usedatagroup
812 \bbl@add\bbl@select@opts{,date,}%
813 \bbl@usedatagroupfalse
814 \fi
815 \bbl@fixname\language
816 % TODO. name@map here?
817 \bbl@provide@locale
818 \bbl@iflanguage\language{%
819 \expandafter\ifx\csname date\language\endcsname\relax
820 \bbl@warning % TODO - why a warning, not an error?
821 {Unknown language '#1'. Either you have\\%
822 misspelled its name, it has not been installed,\\%
823 or you requested it in a previous run. Fix its name,\\%
824 install it or just rerun the file, respectively. In\\%
825 some cases, you may need to remove the aux file.\\%
826 I'll proceed, but expect wrong results.\\%
827 Reported}%
828 \fi
829 % set type
830 \let\bbl@select@type\@ne
831 \expandafter\bbl@switch\expandafter{\language}}

```

The following macro executes conditionally some code based on the selector being used.

```

832 \def\IfBabelSelectorTF#1{%
833 \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
834 \ifin@
835 \expandafter\@firstoftwo
836 \else
837 \expandafter\@secondoftwo
838 \fi}

```

`\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*` 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.

```

839 \let\bbl@hyphlist\@empty
840 \let\bbl@hyphenation@\relax
841 \let\bbl@pttnlist\@empty
842 \let\bbl@patterns@\relax
843 \let\bbl@hymapsel=\cclv
844 \def\bbl@patterns#1{%
845 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
846 \csname l@#1\endcsname
847 \edef\bbl@tempa{#1}%
848 \else
849 \csname l@#1:\f@encoding\endcsname
850 \edef\bbl@tempa{#1:\f@encoding}%
851 \fi
852 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}

```



```

853 % > luatex
854 \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
855 \begingroup
856 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
857 \ifin@else
858 \expandafter\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
859 \hyphenation{%
860 \bbl@hyphenation@
861 \@ifundefined{bbl@hyphenation@#1}%
862 \@empty
863 {\space\csname bbl@hyphenation@#1\endcsname}}%
864 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
865 \fi
866 \endgroup}}

```

`hyphenrules` (*env.*) 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, `\lcode`'s and font encodings are not set at all, so in most cases you should use `otherlanguage*`.

```

867 \def\hyphenrules#1{%
868 \edef\bbl@tempf{#1}%
869 \bbl@fixname\bbl@tempf
870 \bbl@iflanguage\bbl@tempf{%
871 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
872 \ifx\languageshorthands\undefined\else
873 \languageshorthands{none}%
874 \fi
875 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
876 \set@hyphenmins\tw@\thr@@\relax
877 \else
878 \expandafter\expandafter\expandafter\set@hyphenmins
879 \csname\bbl@tempf hyphenmins\endcsname\relax
880 \fi}}
881 \let\endhyphenrules\@empty

```

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

```

882 \def\providehyphenmins#1#2{%
883 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
884 \@namedef{#1hyphenmins}{#2}%
885 \fi}

```

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

```

886 \def\set@hyphenmins#1#2{%
887 \lefthyphenmin#1\relax
888 \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.

```

889 \ifx\ProvidesFile\undefined
890 \def\ProvidesLanguage#1[#2 #3 #4]{%
891 \wlog{Language: #1 #4 #3 <#2>}%
892 }
893 \else
894 \def\ProvidesLanguage#1{%
895 \begingroup
896 \catcode`\ 10 %
897 \@makeother\/%

```

```

898 \ifnextchar[%]
899 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
900 \def\@provideslanguage#1[#2]{%
901 \wlog{Language: #1 #2}%
902 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
903 \endgroup}
904 \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`.

```
905 \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.

```
906 \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’:

```

907 \providecommand\setlocale{%
908 \bbl@error
909 {Not yet available}%
910 {Find an armchair, sit down and wait}}
911 \let\uselocale\setlocale
912 \let\locale\setlocale
913 \let\selectlocale\setlocale
914 \let\textlocale\setlocale
915 \let\textlanguage\setlocale
916 \let\languagegettext\setlocale

```

7.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.
When the format knows about `\PackageError` it must be $\LaTeX 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.
Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

917 \edef\bbl@nulllanguage{\string\language=0}
918 \def\bbl@nocaption{\protect\bbl@nocaption@i}
919 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
920 \global\@namedef{#2}{\textbf{?#1?}}}%
921 \@nameuse{#2}%
922 \edef\bbl@tempa{#1}%
923 \bbl@sreplace\bbl@tempa{name}{}}%
924 \bbl@warning{% TODO.
925 \@backslashchar#1 not set for '\language'. Please,\\%
926 define it after the language has been loaded\\%
927 (typically in the preamble) with:\\%
928 \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
929 Feel free to contribute on github.com/latex3/babel.\\%
930 Reported}}
931 \def\bbl@tentative{\protect\bbl@tentative@i}
932 \def\bbl@tentative@i#1{%
933 \bbl@warning{%
934 Some functions for '#1' are tentative.\\%
935 They might not work as expected and their behavior\\%
936 could change in the future.\\%
937 Reported}}
938 \def\@nolanerr#1{%

```

```

939 \bbl@error
940 {You haven't defined the language '#1' yet.\\%
941   Perhaps you misspelled it or your installation\\%
942   is not complete}%
943 {Your command will be ignored, type <return> to proceed}}
944 \def\bbl@nopatterns#1{%
945   \bbl@warning
946   {No hyphenation patterns were preloaded for\\%
947     the language '#1' into the format.\\%
948     Please, configure your TeX system to add them and\\%
949     rebuild the format. Now I will use the patterns\\%
950     preloaded for \bbl@nulllanguage\space instead}}
951 \let\bbl@usehooks\@gobbletwo
952 \ifx\bbl@onlyswitch\@empty\endinput\fi
953 % Here ended switch.def

```

Here ended the now discarded switch.def. Here also (currently) ends the base option.

```

954 \ifx\directlua\@undefined\else
955   \ifx\bbl@luapatterns\@undefined
956     \input luababel.def
957   \fi
958 \fi
959 <Basic macros>
960 \bbl@trace{Compatibility with language.def}
961 \ifx\bbl@languages\@undefined
962   \ifx\directlua\@undefined
963     \openin1 = language.def % TODO. Remove hardcoded number
964     \ifeof1
965       \closein1
966       \message{I couldn't find the file language.def}
967     \else
968       \closein1
969       \begingroup
970         \def\addlanguage#1#2#3#4#5{%
971           \expandafter\ifx\csname lang@#1\endcsname\relax\else
972             \global\expandafter\let\csname l@#1\endcsname
973             \csname lang@#1\endcsname
974           \fi}%
975         \def\uselanguage#1{%
976           \input language.def
977         \endgroup
978       \fi
979     \fi
980     \chardef\l@english\z@
981 \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.

```

982 \def\addto#1#2{%
983   \ifx#1\@undefined
984     \def#1{#2}%
985   \else
986     \ifx#1\relax
987       \def#1{#2}%
988     \else
989       {\toks@\expandafter{#1#2}%
990        \xdef#1{\the\toks@}}%
991     \fi
992   \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?

```
993 \def\bbl@withactive#1#2{%
994   \begingroup
995     \lccode`~=`#2\relax
996     \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`.

```
997 \def\bbl@redefine#1{%
998   \edef\bbl@tempa{\bbl@stripslash#1}%
999   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1000   \expandafter\def\csname\bbl@tempa\endcsname{
1001     \@onlypreamble\bbl@redefine
```

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

```
1002 \def\bbl@redefine@long#1{%
1003   \edef\bbl@tempa{\bbl@stripslash#1}%
1004   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1005   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1006     \@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_`.

```
1007 \def\bbl@redefineroobust#1{%
1008   \edef\bbl@tempa{\bbl@stripslash#1}%
1009   \bbl@ifunset{\bbl@tempa\space}%
1010     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1011       \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1012     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1013     \@namedef{\bbl@tempa\space}%
1014     \@onlypreamble\bbl@redefineroobust
```

7.3 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.

```
1015 \bbl@trace{Hooks}
1016 \newcommand\AddBabelHook[3][]{%
1017   \bbl@ifunset{\bbl@hk##2}{\EnableBabelHook{#2}}}%
1018   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1019   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1020   \bbl@ifunset{\bbl@ev##2@#3@#1}%
1021     {\bbl@csarg\bbl@add{ev##3@#1}{\bbl@elth{#2}}}%
1022     {\bbl@csarg\let{ev##2@#3@#1}\relax}%
1023   \bbl@csarg\newcommand{ev##2@#3@#1}{\bbl@tempb}}
1024 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk##1}\@firstofone}
1025 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk##1}\@gobble}
1026 \def\bbl@usehooks#1#2{%
1027   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1028   \def\bbl@elth##1{%
1029     \bbl@cs{hk##1}{\bbl@cs{ev##1@#1@#2}}%
1030     \bbl@cs{ev##1@#1}%
1031     \ifx\language\@undefined\else % Test required for Plain (?)
1032       \ifx\UseHook\@undefined\else\UseHook{babel/\language/#1}\fi
1033       \def\bbl@elth##1{%
1034         \bbl@cs{hk##1}{\bbl@c1{ev##1@#1@#2}}%
```

```

1035 \bbl@c1{ev@#1}%
1036 \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).

```

1037 \def\bbl@evargs{,% <- don't delete this comma
1038 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1039 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1040 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1041 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1042 beforestart=0,languagename=2}
1043 \ifx\NewHook\@undefined\else
1044 \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1045 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}%
1046 \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.

```

1047 \bbl@trace{Defining babelensure}
1048 \newcommand\babelensure[2][{}% TODO - revise test files
1049 \AddBabelHook{babel-ensure}{afterextras}{%
1050 \ifcase\bbl@select@type
1051 \bbl@c1{e}%
1052 \fi}%
1053 \begingroup
1054 \let\bbl@ens@include\@empty
1055 \let\bbl@ens@exclude\@empty
1056 \def\bbl@ens@fontenc{\relax}%
1057 \def\bbl@tempb##1{%
1058 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1059 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1060 \def\bbl@tempb##1=#2\@{\@namedef{\bbl@ens@##1}{##2}}%
1061 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1062 \def\bbl@tempc{\bbl@ensure}%
1063 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1064 \expandafter{\bbl@ens@include}}%
1065 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1066 \expandafter{\bbl@ens@exclude}}%
1067 \toks@\expandafter{\bbl@tempc}%
1068 \bbl@exp{%
1069 \endgroup
1070 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1071 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1072 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist
1073 \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1074 \edef##1{\noexpand\bbl@nocaption
1075 {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1076 \fi
1077 \ifx##1\@empty\else
1078 \in@{##1}{#2}%
1079 \ifin@\else
1080 \bbl@ifunset{\bbl@ensure@\languagename}%
1081 {\bbl@exp{%
1082 \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1083 \\\foreignlanguage{\languagename}%

```

```

1084         {\ifx\relax#3\else
1085           \\fontencoding{#3}\\selectfont
1086           \fi
1087           #####1}}}%
1088     }%
1089     \toks@\expandafter{##1}%
1090     \edef##1{%
1091       \bbl@csarg\noexpand{ensure@\language}%
1092       {\the\toks@}}%
1093     \fi
1094     \expandafter\bbl@tempb
1095     \fi}%
1096 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1097 \def\bbl@tempa##1{% elt for include list
1098   \ifx##1\@empty\else
1099     \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1100     \ifin@\else
1101       \bbl@tempb##1\@empty
1102     \fi
1103     \expandafter\bbl@tempa
1104     \fi}%
1105   \bbl@tempa#1\@empty}
1106 \def\bbl@captionslist{%
1107   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1108   \contentsname\listfigurename\listtablename\indexname\figurename
1109   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1110   \alsoname\proofname\glossaryname}

```

7.4 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`.

```

1111 \bbl@trace{Macros for setting language files up}
1112 \def\bbl@ldfinit{%
1113   \let\bbl@sreset\@empty
1114   \let\BabelStrings\bbl@opt@string
1115   \let\BabelOptions\@empty
1116   \let\BabelLanguages\relax
1117   \ifx\originalTeX\@undefined
1118     \let\originalTeX\@empty
1119   \else
1120     \originalTeX
1121   \fi}
1122 \def\LdfInit#1#2{%
1123   \chardef\atcatcode=\catcode`\@
1124   \catcode`\@=11\relax
1125   \chardef\eqcatcode=\catcode`\=

```

```

1126 \catcode`\==12\relax
1127 \expandafter\if\expandafter\@backslashchar
1128         \expandafter\@car\string#2\@nil
1129     \ifx#2\@undefined\else
1130         \ldf@quit{#1}%
1131     \fi
1132 \else
1133     \expandafter\ifx\csname#2\endcsname\relax\else
1134         \ldf@quit{#1}%
1135     \fi
1136 \fi
1137 \bbl@ldfinit}

```

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

```

1138 \def\ldf@quit#1{%
1139     \expandafter\main@language\expandafter{#1}%
1140     \catcode`\@=\atcatcode \let\atcatcode\relax
1141     \catcode`\==\eqcatcode \let\eqcatcode\relax
1142     \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.

```

1143 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1144     \bbl@afterlang
1145     \let\bbl@afterlang\relax
1146     \let\BabelModifiers\relax
1147     \let\bbl@screset\relax}%
1148 \def\ldf@finish#1{%
1149     \loadlocalcfg{#1}%
1150     \bbl@afterldf{#1}%
1151     \expandafter\main@language\expandafter{#1}%
1152     \catcode`\@=\atcatcode \let\atcatcode\relax
1153     \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 `ltxex`.

```

1154 \@onlypreamble\LdfInit
1155 \@onlypreamble\ldf@quit
1156 \@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.

```

1157 \def\main@language#1{%
1158     \def\bbl@main@language{#1}%
1159     \let\language\bbl@main@language % TODO. Set locale name
1160     \bbl@id@assign
1161     \bbl@patterns{\language}

```

We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the `\AtBeginDocument` is executed. Languages do not set `\pagedir`, so we set here for the whole document to the main `\bodydir`.

```

1162 \def\bbl@beforestart{%
1163     \def\@nolanerr##1{%
1164         \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1165     \bbl@usehooks{beforestart}{}%
1166     \global\let\bbl@beforestart\relax}
1167 \AtBeginDocument{%
1168     {\@nameuse{bbl@beforestart}}% Group!
1169     \if@filesw

```

```

1170 \providecommand\babel@aux[2]{}%
1171 \immediate\write\@mainaux{%
1172   \string\providecommand\string\babel@aux[2]{}%
1173   \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1174 \fi
1175 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1176 \ifbbl@single % must go after the line above.
1177   \renewcommand\selectlanguage[1]{}%
1178   \renewcommand\foreignlanguage[2]{#2}%
1179   \global\let\babel@aux\@gobbletwo % Also as flag
1180 \fi
1181 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1182 \def\select@language@x#1{%
1183   \ifcase\bbl@select@type
1184     \bbl@ifsamestring\language#1\{\}\select@language{#1}%
1185   \else
1186     \select@language{#1}%
1187   \fi}

```

7.5 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.

```

1188 \bbl@trace{Shorhands}
1189 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1190   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1191   \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1192   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1193     \begingroup
1194       \catcode`#1\active
1195       \nfss@catcodes
1196       \ifnum\catcode`#1=\active
1197         \endgroup
1198         \bbl@add\nfss@catcodes{\@makeother#1}%
1199       \else
1200         \endgroup
1201       \fi
1202   \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.

```

1203 \def\bbl@remove@special#1{%
1204   \begingroup
1205     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1206       \else\noexpand##1\noexpand##2\fi}%
1207     \def\do{\x\do}%
1208     \def\@makeother{\x\@makeother}%
1209   \edef\x{\endgroup
1210     \def\noexpand\dospecials{\dospecials}%
1211     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1212       \def\noexpand\@sanitize{\@sanitize}%
1213     \fi}%
1214   \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).

```
1215 \def\bbl@active@def#1#2#3#4{%
1216   \@namedef{#3#1}{%
1217     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1218       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1219     \else
1220       \bbl@afterfi\csname#2@sh@#1\endcsname
1221     \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.

```
1222 \long\@namedef{#3@arg#1}##1{%
1223   \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1224     \bbl@afterelse\csname#4#1\endcsname##1%
1225   \else
1226     \bbl@afterfi\csname#2@sh@#1@string##1\endcsname
1227   \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.

```
1228 \def\initiate@active@char#1{%
1229   \bbl@ifunset{active@char\string#1}%
1230   {\bbl@withactive
1231     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1232   {}}
```

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).

```
1233 \def\@initiate@active@char#1#2#3{%
1234   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1235   \ifx#1\undefined
1236     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1237   \else
1238     \bbl@csarg\let{oridef@#2}#1%
1239     \bbl@csarg\edef{oridef@#2}{%
1240       \let\noexpand#1%
1241       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1242   \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*").

```
1243   \ifx#1#3\relax
1244     \expandafter\let\csname normal@char#2\endcsname#3%
```

```

1245 \else
1246 \bbl@info{Making #2 an active character}%
1247 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1248 \namedef{normal@char#2}{%
1249 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1250 \else
1251 \namedef{normal@char#2}{#3}%
1252 \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).

```

1253 \bbl@restoreactive{#2}%
1254 \AtBeginDocument{%
1255 \catcode`#2\active
1256 \if@files
1257 \immediate\write\@mainaux{\catcode`\string#2\active}%
1258 \fi}%
1259 \expandafter\bbl@add@special\csname#2\endcsname
1260 \catcode`#2\active
1261 \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>}).

```

1262 \let\bbl@tempa\@firstoftwo
1263 \if\string^#2%
1264 \def\bbl@tempa{\noexpand\textormath}%
1265 \else
1266 \ifx\bbl@mathnormal\@undefined\else
1267 \let\bbl@tempa\bbl@mathnormal
1268 \fi
1269 \fi
1270 \expandafter\edef\csname active@char#2\endcsname{%
1271 \bbl@tempa
1272 {\noexpand\if@safe@actives
1273 \noexpand\expandafter
1274 \expandafter\noexpand\csname normal@char#2\endcsname
1275 \noexpand\else
1276 \noexpand\expandafter
1277 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1278 \noexpand\fi}%
1279 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1280 \bbl@csarg\edef{doactive#2}{%
1281 \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!).

```

1282 \bbl@csarg\edef{active@#2}{%
1283 \noexpand\active@prefix\noexpand#1%
1284 \expandafter\noexpand\csname active@char#2\endcsname}%
1285 \bbl@csarg\edef{normal@#2}{%
1286 \noexpand\active@prefix\noexpand#1%
1287 \expandafter\noexpand\csname normal@char#2\endcsname}%
1288 \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.

```
1289 \bbl@active@def#2\user@group{user@active}{language@active}%
1290 \bbl@active@def#2\language@group{language@active}{system@active}%
1291 \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 T_EX would see \protect '\protect '. To prevent this from happening a couple of shorthand needs to be defined at user level.

```
1292 \expandafter\edef\csname\user@group @sh#2@\endcsname
1293 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1294 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1295 {\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.

```
1296 \if\string'#2%
1297 \let\prim@s\bbl@prim@s
1298 \let\active@math@prime#1%
1299 \fi
1300 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1301 <<{*More package options}>> ≡
1302 \DeclareOption{math=active}{%
1303 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1304 <</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.

```
1305 \ifpackagewith{babel}{KeepShorthandsActive}%
1306 {\let\bbl@restoreactive\@gobble}%
1307 {\def\bbl@restoreactive#1{%
1308 \bbl@exp{%
1309 \\\AfterBabelLanguage\\CurrentOption
1310 {\catcode`#1=\the\catcode`#1\relax}%
1311 \\\AtEndOfPackage
1312 {\catcode`#1=\the\catcode`#1\relax}}}%
1313 \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.

```
1314 \def\bbl@sh@select#1#2{%
1315 \expandafter\ifx\csname#1sh#2@sel\endcsname\relax
1316 \bbl@afterelse\bbl@scndcs
1317 \else
1318 \bbl@afterfi\csname#1sh#2@sel\endcsname
1319 \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.

```

1320 \begingroup
1321 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1322 {\gdef\active@prefix#1{%
1323   \ifx\protect\@typeset@protect
1324   \else
1325     \ifx\protect\@unexpandable@protect
1326       \noexpand#1%
1327     \else
1328       \protect#1%
1329     \fi
1330   \expandafter\@gobble
1331   \fi}}
1332 {\gdef\active@prefix#1{%
1333   \ifincsname
1334     \string#1%
1335     \expandafter\@gobble
1336   \else
1337     \ifx\protect\@typeset@protect
1338     \else
1339       \ifx\protect\@unexpandable@protect
1340         \noexpand#1%
1341       \else
1342         \protect#1%
1343       \fi
1344       \expandafter\expandafter\expandafter\@gobble
1345     \fi
1346   \fi}}
1347 \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⟩`.

```

1348 \newif\if@safe@actives
1349 \@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.

```

1350 \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`.

```

1351 \chardef\bbl@activated\z@
1352 \def\bbl@activate#1{%
1353   \chardef\bbl@activated\@ne
1354   \bbl@withactive{\expandafter\let\expandafter}#1%
1355   \csname bbl@active@\string#1\endcsname}
1356 \def\bbl@deactivate#1{%
1357   \chardef\bbl@activated\tw@
1358   \bbl@withactive{\expandafter\let\expandafter}#1%
1359   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1360 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1361 \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.

```

1362 \def\babel@texpdf#1#2#3#4{%
1363   \ifx\texorpdfstring\undefined
1364     \textormath{#1}{#3}%
1365   \else
1366     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1367     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1368   \fi}
1369 %
1370 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1371 \def\@decl@short#1#2#3\@nil#4{%
1372   \def\bbl@tempa{#3}%
1373   \ifx\bbl@tempa\empty
1374     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1375     \bbl@ifunset{#1@sh@\string#2@}{}%
1376     {\def\bbl@tempa{#4}%
1377      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1378      \else
1379        \bbl@info
1380        {Redefining #1 shorthand \string#2\%
1381         in language \CurrentOption}%
1382      \fi}%
1383     \@namedef{#1@sh@\string#2@}{#4}%
1384   \else
1385     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1386     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1387     {\def\bbl@tempa{#4}%
1388      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1389      \else
1390        \bbl@info
1391        {Redefining #1 shorthand \string#2\string#3\%
1392         in language \CurrentOption}%
1393      \fi}%
1394     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1395   \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.

```

1396 \def\textormath{%
1397   \ifmmode
1398     \expandafter\@secondoftwo
1399   \else
1400     \expandafter\@firstoftwo
1401   \fi}

```

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

```

1402 \def\user@group{user}
1403 \def\language@group{english} % TODO. I don't like defaults
1404 \def\system@group{system}

```

`\usesshorthands` 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.

```

1405 \def\useshorthands{%
1406   \@ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1407 \def\bbl@usesesh@s#1{%
1408   \bbl@usesesh@x
1409   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1410   {#1}}
1411 \def\bbl@usesesh@x#1#2{%
1412   \bbl@ifshorthand{#2}%
1413   {\def\user@group{user}%
1414     \initiate@active@char{#2}%
1415     #1%
1416     \bbl@activate{#2}}%
1417   {\bbl@error
1418     {I can't declare a shorthand turned off (\string#2)}
1419     {Sorry, but you can't use shorthands which have been\\
1420       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.

```

1421 \def\user@language@group{user@\language@group}
1422 \def\bbl@set@user@generic#1#2{%
1423   \bbl@ifunset{user@generic@active#1}%
1424   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1425     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1426     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1427       \expandafter\noexpand\csname normal@char#1\endcsname}%
1428     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1429       \expandafter\noexpand\csname user@active#1\endcsname}}%
1430   \@empty}
1431 \newcommand\defineshorthand[3][user]{%
1432   \edef\bbl@tempa{\zap@space#1 \@empty}%
1433   \bbl@for\bbl@tempb\bbl@tempa{%
1434     \if*\expandafter\@car\bbl@tempb\@nil
1435       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1436       \@expandtwoargs
1437       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1438     \fi
1439     \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].

```

1440 \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 \aliasshorthand{"}{/} is \active@prefix /active@char/, so we still need to let the latest to \active@char".

```

1441 \def\aliasshorthand#1#2{%
1442   \bbl@ifshorthand{#2}%
1443   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1444     \ifx\document\@notprerr
1445       \@notshorthand{#2}%
1446     \else
1447       \initiate@active@char{#2}%
1448       \expandafter\let\csname active@char\string#2\expandafter\endcsname
1449       \csname active@char\string#1\endcsname
1450       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1451       \csname normal@char\string#1\endcsname
1452       \bbl@activate{#2}%
1453     \fi

```

```

1454 \fi}%
1455 {\bbl@error
1456 {Cannot declare a shorthand turned off (\string#2)}
1457 {Sorry, but you cannot use shorthands which have been\\%
1458 turned off in the package options}}}

```

\@notshorthand

```

1459 \def\@notshorthand#1{%
1460 \bbl@error{%
1461 The character '\string #1' should be made a shorthand character;\\%
1462 add the command \string\usesshorthands\string{#1\string} to
1463 the preamble.\\%
1464 I will ignore your instruction}%
1465 {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.

```

1466 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1467 \DeclareRobustCommand*\shorthandoff{%
1468 \ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1469 \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.

```

1470 \def\bbl@switch@sh#1#2{%
1471 \ifx#2\@nnil\else
1472 \bbl@ifunset{\bbl@active@\string#2}%
1473 {\bbl@error
1474 {I can't switch '\string#2' on or off--not a shorthand}%
1475 {This character is not a shorthand. Maybe you made\\%
1476 a typing mistake? I will ignore your instruction.}}%
1477 {\ifcase#1% off, on, off*
1478 \catcode`#2\relax
1479 \or
1480 \catcode`#2\active
1481 \bbl@ifunset{\bbl@shdef@\string#2}%
1482 {}%
1483 {\bbl@withactive{\expandafter\let\expandafter}#2%
1484 \csname bbl@shdef@\string#2\endcsname
1485 \bbl@csarg\let{shdef@\string#2}\relax}%
1486 \ifcase\bbl@activated\or
1487 \bbl@activate{#2}%
1488 \else
1489 \bbl@deactivate{#2}%
1490 \fi
1491 \or
1492 \bbl@ifunset{\bbl@shdef@\string#2}%
1493 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1494 {}%
1495 \csname bbl@oricat@\string#2\endcsname
1496 \csname bbl@oridef@\string#2\endcsname
1497 \fi}%
1498 \bbl@afterfi\bbl@switch@sh#1%
1499 \fi}

```

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

```

1500 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1501 \def\bbl@putsh#1{%

```

```

1502 \bbl@ifunset{bbl@active@\string#1}%
1503 {\bbl@putsh@i#1\@empty\@nnil}%
1504 {\csname bbl@active@\string#1\endcsname}}
1505 \def\bbl@putsh@i#1#2\@nnil{%
1506 \csname\language@group @sh@\string#1@%
1507 \ifx\@empty#2\else\string#2@\fi\endcsname}
1508 \ifx\bbl@opt@shorthands\@nnil\else
1509 \let\bbl@s@initiate@active@char\initiate@active@char
1510 \def\initiate@active@char#1{%
1511 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1512 \let\bbl@s@switch@sh\bbl@switch@sh
1513 \def\bbl@switch@sh#1#2{%
1514 \ifx#2\@nnil\else
1515 \bbl@afterfi
1516 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1517 \fi}
1518 \let\bbl@s@activate\bbl@activate
1519 \def\bbl@activate#1{%
1520 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1521 \let\bbl@s@deactivate\bbl@deactivate
1522 \def\bbl@deactivate#1{%
1523 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1524 \fi

```

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

```

1525 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

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

```

1526 \def\bbl@prim@s{%
1527 \prime\futurelet\@let@token\bbl@pr@m@s}
1528 \def\bbl@if@primes#1#2{%
1529 \ifx#1\@let@token
1530 \expandafter\@firstoftwo
1531 \else\ifx#2\@let@token
1532 \bbl@afterelse\expandafter\@firstoftwo
1533 \else
1534 \bbl@afterfi\expandafter\@secondoftwo
1535 \fi\fi}
1536 \begingroup
1537 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1538 \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1539 \lowercase{%
1540 \gdef\bbl@pr@m@s{%
1541 \bbl@if@primes"%
1542 \pr@@@s
1543 {\bbl@if@primes*\^{\pr@@@t\egroup}}}}
1544 \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).

```

1545 \initiate@active@char{~}
1546 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1547 \bbl@activate{~}

```


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

```
1548 \expandafter\def\csname OT1dqpos\endcsname{127}
1549 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1550 \ifx\f@encoding\undefined
1551   \def\f@encoding{OT1}
1552 \fi
```

7.6 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.

```
1553 \bbl@trace{Language attributes}
1554 \newcommand\languageattribute[2]{%
1555   \def\bbl@tempc{#1}%
1556   \bbl@fixname\bbl@tempc
1557   \bbl@iflanguage\bbl@tempc{%
1558     \bbl@vforeach{#2}{%
```

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

```
1559     \ifx\bbl@known@attrs\undefined
1560       \in@false
1561     \else
1562       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1563     \fi
1564     \ifin@
1565       \bbl@warning{%
1566         You have more than once selected the attribute '##1'\%
1567         for language #1. Reported}%
1568     \else
```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX -code.

```
1569       \bbl@exp{%
1570         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1571       \edef\bbl@tempa{\bbl@tempc-##1}%
1572       \expandafter\bbl@ifknown@ttr\bbl@tempa\bbl@attributes%
1573       {\csname\bbl@tempc @attr##1\endcsname}%
1574       {\@attrerr{\bbl@tempc}{##1}}%
1575     \fi}}
1576 \@onlypreamble\languageattribute
```

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

```
1577 \newcommand*{\@attrerr}[2]{%
1578   \bbl@error
1579   {The attribute #2 is unknown for language #1.}%
1580   {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}`.

```
1581 \def\bbl@declare@ttribute#1#2#3{%
1582   \bbl@xin@{,#2,}{,\BabelModifiers,}%
```

```

1583 \ifin@
1584 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1585 \fi
1586 \bbl@add@list\bbl@attributes{#1-#2}%
1587 \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

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

```

1588 \def\bbl@ifattributeset#1#2#3#4{%
1589 \ifx\bbl@known@attribs\undefined
1590 \in@false
1591 \else
1592 \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1593 \fi
1594 \ifin@
1595 \bbl@afterelse#3%
1596 \else
1597 \bbl@afterfi#4%
1598 \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.

```

1599 \def\bbl@ifknown@ttrib#1#2{%
1600 \let\bbl@tempa\@secondoftwo
1601 \bbl@loopx\bbl@tempb{#2}{%
1602 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1603 \ifin@
1604 \let\bbl@tempa\@firstoftwo
1605 \else
1606 \fi}%
1607 \bbl@tempa}

```

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

```

1608 \def\bbl@clear@ttribs{%
1609 \ifx\bbl@attributes\undefined\else
1610 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1611 \expandafter\bbl@clear@ttrib\bbl@tempa.
1612 }%
1613 \let\bbl@attributes\undefined
1614 \fi}
1615 \def\bbl@clear@ttrib#1-#2.{%
1616 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1617 \AtBeginDocument{\bbl@clear@ttribs}

```

7.7 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`

```

1618 \bbl@trace{Macros for saving definitions}
1619 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1620 \newcount\babel@savecnt
1621 \babel@beginsave

```

`\babel@save` The macro `\babel@save⟨csmame⟩` saves the current meaning of the control sequence `⟨csmame⟩` to `\originalTeX`³². To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive.

```

1622 \def\babel@save#1{%
1623   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1624   \toks@\expandafter{\originalTeX\let#1=}%
1625   \bbl@exp{%
1626     \def\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1627   \advance\babel@savecnt\@ne}
1628 \def\babel@savevariable#1{%
1629   \toks@\expandafter{\originalTeX #1=}%
1630   \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 on when it isn't already in effect and `\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.

```

1631 \def\bbl@frenchspacing{%
1632   \ifnum\the\scode`\.=\@m
1633     \let\bbl@nonfrenchspacing\relax
1634   \else
1635     \frenchspacing
1636     \let\bbl@nonfrenchspacing\nonfrenchspacing
1637   \fi}
1638 \let\bbl@nonfrenchspacing\nonfrenchspacing
1639 \let\bbl@elt\relax
1640 \edef\bbl@fs@chars{%
1641   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1642   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1643   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1644 \def\bbl@pre@fs{%
1645   \def\bbl@elt##1##2##3{\scode`##1=\the\scode`##1\relax}%
1646   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1647 \def\bbl@post@fs{%
1648   \bbl@save@sfcodes
1649   \edef\bbl@tempa{\bbl@cl{frspc}}%
1650   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1651   \if u\bbl@tempa      % do nothing
1652   \else\if n\bbl@tempa % non french
1653     \def\bbl@elt##1##2##3{%
1654       \ifnum\scode`##1=##2\relax
1655         \babel@savevariable{\scode`##1}%
1656       \scode`##1=##3\relax
1657     \fi}%
1658     \bbl@fs@chars
1659   \else\if y\bbl@tempa % french
1660     \def\bbl@elt##1##2##3{%
1661       \ifnum\scode`##1=##3\relax
1662         \babel@savevariable{\scode`##1}%
1663       \scode`##1=##2\relax
1664     \fi}%

```

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

```

1665 \bbl@fs@chars
1666 \fi\fi\fi}

```

7.8 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.

```

1667 \bbl@trace{Short tags}
1668 \def\babeltags#1{%
1669   \edef\bbl@tempa{\zap@space#1 \@empty}%
1670   \def\bbl@tempb##1=##2\@{%
1671     \edef\bbl@tempc{%
1672       \noexpand\noexpand\csname ##1\endcsname{%
1673         \noexpand\protect
1674         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1675       \noexpand\noexpand\csname text##1\endcsname{%
1676         \noexpand\foreignlanguage{##2}}}
1677     \bbl@tempc}%
1678   \bbl@for\bbl@tempa\bbl@tempa{%
1679     \expandafter\bbl@tempb\bbl@tempa\@}%

```

7.9 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.

```

1682 \bbl@trace{Hyphens}
1683 \@onlypreamble\babelhyphenation
1684 \AtEndOfPackage{%
1685   \newcommand\babelhyphenation[2][\@empty]{%
1686     \ifx\bbl@hyphenation@relax
1687       \let\bbl@hyphenation@\@empty
1688     \fi
1689     \ifx\bbl@hyphlist\@empty\else
1690       \bbl@warning{%
1691         You must not intermingle \string\selectlanguage\space and\%
1692         \string\babelhyphenation\space or some exceptions will not\%
1693         be taken into account. Reported}%
1694     \fi
1695     \ifx\@empty#1%
1696       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1697     \else
1698       \bbl@vforeach{#1}{%
1699         \def\bbl@tempa{##1}%
1700         \bbl@fixname\bbl@tempa
1701         \bbl@iflanguage\bbl@tempa{%
1702           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1703             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1704             {}%
1705             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1706             #2}}}%
1707       \fi}}

```

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

```

1708 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1709 \def\bbl@t@one{T1}
1710 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

³³TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

`\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`.

```

1711 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1712 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1713 \def\bbl@hyphen{%
1714   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1715 \def\bbl@hyphen@i#1#2{%
1716   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
1717     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1718     {\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.

```

1719 \def\bbl@usehyphen#1{%
1720   \leavevmode
1721   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1722   \nobreak\hskip\z@skip}
1723 \def\bbl@@usehyphen#1{%
1724   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1725 \def\bbl@hyphenchar{%
1726   \ifnum\hyphenchar\font=\m@ne
1727     \babelnullhyphen
1728   \else
1729     \char\hyphenchar\font
1730   \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.

```

1731 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1732 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1733 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1734 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1735 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1736 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1737 \def\bbl@hy@repeat{%
1738   \bbl@usehyphen{%
1739     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1740 \def\bbl@hy@@repeat{%
1741   \bbl@usehyphen{%
1742     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1743 \def\bbl@hy@empty{\hskip\z@skip}
1744 \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.

```

1745 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

```

7.10 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.

```

1746 \bbl@trace{Multiencoding strings}
1747 \def\bbl@toglobal#1{\global\let#1#1}
1748 \def\bbl@recatcode#1{% TODO. Used only once?
1749   \@tempcnta="7F
1750   \def\bbl@tempa{%
1751     \ifnum\@tempcnta>"FF\else
1752       \catcode\@tempcnta=#1\relax
1753       \advance\@tempcnta\@ne
1754       \expandafter\bbl@tempa
1755     \fi}%
1756   \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).

```

1757 \ifpackagewith{babel}{nocase}%
1758   {\let\bbl@patchuclc\relax}%
1759   {\def\bbl@patchuclc{%
1760     \global\let\bbl@patchuclc\relax
1761     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1762     \gdef\bbl@uclc##1{%
1763       \let\bbl@encoded\bbl@encoded@uclc
1764       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1765       {##1}%
1766       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1767         \csname\language @bbl@uclc\endcsname}%
1768       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1769   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1770   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1771 % A temporary hack:
1772 \ifx\BabelCaseHack\@undefined
1773 \AtBeginDocument{%
1774   \bbl@exp{%
1775     \\\in@\string\@uclclist}%
1776     {\expandafter\meaning\csname MakeUppercase \endcsname}}%
1777   \ifin@ \else
1778     \expandafter\let\expandafter\bbl@newuc\csname MakeUppercase \endcsname
1779     \protected\@namedef{MakeUppercase }#1{%
1780       \def\reserved@a##1##2{\let##1##2\reserved@a}%
1781       \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b@gobble}%
1782       \protected@edef\reserved@a{\bbl@newuc{#1}}\reserved@a}%
1783     \expandafter\let\expandafter\bbl@newlc\csname MakeLowercase \endcsname
1784     \protected\@namedef{MakeLowercase }#1{%
1785       \def\reserved@a##1##2{\let##2##1\reserved@a}%
1786       \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b@gobble}%
1787       \protected@edef\reserved@a{\bbl@newlc{#1}}\reserved@a}%
1788     \fi}
1789 \fi

1790 <<(*More package options)>> ≡
1791 \DeclareOption{nocase}{}
1792 <</More package options>>

```

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

```

1793 <<{*More package options}>> ≡
1794 \let\bbl@opt@strings\@nnil % accept strings=value
1795 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1796 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1797 \def\BabelStringsDefault{generic}
1798 <</More package options>>

```

Main command This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

```

1799 \@onlypreamble\StartBabelCommands
1800 \def\StartBabelCommands{%
1801   \begingroup
1802   \bbl@reccatcode{11}%
1803   <<Macros local to BabelCommands>>
1804   \def\bbl@provstring##1##2{%
1805     \providecommand##1{##2}%
1806     \bbl@tglobal##1}%
1807   \global\let\bbl@scafter\@empty
1808   \let\StartBabelCommands\bbl@startcmds
1809   \ifx\BabelLanguages\relax
1810     \let\BabelLanguages\CurrentOption
1811   \fi
1812   \begingroup
1813   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1814   \StartBabelCommands}
1815 \def\bbl@startcmds{%
1816   \ifx\bbl@screset\@nnil\else
1817     \bbl@usehooks{stopcommands}{}%
1818   \fi
1819   \endgroup
1820   \begingroup
1821   \@ifstar
1822     {\ifx\bbl@opt@strings\@nnil
1823       \let\bbl@opt@strings\BabelStringsDefault
1824     \fi
1825     \bbl@startcmds@i}%
1826   \bbl@startcmds@i}
1827 \def\bbl@startcmds@i#1#2{%
1828   \edef\bbl@L{\zap@space#1 \@empty}%
1829   \edef\bbl@G{\zap@space#2 \@empty}%
1830   \bbl@startcmds@ii}
1831 \let\bbl@startcommands\StartBabelCommands

```

Parse the encoding info to get the label, input, and font parts.

Select the behavior of `\SetString`. There are two main cases, depending of if there is an optional argument: without it and `strings=encoded`, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and `strings=encoded`, define the strings, but with another value, define strings only if the current label or font encoding is the value of `strings`; otherwise (ie, no strings or a block whose label is not in `strings=`) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1832 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1833   \let\SetString@gobbletwo
1834   \let\bbl@stringdef@gobbletwo
1835   \let\AfterBabelCommands@gobble
1836   \ifx\@empty#1%
1837     \def\bbl@sc@label{generic}%
1838     \def\bbl@encstring##1##2{%
1839       \ProvideTextCommandDefault##1{##2}%

```

```

1840 \bbl@toglobal##1%
1841 \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1842 \let\bbl@sctest\in@true
1843 \else
1844 \let\bbl@sc@charset\space % <- zapped below
1845 \let\bbl@sc@fontenc\space % <- " "
1846 \def\bbl@tempa##1=##2\@nil{%
1847 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1848 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1849 \def\bbl@tempa##1 ##2{% space -> comma
1850 ##1%
1851 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1852 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1853 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1854 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1855 \def\bbl@encstring##1##2{%
1856 \bbl@foreach\bbl@sc@fontenc{%
1857 \bbl@ifunset{T@####1}%
1858 }%
1859 {\ProvideTextCommand##1{####1}{##2}%
1860 \bbl@toglobal##1%
1861 \expandafter
1862 \bbl@toglobal\csname####1\string##1\endcsname}}}%
1863 \def\bbl@sctest{%
1864 \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
1865 \fi
1866 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1867 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1868 \let\AfterBabelCommands\bbl@aftercmds
1869 \let\SetString\bbl@setstring
1870 \let\bbl@stringdef\bbl@encstring
1871 \else % ie, strings=value
1872 \bbl@sctest
1873 \ifin@
1874 \let\AfterBabelCommands\bbl@aftercmds
1875 \let\SetString\bbl@setstring
1876 \let\bbl@stringdef\bbl@provstring
1877 \fi\fi\fi
1878 \bbl@scswitch
1879 \ifx\bbl@G\@empty
1880 \def\SetString##1##2{%
1881 \bbl@error{Missing group for string \string##1}%
1882 {You must assign strings to some category, typically\\%
1883 captions or extras, but you set none}}%
1884 \fi
1885 \ifx\@empty#1%
1886 \bbl@usehooks{defaultcommands}{}%
1887 \else
1888 \@expandtwoargs
1889 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1890 \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 versions 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).

```

1891 \def\bbl@forlang#1#2{%
1892 \bbl@for#1\bbl@L{%
1893 \bbl@xin@{, #1, }{\BabelLanguages,}%

```



```

1894 \ifin@#2\relax\fi}}
1895 \def\bbl@scswitch{%
1896 \bbl@forlang\bbl@tempa{%
1897 \ifx\bbl@G\@empty\else
1898 \ifx\SetString@gobbletwo\else
1899 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1900 \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1901 \ifin@else
1902 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1903 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1904 \fi
1905 \fi
1906 \fi}}
1907 \AtEndOfPackage{%
1908 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1909 \let\bbl@scswitch\relax}
1910 \@onlypreamble\EndBabelCommands
1911 \def\EndBabelCommands{%
1912 \bbl@usehooks{stopcommands}{}%
1913 \endgroup
1914 \endgroup
1915 \bbl@scafter}
1916 \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.

```

1917 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1918 \bbl@forlang\bbl@tempa{%
1919 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1920 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1921 {\bbl@exp{%
1922 \global\bbbl@add<\bbl@G\bbl@tempa>{\bbbl@scset\1<\bbl@LC>}}}%
1923 }%
1924 \def\BabelString{#2}%
1925 \bbl@usehooks{stringprocess}{}%
1926 \expandafter\bbl@stringdef
1927 \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.

```

1928 \ifx\bbl@opt@strings\relax
1929 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1930 \bbl@patchuclc
1931 \let\bbl@encoded\relax
1932 \def\bbl@encoded@uclc#1{%
1933 \@inmathwarn#1%
1934 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1935 \expandafter\ifx\csname ?\string#1\endcsname\relax
1936 \TextSymbolUnavailable#1%
1937 \else
1938 \csname ?\string#1\endcsname
1939 \fi
1940 \else
1941 \csname\cf@encoding\string#1\endcsname
1942 \fi}
1943 \else
1944 \def\bbl@scset#1#2{\def#1{#2}}
1945 \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.

```

1946 <<*Macros local to BabelCommands>> ≡
1947 \def\SetStringLoop##1##2{%
1948   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1949   \count@ \z@
1950   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1951     \advance\count@\@ne
1952     \toks@\expandafter{\bbl@tempa}%
1953     \bbl@exp{%
1954       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1955       \count@=\the\count@\relax}}}%
1956 <</Macros local to BabelCommands>>

```

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

```

1957 \def\bbl@aftercmds#1{%
1958   \toks@\expandafter{\bbl@scafter#1}%
1959   \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.

```

1960 <<*Macros local to BabelCommands>> ≡
1961 \newcommand\SetCase[3][]{%
1962   \bbl@patchuclc
1963   \bbl@forlang\bbl@tempa{%
1964     \expandafter\bbl@encstring
1965     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1966     \expandafter\bbl@encstring
1967     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1968     \expandafter\bbl@encstring
1969     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1970 <</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.

```

1971 <<*Macros local to BabelCommands>> ≡
1972 \newcommand\SetHyphenMap[1]{%
1973   \bbl@forlang\bbl@tempa{%
1974     \expandafter\bbl@stringdef
1975     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1976 <</Macros local to BabelCommands>>

```

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

```

1977 \newcommand\BabelLower[2]{% one to one.
1978   \ifnum\lccode#1=#2\else
1979     \babel@savevariable{\lccode#1}%
1980     \lccode#1=#2\relax
1981   \fi}
1982 \newcommand\BabelLowerMM[4]{% many-to-many
1983   \@tempcnta=#1\relax
1984   \@tempcntb=#4\relax
1985   \def\bbl@tempa{%
1986     \ifnum\@tempcnta>#2\else
1987       \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1988       \advance\@tempcnta#3\relax
1989       \advance\@tempcntb#3\relax
1990       \expandafter\bbl@tempa
1991     \fi}%
1992   \bbl@tempa}

```

```

1993 \newcommand\BabelLowerMO[4]{% many-to-one
1994   \@tempcnta=#1\relax
1995   \def\bbl@tempa{%
1996     \ifnum\@tempcnta>#2\else
1997       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1998       \advance\@tempcnta#3
1999       \expandafter\bbl@tempa
2000     \fi}%
2001   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2002 <{*More package options}> ≡
2003 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2004 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2005 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2006 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2007 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2008 <{/More package options}>

```

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

```

2009 \AtEndOfPackage{%
2010   \ifx\bbl@opt@hyphenmap\undefined
2011     \bbl@xin@{,}{\bbl@language@opts}%
2012     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2013   \fi}

```

This section 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.

```

2014 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2015   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2016 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2017   \bbl@trim@def\bbl@tempa{#2}%
2018   \bbl@xin@{.template}{\bbl@tempa}%
2019   \ifin@
2020     \bbl@ini@captions@template{#3}{#1}%
2021   \else
2022     \edef\bbl@tempd{%
2023       \expandafter\expandafter\expandafter
2024       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2025     \bbl@xin@
2026       {\expandafter\string\csname #2name\endcsname}%
2027     {\bbl@tempd}%
2028     \ifin@ % Renew caption
2029       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2030     \ifin@
2031       \bbl@exp{%
2032         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2033         {\bbl@scset\<#2name>\<#1#2name>}%
2034         {}}%
2035       \else % Old way converts to new way
2036         \bbl@ifunset{#1#2name}%
2037         {\bbl@exp{%
2038           \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2039           \\bbl@ifsamestring{\bbl@tempa}{\language}%
2040           {\def\<#2name>{\<#1#2name>}}%
2041           {}}}%
2042         {}%
2043       \fi
2044     \else
2045       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2046       \ifin@ % New way
2047         \bbl@exp{%
2048           \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}%

```

```

2049      \\bbl@ifsamestring{\bbl@tempa}{\language}%
2050      {\bbl@scset\<#2name>\<#1#2name>}%
2051      {}}%
2052      \else % Old way, but defined in the new way
2053      \bbl@exp{%
2054      \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2055      \\bbl@ifsamestring{\bbl@tempa}{\language}%
2056      {\def\<#2name>{\<#1#2name>}}%
2057      {}}%
2058      \fi%
2059      \fi
2060      \@namedef{#1#2name}{#3}%
2061      \toks@\expandafter{\bbl@captionslist}%
2062      \bbl@exp{\\in@{\<#2name>}{\the\toks@}}%
2063      \ifin\else
2064      \bbl@exp{\\bbl@add\\bbl@captionslist{\<#2name>}}%
2065      \bbl@tglobal\bbl@captionslist
2066      \fi
2067      \fi}
2068 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')

```

7.11 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.

```

2069 \bbl@trace{Macros related to glyphs}
2070 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2071   \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2072   \setbox\z@ \hbox{\lower\dimen\z@ \box\z@}\ht\z@ \ht\tw@ \dp\z@ \dp\tw@}

```

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

```

2073 \def\save@sf@q#1{\leavevmode
2074   \begingroup
2075   \edef\SF{\spacefactor\the\spacefactor}#1\SF
2076   \endgroup}

```

7.12 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`.

7.12.1 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.

```

2077 \ProvideTextCommand{\quotedblbase}{OT1}{%
2078   \save@sf@q{\set@low@box{\textquotedblright\}}%
2079   \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.

```

2080 \ProvideTextCommandDefault{\quotedblbase}{%
2081   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2082 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2083   \save@sf@q{\set@low@box{\textquoteright\}}%
2084   \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.

```

2085 \ProvideTextCommandDefault{\quotesinglbase}{%
2086   \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2087 \ProvideTextCommand{\guillemetleft}{OT1}{%
2088   \ifmmode
2089     \ll
2090   \else
2091     \save@sf@q{\nobreak
2092       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2093   \fi}
2094 \ProvideTextCommand{\guillemetright}{OT1}{%
2095   \ifmmode
2096     \gg
2097   \else
2098     \save@sf@q{\nobreak
2099       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2100   \fi}
2101 \ProvideTextCommand{\guillemotleft}{OT1}{%
2102   \ifmmode
2103     \ll
2104   \else
2105     \save@sf@q{\nobreak
2106       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2107   \fi}
2108 \ProvideTextCommand{\guillemotright}{OT1}{%
2109   \ifmmode
2110     \gg
2111   \else
2112     \save@sf@q{\nobreak
2113       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2114   \fi}

```

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

```

2115 \ProvideTextCommandDefault{\guillemetleft}{%
2116   \UseTextSymbol{OT1}{\guillemetleft}}
2117 \ProvideTextCommandDefault{\guillemetright}{%
2118   \UseTextSymbol{OT1}{\guillemetright}}
2119 \ProvideTextCommandDefault{\guillemotleft}{%
2120   \UseTextSymbol{OT1}{\guillemotleft}}
2121 \ProvideTextCommandDefault{\guillemotright}{%
2122   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2123 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2124   \ifmmode
2125     <%
2126   \else
2127     \save@sf@q{\nobreak
2128       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2129   \fi}
2130 \ProvideTextCommand{\guilsinglright}{OT1}{%
2131   \ifmmode
2132     >%
2133   \else
2134     \save@sf@q{\nobreak
2135       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2136   \fi}

```

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

```

2137 \ProvideTextCommandDefault{\guilsinglleft}{%
2138   \UseTextSymbol{OT1}{\guilsinglleft}}
2139 \ProvideTextCommandDefault{\guilsinglright}{%
2140   \UseTextSymbol{OT1}{\guilsinglright}}

```

7.12.2 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.

```
2141 \DeclareTextCommand{\ij}{OT1}{%
2142   i\kern-0.02em\bb1@allowhyphens j}
2143 \DeclareTextCommand{\IJ}{OT1}{%
2144   I\kern-0.02em\bb1@allowhyphens J}
2145 \DeclareTextCommand{\ij}{T1}{\char188}
2146 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2147 \ProvideTextCommandDefault{\ij}{%
2148   \UseTextSymbol{OT1}{\ij}}
2149 \ProvideTextCommandDefault{\IJ}{%
2150   \UseTextSymbol{OT1}{\IJ}}
```

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

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

```
2151 \def\crttic@{\hrule height0.1ex width0.3em}
2152 \def\crttic@{\hrule height0.1ex width0.33em}
2153 \def\ddj@{%
2154   \setbox0\hbox{d}\dimen@=\ht0
2155   \advance\dimen@1ex
2156   \dimen@.45\dimen@
2157   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2158   \advance\dimen@ii.5ex
2159   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2160 \def\DDJ@{%
2161   \setbox0\hbox{D}\dimen@=.55\ht0
2162   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2163   \advance\dimen@ii.15ex % correction for the dash position
2164   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2165   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2166   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2167 %
2168 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2169 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2170 \ProvideTextCommandDefault{\dj}{%
2171   \UseTextSymbol{OT1}{\dj}}
2172 \ProvideTextCommandDefault{\DJ}{%
2173   \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.

```
2174 \DeclareTextCommand{\SS}{OT1}{SS}
2175 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

7.12.3 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 2176 \ProvideTextCommandDefault{\glq}{%
2177   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2178 \ProvideTextCommand{\grq}{T1}{%
2179   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2180 \ProvideTextCommand{\grq}{TU}{%
2181   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2182 \ProvideTextCommand{\grq}{OT1}{%
2183   \save@sf@q{\kern-.0125em
2184     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2185     \kern.07em\relax}}
2186 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2187 \ProvideTextCommandDefault{\glqq}{%
2188   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2189 \ProvideTextCommand{\grqq}{T1}{%
2190   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2191 \ProvideTextCommand{\grqq}{TU}{%
2192   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2193 \ProvideTextCommand{\grqq}{OT1}{%
2194   \save@sf@q{\kern-.07em
2195     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2196     \kern.07em\relax}}
2197 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2198 \ProvideTextCommandDefault{\flq}{%
2199   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2200 \ProvideTextCommandDefault{\frq}{%
2201   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2202 \ProvideTextCommandDefault{\flqq}{%
2203   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2204 \ProvideTextCommandDefault{\frqq}{%
2205   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

7.12.4 Umlauts and tremas

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

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

```
2206 \def\umlauthigh{%
2207   \def\bbl@umlauta##1{\leavevmode\bgroup%
2208     \expandafter\accent\csname\f@encoding dqpos\endcsname
2209     ##1\bbl@allowhyphens\egroup}%
2210   \let\bbl@umlaute\bbl@umlauta}
2211 \def\umlautlow{%
2212   \def\bbl@umlauta{\protect\lower@umlaut}}
2213 \def\umlautelow{%
2214   \def\bbl@umlaute{\protect\lower@umlaut}}
2215 \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.

```
2216 \expandafter\ifx\csname U@D\endcsname\relax
2217   \csname newdimen\endcsname\U@D
2218 \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.

```

2219 \def\lower@umlaut#1{%
2220   \leavevmode\bgroup
2221   \U@D 1ex%
2222   {\setbox\z@\hbox{%
2223     \expandafter\char\csname\fontencoding dqpos\endcsname}%
2224     \dimen@ -.45ex\advance\dimen@\ht\z@
2225     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2226   \expandafter\accent\csname\fontencoding dqpos\endcsname
2227   \fontdimen5\font\U@D #1%
2228   \egroup}

```

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

```

2229 \AtBeginDocument{%
2230   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlaut{a}}%
2231   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaut{e}}%
2232   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaut{i}}%
2233   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaut{\i}}%
2234   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlaut{o}}%
2235   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlaut{u}}%
2236   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlaut{A}}%
2237   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaut{E}}%
2238   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaut{I}}%
2239   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlaut{O}}%
2240   \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.

```

2241 \ifx\l@english\undefined
2242   \chardef\l@english\z@
2243 \fi
2244 % The following is used to cancel rules in ini files (see Amharic).
2245 \ifx\l@unhyphenated\undefined
2246   \newlanguage\l@unhyphenated
2247 \fi

```

7.13 Layout

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

```

2248 \bbl@trace{Bidi layout}
2249 \providecommand\IfBabelLayout[3]{#3}%
2250 \newcommand\BabelPatchSection[1]{%
2251   \@ifundefined{#1}{}{%
2252     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2253     \@namedef{#1}{%
2254       \ifstar{\bbl@presec{s}{#1}}%
2255       {\@dblarg{\bbl@presec{x}{#1}}}}}%
2256 \def\bbl@presec@x#1[#2]#3{%
2257   \bbl@exp{%
2258     \select@language{x{\bbl@main@language}%
2259     \bbl@cs{sspre@#1}%

```



```

2260   \bbl@cs{ss@#1}%
2261   [\foreignlanguage{\language}\unexpanded{#2}]%
2262   {\foreignlanguage{\language}\unexpanded{#3}}}%
2263   \select@language@x{\language}}%
2264 \def\bbl@presec@#1#2{%
2265   \bbl@exp{%
2266     \select@language@x{\bbl@main@language}%
2267     \bbl@cs{sspre@#1}%
2268     \bbl@cs{ss@#1}*%
2269     {\foreignlanguage{\language}\unexpanded{#2}}}%
2270     \select@language@x{\language}}}%
2271 \IfBabelLayout{sectioning}%
2272   {\BabelPatchSection{part}%
2273    \BabelPatchSection{chapter}%
2274    \BabelPatchSection{section}%
2275    \BabelPatchSection{subsection}%
2276    \BabelPatchSection{subsubsection}%
2277    \BabelPatchSection{paragraph}%
2278    \BabelPatchSection{subparagraph}}%
2279   \def\babel@toc#1{%
2280     \select@language@x{\bbl@main@language}}}%
2281 \IfBabelLayout{captions}%
2282   {\BabelPatchSection{caption}}}%

```

7.14 Load engine specific macros

Some macros are not defined in all engines, so, after loading the files define them if necessary to raise an error.

```

2283 \bbl@trace{Input engine specific macros}
2284 \ifcase\bbl@engine
2285   \input txtbabel.def
2286 \or
2287   \input luababel.def
2288 \or
2289   \input xebabel.def
2290 \fi
2291 \providecommand\babelfont{%
2292   \bbl@error
2293   {This macro is available only in LuaLaTeX and XeLaTeX.}%
2294   {Consider switching to these engines.}}
2295 \providecommand\babelprehyphenation{%
2296   \bbl@error
2297   {This macro is available only in LuaLaTeX.}%
2298   {Consider switching to that engine.}}
2299 \ifx\babelposthyphenation\undefined
2300   \let\babelposthyphenation\babelprehyphenation
2301   \let\babelpatterns\babelprehyphenation
2302   \let\babelcharproperty\babelprehyphenation
2303 \fi

```

7.15 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.

```

2304 \bbl@trace{Creating languages and reading ini files}
2305 \let\bbl@extend@ini@gobble
2306 \newcommand\babelprovide[2][]{%
2307   \let\bbl@savelangname\language
2308   \edef\bbl@savelocaleid{\the\localeid}%
2309   % Set name and locale id
2310   \edef\language{#2}%

```

```

2311 \bbl@id@assign
2312 % Initialize keys
2313 \bbl@foreach{captions,date,import,main,script,language,%
2314     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2315     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2316     Alph,labels,labels*,calendar,date}%
2317 {\bbl@csarg\let{KVP@##1}\@nnil}%
2318 \global\let\bbl@release@transforms\@empty
2319 \let\bbl@calendars\@empty
2320 \global\let\bbl@inidata\@empty
2321 \global\let\bbl@extend@ini\@gobble
2322 \gdef\bbl@key@list{;}%
2323 \bbl@forkv{#1}{%
2324     \in@{/}{##1}%
2325     \ifin@
2326         \global\let\bbl@extend@ini\bbl@extend@ini@aux
2327         \bbl@renewinikey##1\@{##2}%
2328     \else
2329         \bbl@csarg\ifx{KVP@##1}\@nnil\else
2330             \bbl@error
2331             {Unknown key '##1' in \string\babelprovide}%
2332             {See the manual for valid keys}%
2333         \fi
2334         \bbl@csarg\def{KVP@##1}{##2}%
2335     \fi}%
2336 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2337 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2338 % == init ==
2339 \ifx\bbl@screset\@undefined
2340     \bbl@ldfinit
2341 \fi
2342 % == date (as option) ==
2343 % \ifx\bbl@KVP@date\@nnil\else
2344 % \fi
2345 % ==
2346 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2347 \ifcase\bbl@howloaded
2348     \let\bbl@lbkflag\@empty % new
2349 \else
2350     \ifx\bbl@KVP@hyphenrules\@nnil\else
2351         \let\bbl@lbkflag\@empty
2352     \fi
2353     \ifx\bbl@KVP@import\@nnil\else
2354         \let\bbl@lbkflag\@empty
2355     \fi
2356 \fi
2357 % == import, captions ==
2358 \ifx\bbl@KVP@import\@nnil\else
2359     \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
2360     {\ifx\bbl@initload\relax
2361         \begingroup
2362             \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2363             \bbl@input@texini{##2}%
2364         \endgroup
2365     \else
2366         \xdef\bbl@KVP@import{\bbl@initload}%
2367     \fi}%
2368 \}%
2369 \let\bbl@KVP@date\@empty
2370 \fi
2371 \ifx\bbl@KVP@captions\@nnil
2372     \let\bbl@KVP@captions\bbl@KVP@import
2373 \fi

```

```

2374 % ==
2375 \ifx\bbbl@KVP@transforms\@nnil\else
2376   \bbbl@replace\bbbl@KVP@transforms{ }{,}%
2377 \fi
2378 % == Load ini ==
2379 \ifcase\bbbl@howloaded
2380   \bbbl@provide@new{#2}%
2381 \else
2382   \bbbl@ifblank{#1}%
2383   {}% With \bbbl@load@basic below
2384   {\bbbl@provide@renew{#2}}%
2385 \fi
2386 % Post tasks
2387 % -----
2388 % == subsequent calls after the first provide for a locale ==
2389 \ifx\bbbl@inidata\@empty\else
2390   \bbbl@extend@ini{#2}%
2391 \fi
2392 % == ensure captions ==
2393 \ifx\bbbl@KVP@captions\@nnil\else
2394   \bbbl@ifunset{\bbbl@extracaps@#2}%
2395   {\bbbl@exp{\babelensure[exclude=\today]{#2}}}%
2396   {\bbbl@exp{\babelensure[exclude=\today,
2397     include=\bbbl@extracaps@#2]}{#2}}%
2398   \bbbl@ifunset{\bbbl@ensure@language}%
2399   {\bbbl@exp{%
2400     \DeclareRobustCommand\<\bbbl@ensure@language>[1]{%
2401       \foreignlanguage{language}%
2402       {###1}}}%
2403   }%
2404   \bbbl@exp{%
2405     \bbbl@tglobal\<\bbbl@ensure@language>%
2406     \bbbl@tglobal\<\bbbl@ensure@language\space>%
2407   \fi
2408 % ==
2409 % At this point all parameters are defined if 'import'. Now we
2410 % execute some code depending on them. But what about if nothing was
2411 % imported? We just set the basic parameters, but still loading the
2412 % whole ini file.
2413 \bbbl@load@basic{#2}%
2414 % == script, language ==
2415 % Override the values from ini or defines them
2416 \ifx\bbbl@KVP@script\@nnil\else
2417   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2418 \fi
2419 \ifx\bbbl@KVP@language\@nnil\else
2420   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2421 \fi
2422 \ifcase\bbbl@engine\or
2423   \bbbl@ifunset{\bbbl@chrng@language}{}%
2424   {\directlua{
2425     Babel.set_chrnges_b('\bbbl@cl{sbcpr}', '\bbbl@cl{chrng}') }}%
2426 \fi
2427 % == onchar ==
2428 \ifx\bbbl@KVP@onchar\@nnil\else
2429   \bbbl@luahyphenate
2430   \bbbl@exp{%
2431     \AddToHook{env/document/before}{\select@language{#2}}}%
2432   \directlua{
2433     if Babel.locale_mapped == nil then
2434       Babel.locale_mapped = true
2435       Babel.linebreaking.add_before(Babel.locale_map)
2436       Babel.loc_to_scr = {}

```

```

2437     Babel.chr_to_loc = Babel.chr_to_loc or {}
2438   end}%
2439   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2440   \ifin@
2441     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2442       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2443     \fi
2444     \bbl@exp{\bbl@add\bbl@starthyphens
2445       {\bbl@patterns@lua{\language}}}%
2446     % TODO - error/warning if no script
2447     \directlua{
2448       if Babel.script_blocks['\bbl@cl{sbc}'] then
2449         Babel.loc_to_scr[\the\localeid] =
2450           Babel.script_blocks['\bbl@cl{sbc}']
2451         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2452         Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
2453       end
2454     }%
2455   \fi
2456   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2457   \ifin@
2458     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{%
2459     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{%
2460     \directlua{
2461       if Babel.script_blocks['\bbl@cl{sbc}'] then
2462         Babel.loc_to_scr[\the\localeid] =
2463           Babel.script_blocks['\bbl@cl{sbc}']
2464       end}%
2465     \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2466       \AtBeginDocument{%
2467         \bbl@patchfont{\bbl@mapselect}%
2468         {\selectfont}}%
2469       \def\bbl@mapselect{%
2470         \let\bbl@mapselect\relax
2471         \edef\bbl@prefontid{\fontid\font}%
2472       \def\bbl@mapdir##1{%
2473         {\def\language{##1}%
2474         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2475         \bbl@switchfont
2476         \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2477           \directlua{
2478             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2479             ['/\bbl@prefontid'] = \fontid\font\space}%
2480           \fi}}%
2481       \fi
2482       \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2483     \fi
2484     % TODO - catch non-valid values
2485   \fi
2486   % == mapfont ==
2487   % For bidi texts, to switch the font based on direction
2488   \ifx\bbl@KVP@mapfont\@nnil\else
2489     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{%
2490       {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2491         mapfont. Use 'direction'.%
2492         {See the manual for details.}}}%
2493     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{%
2494     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{%
2495     \ifx\bbl@mapselect\undefined % TODO. See onchar.
2496       \AtBeginDocument{%
2497         \bbl@patchfont{\bbl@mapselect}%
2498         {\selectfont}}%
2499       \def\bbl@mapselect{%

```

```

2500     \let\bbl@mapselect\relax
2501     \edef\bbl@prefontid{\fontid\font}}%
2502     \def\bbl@mapdir##1{%
2503       {\def\language\language##1}%
2504       \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2505       \bbl@switchfont
2506       \directlua{Babel.fontmap
2507         [\the\csname bbl@wdir@##1\endcsname]%
2508         [\bbl@prefontid]=\fontid\font}}}%
2509     \fi
2510     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir\language}}}%
2511   \fi
2512   % == Line breaking: intraspace, intrapenalty ==
2513   % For CJK, East Asian, Southeast Asian, if interspace in ini
2514   \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2515     \bbl@csarg\edef{intsp#2}{\bbl@KVP@intraspace}%
2516   \fi
2517   \bbl@provide@intraspace
2518   % == Line breaking: CJK quotes ==
2519   \ifcase\bbl@engine\or
2520     \bbl@xin@{/c}{\bbl@cl{lnbrk}}%
2521   \ifin@
2522     \bbl@ifunset{bbl@quote@\language}{}%
2523     {\directlua{
2524       Babel.locale_props[\the\localeid].cjk_quotes = {}
2525       local cs = 'op'
2526       for c in string.utfvalues(
2527         [[\csname bbl@quote@\language\endcsname]]) do
2528         if Babel.cjk_characters[c].c == 'qu' then
2529           Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2530         end
2531         cs = ( cs == 'op') and 'cl' or 'op'
2532       end
2533     }}%
2534   \fi
2535   \fi
2536   % == Line breaking: justification ==
2537   \ifx\bbl@KVP@justification\@nnil\else
2538     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2539   \fi
2540   \ifx\bbl@KVP@linebreaking\@nnil\else
2541     \bbl@xin@{\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2542   \ifin@
2543     \bbl@csarg\xdef
2544       {\lnbrk@\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2545   \fi
2546   \fi
2547   \bbl@xin@{/e}{\bbl@cl{lnbrk}}%
2548   \ifin@\else\bbl@xin@{/k}{\bbl@cl{lnbrk}}\fi
2549   \ifin@\bbl@arabicjust\fi
2550   % == Line breaking: hyphenate.other.(locale|script) ==
2551   \ifx\bbl@lbkflag\@empty
2552     \bbl@ifunset{bbl@hyotl@\language}{}%
2553     {\bbl@csarg\bbl@replace{hyotl@\language}{ }{,}%
2554       \bbl@startcommands*\language}%
2555     \bbl@csarg\bbl@foreach{hyotl@\language}{%
2556       \ifcase\bbl@engine
2557         \ifnum##1<257
2558           \SetHyphenMap{\BabelLower{##1}{##1}}%
2559         \fi
2560       \else
2561         \SetHyphenMap{\BabelLower{##1}{##1}}%
2562       \fi}%

```

```

2563 \bbl@endcommands}%
2564 \bbl@ifunset{\bbl@hyots@language}{}%
2565 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2566 \bbl@csarg\bbl@foreach{hyots@language}{}%
2567 \ifcase\bbl@engine
2568 \ifnum##1<257
2569 \global\lccode##1=##1\relax
2570 \fi
2571 \else
2572 \global\lccode##1=##1\relax
2573 \fi}%
2574 \fi
2575 % == Counters: maparabic ==
2576 % Native digits, if provided in ini (TeX level, xe and lua)
2577 \ifcase\bbl@engine\else
2578 \bbl@ifunset{\bbl@dgnat@language}{}%
2579 {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
2580 \expandafter\expandafter\expandafter
2581 \bbl@setdigits\csname bbl@dgnat@language\endcsname
2582 \ifx\bbl@KVP@maparabic\@nnil\else
2583 \ifx\bbl@latinarabic\@undefined
2584 \expandafter\let\expandafter\@arabic
2585 \csname bbl@counter@language\endcsname
2586 \else % ie, if layout=counters, which redefines \@arabic
2587 \expandafter\let\expandafter\bbl@latinarabic
2588 \csname bbl@counter@language\endcsname
2589 \fi
2590 \fi
2591 \fi}%
2592 \fi
2593 % == Counters: mapdigits ==
2594 % Native digits (lua level).
2595 \ifodd\bbl@engine
2596 \ifx\bbl@KVP@mapdigits\@nnil\else
2597 \bbl@ifunset{\bbl@dgnat@language}{}%
2598 {\RequirePackage{luatexbase}%
2599 \bbl@activate@preotf
2600 \directlua{
2601 Babel = Babel or {} %%% -> presets in luababel
2602 Babel.digits_mapped = true
2603 Babel.digits = Babel.digits or {}
2604 Babel.digits[\the\localeid] =
2605 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2606 if not Babel.numbers then
2607 function Babel.numbers(head)
2608 local LOCALE = Babel.attr_locale
2609 local GLYPH = node.id'glyph'
2610 local inmath = false
2611 for item in node.traverse(head) do
2612 if not inmath and item.id == GLYPH then
2613 local temp = node.get_attribute(item, LOCALE)
2614 if Babel.digits[temp] then
2615 local chr = item.char
2616 if chr > 47 and chr < 58 then
2617 item.char = Babel.digits[temp][chr-47]
2618 end
2619 end
2620 elseif item.id == node.id'math' then
2621 inmath = (item.subtype == 0)
2622 end
2623 end
2624 return head
2625 end

```

```

2626         end
2627     }}%
2628     \fi
2629 \fi
2630 % == Counters: alph, Alph ==
2631 % What if extras<lang> contains a \babel@save\@alph? It won't be
2632 % restored correctly when exiting the language, so we ignore
2633 % this change with the \bbl@alph@saved trick.
2634 \ifx\bbl@KVP@alph\@nnil\else
2635     \bbl@extras@wrap{\bbl@alph@saved}%
2636     {\let\bbl@alph@saved\@alph}%
2637     {\let\@alph\bbl@alph@saved
2638     \babel@save\@alph}%
2639     \bbl@exp{%
2640         \bbl@add\<extras\languagename>{%
2641             \let\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2642 \fi
2643 \ifx\bbl@KVP@Alph\@nnil\else
2644     \bbl@extras@wrap{\bbl@Alph@saved}%
2645     {\let\bbl@Alph@saved\@Alph}%
2646     {\let\@Alph\bbl@Alph@saved
2647     \babel@save\@Alph}%
2648     \bbl@exp{%
2649         \bbl@add\<extras\languagename>{%
2650             \let\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2651 \fi
2652 % == Calendars ==
2653 \ifx\bbl@KVP@calendar\@nnil
2654     \edef\bbl@KVP@calendar{\bbl@c1{calpr}}}%
2655 \fi
2656 \def\bbl@tempe##1 ##2\@{% Get first calendar
2657     \def\bbl@tempa{##1}}%
2658     \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@}%
2659 \def\bbl@tempe##1.##2.##3\@{%
2660     \def\bbl@tempc{##1}%
2661     \def\bbl@tempb{##2}}%
2662 \expandafter\bbl@tempe\bbl@tempa..\@
2663 \bbl@csarg\edef{calpr\languagename}{%
2664     \ifx\bbl@tempc\@empty\else
2665         calendar=\bbl@tempc
2666     \fi
2667     \ifx\bbl@tempb\@empty\else
2668         ,variant=\bbl@tempb
2669     \fi}%
2670 % == require.babel in ini ==
2671 % To load or reload the babel-*.tex, if require.babel in ini
2672 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2673     \bbl@ifunset{bbl@rqtex\languagename}{}%
2674     {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2675         \let\BabelBeforeIni\gobbletwo
2676         \chardef\atcatcode=\catcode`\@
2677         \catcode`\@=11\relax
2678         \bbl@inputtexini{\bbl@cs{rqtex@\languagename}}%
2679         \catcode`\@=\atcatcode
2680         \let\atcatcode\relax
2681         \global\bbl@csarg\let{rqtex@\languagename}\relax
2682     \fi}%
2683 \bbl@foreach\bbl@calendars{%
2684     \bbl@ifunset{bbl@ca##1}{%
2685         \chardef\atcatcode=\catcode`\@
2686         \catcode`\@=11\relax
2687         \InputIfFileExists{babel-ca-##1.tex}{}}%
2688     \catcode`\@=\atcatcode

```

```

2689     \let\atcatcode\relax}%
2690   }%}}%
2691 \fi
2692 % == frenchspacing ==
2693 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2694 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2695 \ifin@
2696   \bbbl@extras@wrap{\bbbl@pre@fs}%
2697   {\bbbl@pre@fs}%
2698   {\bbbl@post@fs}%
2699 \fi
2700 % == Release saved transforms ==
2701 \bbbl@release@transforms\relax % \relax closes the last item.
2702 % == main ==
2703 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2704   \let\language\bbbl@savelangname
2705   \chardef\localeid\bbbl@savelocaleid\relax
2706 \fi}

```

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

```

2707 \def\bbbl@provide@new#1{%
2708   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2709   \@namedef{extras#1}{}%
2710   \@namedef{noextras#1}{}%
2711   \bbbl@startcommands*{#1}{captions}%
2712   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2713     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2714       \ifx##1\@empty\else
2715         \bbbl@exp{%
2716           \\\SetString\\##1{%
2717             \\\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2718           \expandafter\bbbl@tempb
2719         \fi}%
2720     \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2721   \else
2722     \ifx\bbbl@initoload\relax
2723       \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2724     \else
2725       \bbbl@read@ini{\bbbl@initoload}2% % Same
2726     \fi
2727   \fi
2728   \StartBabelCommands*{#1}{date}%
2729   \ifx\bbbl@KVP@date\@nnil
2730     \bbbl@exp{%
2731       \\\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2732   \else
2733     \bbbl@savetoday
2734     \bbbl@savedate
2735   \fi
2736   \bbbl@endcommands
2737   \bbbl@load@basic{#1}%
2738   % == hyphenmins == (only if new)
2739   \bbbl@exp{%
2740     \gdef\<#1hyphenmins>{%
2741       {\bbbl@ifunset{\bbbl@lftm#1}{2}{\bbbl@cs{lftm#1}}}%
2742       {\bbbl@ifunset{\bbbl@rgtm#1}{3}{\bbbl@cs{rgtm#1}}}}}%
2743   % == hyphenrules (also in renew) ==
2744   \bbbl@provide@hyphens{#1}%
2745   \ifx\bbbl@KVP@main\@nnil\else
2746     \expandafter\main@language\expandafter{#1}%
2747   \fi}
2748 %

```



```

2749 \def\bbl@provide@renew#1{%
2750   \ifx\bbl@KVP@captions\@nnil\else
2751     \StartBabelCommands*{#1}{captions}%
2752     \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2753     \EndBabelCommands
2754   \fi
2755   \ifx\bbl@KVP@date\@nnil\else
2756     \StartBabelCommands*{#1}{date}%
2757     \bbl@savetoday
2758     \bbl@savestate
2759     \EndBabelCommands
2760   \fi
2761   % == hyphenrules (also in new) ==
2762   \ifx\bbl@lbkflag\@empty
2763     \bbl@provide@hyphens{#1}%
2764   \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.)

```

2765 \def\bbl@load@basic#1{%
2766   \ifcase\bbl@howloaded\or\or
2767     \ifcase\csname bbl@llevel\language\endcsname
2768       \bbl@csarg\let{lname@\language}\relax
2769     \fi
2770   \fi
2771   \bbl@ifunset{\bbl@lname@#1}%
2772   {\def\BabelBeforeIni##1##2{%
2773     \begingroup
2774       \let\bbl@ini@captions@aux\@gobbles
2775       \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
2776       \bbl@read@ini{##1}1%
2777       \ifx\bbl@initoload\relax\endinput\fi
2778     \endgroup}%
2779     \begingroup   % boxed, to avoid extra spaces:
2780     \ifx\bbl@initoload\relax
2781       \bbl@input@texini{#1}%
2782     \else
2783       \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2784     \fi
2785   \endgroup}%
2786   {}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2787 \def\bbl@provide@hyphens#1{%
2788   \let\bbl@tempa\relax
2789   \ifx\bbl@KVP@hyphenrules\@nnil\else
2790     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2791     \bbl@foreach\bbl@KVP@hyphenrules{%
2792       \ifx\bbl@tempa\relax   % if not yet found
2793         \bbl@ifsamestring{##1}{+}%
2794         {\bbl@exp{\addlanguage\<l@##1>}}}%
2795       {}%
2796       \bbl@ifunset{l@##1}%
2797       {}%
2798       {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
2799     \fi}%
2800   \fi
2801   \ifx\bbl@tempa\relax % if no opt or no language in opt found
2802     \ifx\bbl@KVP@import\@nnil
2803       \ifx\bbl@initoload\relax\else
2804         \bbl@exp{%
2805           \bbl@ifblank{\bbl@cs{hyphr@#1}}%
2806           {}%

```

```

2807         {\let\\bbl@tempa<l@\\bbl@c1{hyphr}>}}}%
2808     \fi
2809     \else % if importing
2810         \bbl@exp{%                and hyphenrules is not empty
2811             \\bbl@ifblank{\\bbl@cs{hyphr@#1}}}%
2812             }%
2813             {\let\\bbl@tempa<l@\\bbl@c1{hyphr}>}}}%
2814     \fi
2815 \fi
2816 \bbl@ifunset{bbl@tempa}%        ie, relax or undefined
2817 {\bbl@ifunset{l@#1}%            no hyphenrules found - fallback
2818     {\bbl@exp{\\adddialect<l@#1>\language}}}%
2819     }}%                          so, l@<lang> is ok - nothing to do
2820 {\bbl@exp{\\adddialect<l@#1>\bbl@tempa}}}% found in opt list or ini

```

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

```

2821 \def\bbl@input@texini#1{%
2822     \bbl@bsphack
2823     \bbl@exp{%
2824         \catcode\\%=14 \catcode\\=0
2825         \catcode\\={1 \catcode\\}=2
2826         \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2827         \catcode\\%=\\the\catcode`%\relax
2828         \catcode\\=\\the\catcode`\\relax
2829         \catcode\\={\\the\catcode`%\relax
2830         \catcode\\}=\\the\catcode`%\relax}%
2831     \bbl@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 \bbl@read@ini.

```

2832 \def\bbl@iniline#1\bbl@iniline{%
2833     \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2834 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2835 \def\bbl@iniskip#1\@@{%          if starts with ;
2836 \def\bbl@inistore#1=#2\@@{%      full (default)
2837     \bbl@trim@def\bbl@tempa{#1}%
2838     \bbl@trim\toks@{#2}%
2839     \bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2840     \ifin@\\else
2841         \bbl@xin@{,identification/include.}%
2842         {\bbl@section/\bbl@tempa}%
2843     \ifin@\\edef\bbl@required@inis{\the\toks@}\fi
2844     \bbl@exp{%
2845         \\g@addto@macro\\bbl@inidata{%
2846             \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2847     \fi}
2848 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2849     \bbl@trim@def\bbl@tempa{#1}%
2850     \bbl@trim\toks@{#2}%
2851     \bbl@xin@{.identification.}{.\bbl@section.}%
2852     \ifin@
2853         \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2854             \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2855     \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.

```

2856 \def\bbl@loop@ini{%
2857     \loop

```

```

2858 \if T\ifeof\bb@readstream F\fi T\relax % Trick, because inside \loop
2859 \endlinechar\m@ne
2860 \read\bb@readstream to \bb@line
2861 \endlinechar`^^M
2862 \ifx\bb@line\@empty\else
2863 \expandafter\bb@iniline\bb@line\bb@iniline
2864 \fi
2865 \repeat}
2866 \ifx\bb@readstream\@undefined
2867 \csname newread\endcsname\bb@readstream
2868 \fi
2869 \def\bb@read@ini#1#2{%
2870 \global\let\bb@extend@ini\@gobble
2871 \openin\bb@readstream=babel-#1.ini
2872 \ifeof\bb@readstream
2873 \bb@error
2874 {There is no ini file for the requested language\\%
2875 (#1: \language). Perhaps you misspelled it or your\\%
2876 installation is not complete.}%
2877 {Fix the name or reinstall babel.}%
2878 \else
2879 % == Store ini data in \bb@inidata ==
2880 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2881 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2882 \bb@info{Importing
2883 \ifcase#2font and identification \or basic \fi
2884 data for \language\\%
2885 from babel-#1.ini. Reported}%
2886 \ifnum#2=\z@
2887 \global\let\bb@inidata\@empty
2888 \let\bb@inistore\bb@inistore@min % Remember it's local
2889 \fi
2890 \def\bb@section{identification}%
2891 \let\bb@required@inis\@empty
2892 \bb@exp{\\bb@inistore tag.ini=#1\\@@}%
2893 \bb@inistore load.level=#2\\@@
2894 \bb@loop@ini
2895 \ifx\bb@required@inis\@empty\else
2896 \bb@replace\bb@required@inis{ }{,}%
2897 \bb@foreach\bb@required@inis{%
2898 \openin\bb@readstream=##1.ini
2899 \bb@loop@ini}%
2900 \fi
2901 % == Process stored data ==
2902 \bb@csarg\xdef{lini@\language}{#1}%
2903 \bb@read@ini@aux
2904 % == 'Export' data ==
2905 \bb@ini@exports{#2}%
2906 \global\bb@csarg\let{inidata@\language}\bb@inidata
2907 \global\let\bb@inidata\@empty
2908 \bb@exp{\\bb@add@list\\bb@ini@loaded{\language}}%
2909 \bb@toglobal\bb@ini@loaded
2910 \fi}
2911 \def\bb@read@ini@aux{%
2912 \let\bb@savestrings\@empty
2913 \let\bb@savetoday\@empty
2914 \let\bb@savestate\@empty
2915 \def\bb@elt##1##2##3{%
2916 \def\bb@section{##1}%
2917 \in@{=date.}{##1}% Find a better place
2918 \ifin@
2919 \bb@ifunset{bb@inikv@##1}%
2920 {\bb@ini@calendar{##1}}%

```

```

2921     {}%
2922 \fi
2923 \in@{=identification/extension.}{=##1/##2}%
2924 \ifin@
2925     \bbl@ini@extension{##2}%
2926 \fi
2927 \bbl@ifunset{bbl@inikv@##1}{}%
2928     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2929 \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.

```

2930 \def\bbl@extend@ini@aux#1{%
2931     \bbl@startcommands*{#1}{captions}%
2932     % Activate captions/... and modify exports
2933     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2934         \setlocalecaption{#1}{##1}{##2}%
2935     \def\bbl@inikv@captions##1##2{%
2936         \bbl@ini@captions@aux{##1}{##2}%
2937     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2938     \def\bbl@exportkey##1##2##3{%
2939         \bbl@ifunset{bbl@kv@##2}{}%
2940         {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2941             \bbl@exp{\global\let\<bbl@##1@language\>\<bbl@kv@##2>}%
2942             \fi}}%
2943     % As with \bbl@read@ini, but with some changes
2944     \bbl@read@ini@aux
2945     \bbl@ini@exports\tw@
2946     % Update inidata@lang by pretending the ini is read.
2947     \def\bbl@elt##1##2##3{%
2948         \def\bbl@section{##1}%
2949         \bbl@iniline##2=##3\bbl@iniline}%
2950     \csname bbl@inidata@#1\endcsname
2951     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2952     \StartBabelCommands*{#1}{date}% And from the import stuff
2953     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2954     \bbl@savetoday
2955     \bbl@savedate
2956     \bbl@endcommands}

```

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

```

2957 \def\bbl@ini@calendar#1{%
2958     \lowercase{\def\bbl@tempa{=##1=}}%
2959     \bbl@replace\bbl@tempa{=date.gregorian}{}%
2960     \bbl@replace\bbl@tempa{=date.}{}%
2961     \in@{.licr=}{#1}%
2962     \ifin@
2963         \ifcase\bbl@engine
2964             \bbl@replace\bbl@tempa{.licr=}{}%
2965         \else
2966             \let\bbl@tempa\relax
2967         \fi
2968     \fi
2969     \ifx\bbl@tempa\relax\else
2970         \bbl@replace\bbl@tempa{=}{}%
2971         \ifx\bbl@tempa\@empty\else
2972             \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2973         \fi
2974         \bbl@exp{%
2975             \def\<bbl@inikv@#1>####1####2{%
2976                 \bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2977     \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).

```

2978 \def\bbl@renewinikey#1/#2\@#3{%
2979   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2980   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2981   \bbl@trim\toks@{#3}%                       value
2982   \bbl@exp{%
2983     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2984     \\g@addto@macro\\bbl@inidata{%
2985       \\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.

```

2986 \def\bbl@exportkey#1#2#3{%
2987   \bbl@ifunset{\bbl@kv@#2}%
2988   {\bbl@csarg\gdef{#1@\language}\@empty}%
2989   {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2990     \bbl@csarg\gdef{#1@\language}\@empty}%
2991   \else
2992     \bbl@exp{\global\let<\bbl@#1@\language>\<\bbl@kv@#2>}%
2993   \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.

```

2994 \def\bbl@iniwarning#1{%
2995   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2996   {\bbl@warning{%
2997     From babel-\bbl@cs{lini@\language}.ini:\%
2998     \bbl@cs{@kv@identification.warning#1}\%
2999     Reported }}}
3000 %
3001 \let\bbl@release@transforms\@empty

```

BCP 47 extensions are separated by a single letter (eg, `latin-x-medieval`). The following macro handles this special case to create correctly the correspondig info.

```

3002 \def\bbl@ini@extension#1{%
3003   \def\bbl@tempa{#1}%
3004   \bbl@replace\bbl@tempa{extension.}{}%
3005   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
3006   \bbl@ifunset{\bbl@info@#1}%
3007   {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
3008     \bbl@exp{%
3009       \\g@addto@macro\\bbl@moreinfo{%
3010         \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
3011   {}}
3012 \let\bbl@moreinfo\@empty
3013 %
3014 \def\bbl@ini@exports#1{%
3015   % Identification always exported
3016   \bbl@iniwarning{}%
3017   \ifcase\bbl@engine
3018     \bbl@iniwarning{.pdflatex}%
3019   \or
3020     \bbl@iniwarning{.lualatex}%
3021   \or
3022     \bbl@iniwarning{.xelatex}%
3023   \fi%
3024   \bbl@exportkey{llevel}{identification.load.level}{}%
3025   \bbl@exportkey{elname}{identification.name.english}{}%
3026   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
3027     {\csname \bbl@elname@\language\endcsname}}%
3028   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%

```

```

3029 \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
3030 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3031 \bbl@exportkey{esname}{identification.script.name}{}%
3032 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3033   {\csname bbl@esname@languagename\endcsname}}%
3034 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3035 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3036 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3037 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
3038 \bbl@moreinfo
3039 % Also maps bcp47 -> languagename
3040 \ifbbl@bcptoname
3041   \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpr}}{\languagename}%
3042 \fi
3043 % Conditional
3044 \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
3045   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3046   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3047   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3048   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3049   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3050   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3051   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3052   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3053   \bbl@exportkey{intsp}{typography.intraspaces}{}%
3054   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3055   \bbl@exportkey{chrng}{characters.ranges}{}%
3056   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3057   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3058   \ifnum#1=\tw@       % only (re)new
3059     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3060     \bbl@tglobal\bbl@savetoday
3061     \bbl@tglobal\bbl@savestate
3062     \bbl@savestrings
3063   \fi
3064 \fi}

```

A shared handler for key=val lines to be stored in \bbl@kv@<section>.<key>.

```

3065 \def\bbl@inikv#1#2{%      key=value
3066   \toks@{#2}%             This hides #'s from ini values
3067   \bbl@csarg\xdef{@kv@bbl@section.#1}{\the\toks@}}

```

By default, the following sections are just read. Actions are taken later.

```

3068 \let\bbl@inikv@identification\bbl@inikv
3069 \let\bbl@inikv@date\bbl@inikv
3070 \let\bbl@inikv@typography\bbl@inikv
3071 \let\bbl@inikv@characters\bbl@inikv
3072 \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 \localenumerals, and another one preserving the trailing .1 for the ‘units’.

```

3073 \def\bbl@inikv@counters#1#2{%
3074   \bbl@ifsamestring{#1}{digits}%
3075   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3076     decimal digits}%
3077     {Use another name.}}%
3078   }%
3079   \def\bbl@tempc{#1}%
3080   \bbl@trim@def{\bbl@tempb*}{#2}%
3081   \in@{.1$}{#1$}%
3082   \ifin@
3083     \bbl@replace\bbl@tempc{.1}{}%
3084     \bbl@csarg\xprotected@xdef{cntr@bbl@tempc @\languagename}{%

```

```

3085 \noexpand\bb1@alphanumeric{\bb1@tempc}}%
3086 \fi
3087 \in@{.F.}{#1}%
3088 \ifin@ \else \in@{.S.}{#1} \fi
3089 \ifin@
3090 \bb1@csarg\protected@xdef{cntr@#1@\language name}{\bb1@tempb*}%
3091 \else
3092 \toks@{}% Required by \bb1@buildifcase, which returns \bb1@tempa
3093 \expandafter\bb1@buildifcase\bb1@tempb* \ \ % Space after \
3094 \bb1@csarg{\global\expandafter\let}{cntr@#1@\language name}\bb1@tempa
3095 \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.

```

3096 \ifcase\bb1@engine
3097 \bb1@csarg\def{inikv@captions.licr}#1#2{%
3098 \bb1@ini@captions@aux{#1}{#2}}
3099 \else
3100 \def\bb1@inikv@captions#1#2{%
3101 \bb1@ini@captions@aux{#1}{#2}}
3102 \fi

```

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

```

3103 \def\bb1@ini@captions@template#1#2{% string language tempa=capt-name
3104 \bb1@replace\bb1@tempa{.template}{}}%
3105 \def\bb1@toreplace{#1}{}%
3106 \bb1@replace\bb1@toreplace{[ ]}{\nobreakspace}}%
3107 \bb1@replace\bb1@toreplace{[ ]}{\csname}%
3108 \bb1@replace\bb1@toreplace{[ ]}{\csname the}%
3109 \bb1@replace\bb1@toreplace{[ ]}{name\endcsname}}%
3110 \bb1@replace\bb1@toreplace{[ ]}{\endcsname}}%
3111 \bb1@xin@{,\bb1@tempa,}{,chapter,appendix,part,}%
3112 \ifin@
3113 \@nameuse{\bb1@patch\bb1@tempa}%
3114 \global\bb1@csarg\let{\bb1@tempa fmt@#2}\bb1@toreplace
3115 \fi
3116 \bb1@xin@{,\bb1@tempa,}{,figure,table,}%
3117 \ifin@
3118 \toks@\expandafter{\bb1@toreplace}%
3119 \bb1@exp{\gdef\<fnum@\bb1@tempa>{\the\toks@}}%
3120 \fi}
3121 \def\bb1@ini@captions@aux#1#2{%
3122 \bb1@trim@def\bb1@tempa{#1}%
3123 \bb1@xin@{.template}{\bb1@tempa}%
3124 \ifin@
3125 \bb1@ini@captions@template{#2}\language name
3126 \else
3127 \bb1@ifblank{#2}%
3128 {\bb1@exp{%
3129 \toks@{\bb1@nocaption{\bb1@tempa}{\language name\bb1@tempa name}}}%
3130 {\bb1@trim\toks@{#2}}}%
3131 \bb1@exp{%
3132 \bb1@add\bb1@savestrings{%
3133 \SetString\<\bb1@tempa name>{\the\toks@}}}%
3134 \toks@\expandafter{\bb1@captionslist}%
3135 \bb1@exp{\in@{\<\bb1@tempa name>}{\the\toks@}}%
3136 \ifin@ \else
3137 \bb1@exp{%
3138 \bb1@add\<\bb1@extracaps@\language name>{\<\bb1@tempa name>}}%
3139 \bb1@toglobal\<\bb1@extracaps@\language name>}}%
3140 \fi
3141 \fi}

```

Labels. Captions must contain just strings, no format at all, so there is new group in ini files.

```

3142 \def\bbl@list@the{%
3143   part,chapter,section,subsection,subsubsection,paragraph,%
3144   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3145   table,page,footnote,mpfootnote,mpfn}
3146 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3147   \bbl@ifunset{\bbl@map@#1\language}%
3148     {\@nameuse{#1}}%
3149     {\@nameuse{\bbl@map@#1\language}}}%
3150 \def\bbl@inikv@labels#1#2{%
3151   \in@{.map}{#1}%
3152   \ifin@
3153     \ifx\bbl@KVP@labels\@nnil\else
3154       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3155       \ifin@
3156         \def\bbl@tempc{#1}%
3157         \bbl@replace\bbl@tempc{.map}{}%
3158         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3159         \bbl@exp{%
3160           \gdef\<bbl@map@\bbl@tempc @\language>%
3161             {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3162         \bbl@foreach\bbl@list@the{%
3163           \bbl@ifunset{the##1}{}%
3164             {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3165               \bbl@exp{%
3166                 \\bbl@sreplace\<the##1>%
3167                 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3168                 \\bbl@sreplace\<the##1>%
3169                 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3170             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3171               \toks@\expandafter\expandafter\expandafter{%
3172                 \csname the##1\endcsname}%
3173               \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3174             \fi}}%
3175   \fi
3176 \fi
3177 %
3178 \else
3179   %
3180   % The following code is still under study. You can test it and make
3181   % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3182   % language dependent.
3183   \in@{enumerate.}{#1}%
3184   \ifin@
3185     \def\bbl@tempa{#1}%
3186     \bbl@replace\bbl@tempa{enumerate.}{}%
3187     \def\bbl@toreplace{#2}%
3188     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3189     \bbl@replace\bbl@toreplace{[]}{\csname the}%
3190     \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3191     \toks@\expandafter{\bbl@toreplace}%
3192     % TODO. Execute only once:
3193     \bbl@exp{%
3194       \\bbl@add\<extras\language>{%
3195         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3196         \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3197       \\bbl@tglobal\<extras\language>}%
3198   \fi
3199 \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.

```

3200 \def\bbl@chapttype{chapter}
3201 \ifx\@makechapterhead\@undefined
3202   \let\bbl@patchchapter\relax
3203 \else\ifx\thechapter\@undefined
3204   \let\bbl@patchchapter\relax
3205 \else\ifx\ps@headings\@undefined
3206   \let\bbl@patchchapter\relax
3207 \else
3208   \def\bbl@patchchapter{%
3209     \global\let\bbl@patchchapter\relax
3210     \gdef\bbl@chfmt{%
3211       \bbl@ifunset{\bbl@\bbl@chapttype fmt@\language\name}%
3212       {\@chapapp\space\thechapter}
3213       {\@nameuse{\bbl@\bbl@chapttype fmt@\language\name}}}%
3214     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3215     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3216     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3217     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3218     \bbl@tglobal\appendix
3219     \bbl@tglobal\ps@headings
3220     \bbl@tglobal\chaptermark
3221     \bbl@tglobal\@makechapterhead}
3222   \let\bbl@patchappendix\bbl@patchchapter
3223 \fi\fi\fi
3224 \ifx\@part\@undefined
3225   \let\bbl@patchpart\relax
3226 \else
3227   \def\bbl@patchpart{%
3228     \global\let\bbl@patchpart\relax
3229     \gdef\bbl@partformat{%
3230       \bbl@ifunset{\bbl@partfmt@\language\name}%
3231       {\partname\nobreakspace\thepart}
3232       {\@nameuse{\bbl@partfmt@\language\name}}}%
3233     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3234     \bbl@tglobal\@part}
3235 \fi

```

Date. Arguments (year, month, day) are *not* protected, on purpose. In \today, arguments are always gregorian, and therefore always converted with other calendars. TODO. Document

```

3236 \let\bbl@calendar\@empty
3237 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3238 \def\bbl@localedate#1#2#3#4{%
3239   \begingroup
3240     \edef\bbl@they{#2}%
3241     \edef\bbl@them{#3}%
3242     \edef\bbl@thed{#4}%
3243     \edef\bbl@tempe{%
3244       \bbl@ifunset{\bbl@calpr@\language\name}{\bbl@cl{calpr}},%
3245       #1}%
3246     \bbl@replace\bbl@tempe{ }{}%
3247     \bbl@replace\bbl@tempe{CONVERT}{convert}% Hackish
3248     \bbl@replace\bbl@tempe{convert}{convert}%
3249     \let\bbl@ld@calendar\@empty
3250     \let\bbl@ld@variant\@empty
3251     \let\bbl@ld@convert\relax
3252     \def\bbl@tempb##1=##2\@@{\@namedef{\bbl@ld@##1}{##2}}%
3253     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3254     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3255     \ifx\bbl@ld@calendar\@empty\else
3256       \ifx\bbl@ld@convert\relax\else
3257         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3258         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed

```

```

3259 \fi
3260 \fi
3261 \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3262 \edef\bbl@calendar{% Used in \month..., too
3263 \bbl@ld@calendar
3264 \ifx\bbl@ld@variant\@empty\else
3265 .\bbl@ld@variant
3266 \fi}%
3267 \bbl@cased
3268 {\@nameuse{bbl@date@\language name @\bbl@calendar}%
3269 \bbl@they\bbl@them\bbl@thed}%
3270 \endgroup}
3271 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3272 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3273 \bbl@trim@def\bbl@tempa{#1.#2}%
3274 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3275 {\bbl@trim@def\bbl@tempa{#3}%
3276 \bbl@trim\toks@{#5}%
3277 \@temptokena\expandafter{\bbl@savestate}%
3278 \bbl@exp{% Reverse order - in ini last wins
3279 \def\\bbl@savestate{%
3280 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3281 \the\@temptokena}}}%
3282 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3283 {\lowercase{\def\bbl@tempb{#6}}}%
3284 \bbl@trim@def\bbl@toreplace{#5}%
3285 \bbl@TG@@date
3286 \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3287 \ifx\bbl@savetoday\@empty
3288 \bbl@exp{% TODO. Move to a better place.
3289 \\AfterBabelCommands{%
3290 \def\<\language name date>{\\protect\<\language name date >}%
3291 \\newcommand\<\language name date >[4][]{%
3292 \\bbl@usedategroupttrue
3293 \<bbl@ensure@\language name>{%
3294 \\localedate[####1]{####2}{####3}{####4}}}%
3295 \def\\bbl@savetoday{%
3296 \\SetString\\today{%
3297 \<\language name date>[convert]%
3298 {\the\year}{\the\month}{\the\day}}}%
3299 \fi}%
3300 {}}}

```

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).

```

3301 \let\bbl@calendar\@empty
3302 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3303 \@nameuse{bbl@ca#2}#1\@}
3304 \newcommand\babelDateSpace{\nobreakspace}
3305 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3306 \newcommand\babelDated[1]{\number#1}
3307 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3308 \newcommand\babelDateM[1]{\number#1}
3309 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3310 \newcommand\babelDateMMMM[1]{%
3311 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3312 \newcommand\babelDatey[1]{\number#1}%
3313 \newcommand\babelDateyy[1]{%
3314 \ifnum#1<10 0\number#1 %
3315 \else\ifnum#1<100 \number#1 %

```

```

3316 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3317 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3318 \else
3319 \bbl@error
3320 {Currently two-digit years are restricted to the\
3321 range 0-9999.}%
3322 {There is little you can do. Sorry.}%
3323 \fi\fi\fi\fi}}
3324 \newcommand\BabelDateyyy[1]{\number#1} % TODO - add leading 0
3325 \def\bbl@replace@finish@iii#1{%
3326 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3327 \def\bbl@TG@date{%
3328 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3329 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot{}}%
3330 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3331 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3332 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3333 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3334 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3335 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3336 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3337 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3338 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecntr{####1|}}%
3339 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecntr{####2|}}%
3340 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecntr{####3|}}%
3341 \bbl@replace@finish@iii\bbl@toreplace}
3342 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3343 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3344 \let\bbl@release@transforms\empty
3345 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3346 \bbl@transforms\babelprehyphenation}
3347 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3348 \bbl@transforms\babelposthyphenation}
3349 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3350 #1[#2]{#3}{#4}{#5}}
3351 \begingroup % A hack. TODO. Don't require an specific order
3352 \catcode`\%=12
3353 \catcode`\&=14
3354 \gdef\bbl@transforms#1#2#3{%&
3355 \ifx\bbl@KVP@transforms\@nnil\else
3356 \directlua{
3357 local str = [=[#2]=]
3358 str = str:gsub('%.%d+%.%d+$', '')
3359 tex.print([[ \def\string\babeltempa{] .. str .. [{}]])
3360 }&
3361 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3362 \ifin@
3363 \in@{.0$}{#2$}&
3364 \ifin@
3365 \directlua{
3366 local str = string.match([[ \bbl@KVP@transforms]],
3367 '%([^(%[-])%([%])[-\babeltempa]')
3368 if str == nil then
3369 tex.print([[ \def\string\babeltempb{] .. str .. [{}]])
3370 else
3371 tex.print([[ \def\string\babeltempb{,attribute=] .. str .. [{}]])
3372 end
3373 }
3374 \toks@{#3}&
3375 \bbl@exp{&
3376 \\\g@addto@macro\ \bbl@release@transforms{&

```

```

3377         \relax &% Closes previous \bbl@transforms@aux
3378         \\bbl@transforms@aux
3379         \\#1{label=\babeltempa\babeltempb}{\language\the\toks@}}&%
3380     \else
3381         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3382     \fi
3383 \fi
3384 \fi}
3385 \endgroup

```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```

3386 \def\bbl@provide@lsys#1{%
3387     \bbl@ifunset{bbl@lname@#1}%
3388     {\bbl@load@info{#1}}%
3389     {}%
3390     \bbl@csarg\let{lsys@#1}\@empty
3391     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3392     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3393     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3394     \bbl@ifunset{bbl@lname@#1}{}%
3395     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3396     \ifcase\bbl@engine\or\or
3397         \bbl@ifunset{bbl@prehc@#1}{}%
3398         {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3399         {}%
3400         {\ifx\bbl@xenohyph\@undefined
3401             \global\let\bbl@xenohyph\bbl@xenohyph@d
3402             \ifx\AtBeginDocument\@notprerr
3403                 \expandafter\@secondoftwo % to execute right now
3404             \fi
3405             \AtBeginDocument{%
3406                 \bbl@patchfont{\bbl@xenohyph}%
3407                 \expandafter\selectlanguage\expandafter{\language}}%
3408             \fi}%
3409     \fi
3410     \bbl@csarg\bbl@tglobal{lsys@#1}}
3411 \def\bbl@xenohyph@d{%
3412     \bbl@ifset{bbl@prehc\language}%
3413     {\ifnum\hyphenchar\font=\defaultshyphenchar
3414         \iffontchar\font\bbl@cl{prehc}\relax
3415         \hyphenchar\font\bbl@cl{prehc}\relax
3416     \else\iffontchar\font"200B
3417         \hyphenchar\font"200B
3418     \else
3419         \bbl@warning
3420         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3421         in the current font, and therefore the hyphen\\%
3422         will be printed. Try changing the fontspec's\\%
3423         'HyphenChar' to another value, but be aware\\%
3424         this setting is not safe (see the manual)}%
3425         \hyphenchar\font\defaultshyphenchar
3426     \fi\fi
3427     \fi}%
3428     {\hyphenchar\font\defaultshyphenchar}}
3429 % \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).

```

3430 \def\bbl@load@info#1{%
3431     \def\BabelBeforeIni##1##2{%
3432         \begingroup

```

```

3433 \bbl@read@ini{##1}0%
3434 \endinput % babel- .tex may contain onlypreamble's
3435 \endgroup}% boxed, to avoid extra spaces:
3436 {\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. The first macro is the generic “localized” command.

```

3437 \def\bbl@setdigits#1#2#3#4#5{%
3438 \bbl@exp{%
3439 \def<\language name digits>####1{% ie, \langdigits
3440 \<bbl@digits@language name>####1\\\nil}%
3441 \let<bbl@cntr@digits@language name>\<language name digits>%
3442 \def<\language name counter>####1{% ie, \langcounter
3443 \\\expandafter\<bbl@counter@language name>%
3444 \\\csname c@####1\endcsname}%
3445 \def<bbl@counter@language name>####1{% ie, \bbl@counter@lang
3446 \\\expandafter\<bbl@digits@language name>%
3447 \\\number####1\\\nil}}%
3448 \def\bbl@tempa##1##2##3##4##5{%
3449 \bbl@exp{% Wow, quite a lot of hashes! :-(
3450 \def<bbl@digits@language name>#####1{%
3451 \\\ifx#####1\\\nil % ie, \bbl@digits@lang
3452 \\\else
3453 \\\ifx0#####1#1%
3454 \\\else\\\ifx1#####1#2%
3455 \\\else\\\ifx2#####1#3%
3456 \\\else\\\ifx3#####1#4%
3457 \\\else\\\ifx4#####1#5%
3458 \\\else\\\ifx5#####1#1%
3459 \\\else\\\ifx6#####1#2%
3460 \\\else\\\ifx7#####1#3%
3461 \\\else\\\ifx8#####1#4%
3462 \\\else\\\ifx9#####1#5%
3463 \\\else#####1%
3464 \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3465 \\\expandafter\<bbl@digits@language name>%
3466 \\\fi}}}%
3467 \bbl@tempa}

```

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

```

3468 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3469 \ifx\\#1% % \ before, in case #1 is multiletter
3470 \bbl@exp{%
3471 \def\\bbl@tempa####1{%
3472 \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3473 \else
3474 \toks@\expandafter{\the\toks@\or #1}%
3475 \expandafter\bbl@buildifcase
3476 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left ‘unused’ in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as a special case, for a fixed form (see babel-he.ini, for example).

```

3477 \newcommand\localenumerat[2]{\bbl@cs{cntr@#1@language name}{#2}}
3478 \def\bbl@localecntr#1#2{\localenumerat{#2}{#1}}
3479 \newcommand\localecounter[2]{%
3480 \expandafter\bbl@localecntr
3481 \expandafter{\number\csname c@#2\endcsname}{#1}}
3482 \def\bbl@alphnumerat#1#2{%
3483 \expandafter\bbl@alphnumerat@i\number#2 76543210\@@{#1}}
3484 \def\bbl@alphnumerat@i#1#2#3#4#5#6#7#8\@@#9{%

```

```

3485 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3486 \bbl@alphnumeral@ii{#9}00000#1\or
3487 \bbl@alphnumeral@ii{#9}00000#1#2\or
3488 \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3489 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3490 \bbl@alphnum@invalid{>9999}%
3491 \fi}
3492 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3493 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3494 {\bbl@cs{cntr@#1.4@\language}#5%
3495 \bbl@cs{cntr@#1.3@\language}#6%
3496 \bbl@cs{cntr@#1.2@\language}#7%
3497 \bbl@cs{cntr@#1.1@\language}#8%
3498 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3499 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3500 {\bbl@cs{cntr@#1.S.321@\language}}%
3501 \fi}%
3502 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
3503 \def\bbl@alphnum@invalid#1{%
3504 \bbl@error{Alphabetic numeral too large (#1)}%
3505 {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.

```

3506 \def\bbl@localeinfo#1#2{%
3507 \bbl@ifunset{bbl@info@#2}{#1}%
3508 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}{#1}%
3509 {\bbl@cs{csname bbl@info@#2\endcsname @\language}}}%
3510 \newcommand\bbl@localeinfo[1]{%
3511 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3512 \bbl@afterelse\bbl@localeinfo{}%
3513 \else
3514 \bbl@localeinfo
3515 {\bbl@error{I've found no info for the current locale.\\%
3516 The corresponding ini file has not been loaded\\%
3517 Perhaps it doesn't exist}%
3518 {See the manual for details.}}%
3519 {#1}%
3520 \fi}
3521 % \@namedef{bbl@info@name.locale}{lname}
3522 \@namedef{bbl@info@tag.ini}{lini}
3523 \@namedef{bbl@info@name.english}{elname}
3524 \@namedef{bbl@info@name.opentype}{lname}
3525 \@namedef{bbl@info@tag.bcp47}{tbc}
3526 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3527 \@namedef{bbl@info@tag.opentype}{lotf}
3528 \@namedef{bbl@info@script.name}{esname}
3529 \@namedef{bbl@info@script.name.opentype}{sname}
3530 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3531 \@namedef{bbl@info@script.tag.opentype}{sotf}
3532 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3533 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3534 % Extensions are dealt with in a special way
3535 % Now, an internal \LaTeX{} macro:
3536 \providecommand\BCPdata[1]{\bbl@localeinfo*{#1.tag.bcp47}}

```

With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it.

```

3537 <<{*More package options}>> ≡
3538 \DeclareOption{ensureinfo=off}{}
3539 <</More package options>>
3540 %
3541 \let\bbl@ensureinfo\@gobble
3542 \newcommand\BabelEnsureInfo{%
3543 \ifx\InputIfFileExists\@undefined\else

```

```

3544 \def\bbl@ensureinfo##1{%
3545     \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3546 \fi
3547 \bbl@foreach\bbl@loaded{%
3548     \def\language{##1}%
3549     \bbl@ensureinfo{##1}}}%
3550 \@ifpackagewith{babel}{\ensureinfo=off}{}%
3551 {\AtEndOfPackage{% Test for plain.
3552     \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3553 \newcommand\getlocaleproperty{%
3554     \@ifstar\bbl@getproperty@s\bbl@getproperty@x%
3555 \def\bbl@getproperty@s#1#2#3{%
3556     \let#1\relax
3557     \def\bbl@elt##1##2##3{%
3558         \bbl@ifsamestring{##1/##2}{##3}%
3559         {\providecommand#1{##3}%
3560         \def\bbl@elt####1####2####3{}}}%
3561     }%
3562     \bbl@cs{inidata@#2}}%
3563 \def\bbl@getproperty@x#1#2#3{%
3564     \bbl@getproperty@s{#1}{#2}{#3}%
3565     \ifx#1\relax
3566         \bbl@error
3567             {Unknown key for locale '#2':\%
3568             #3\%
3569             \string#1 will be set to \relax}%
3570             {Perhaps you misspelled it.}%
3571     \fi}
3572 \let\bbl@ini@loaded\empty
3573 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

8 Adjusting the Babel bahavior

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

```

3574 \newcommand\babeladjust[1]{% TODO. Error handling.
3575     \bbl@forkv{#1}{%
3576         \bbl@ifunset{\bbl@ADJ@##1@##2}%
3577         {\bbl@cs{ADJ@##1}{##2}}%
3578         {\bbl@cs{ADJ@##1@##2}}}
3579 %
3580 \def\bbl@adjust@lua#1#2{%
3581     \ifvmode
3582         \ifnum\currentgrouplevel=\z@
3583             \directlua{ Babel.#2 }%
3584             \expandafter\expandafter\expandafter\@gobble
3585         \fi
3586     \fi
3587     {\bbl@error % The error is gobbled if everything went ok.
3588         {Currently, #1 related features can be adjusted only\%
3589         in the main vertical list.}%
3590         {Maybe things change in the future, but this is what it is.}}}
3591 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3592     \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3593 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3594     \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3595 \@namedef{\bbl@ADJ@bidi.text@on}{%
3596     \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3597 \@namedef{\bbl@ADJ@bidi.text@off}{%

```

```

3598 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3599 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3600 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3601 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3602 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3603 %
3604 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3605 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3606 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3607 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3608 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3609 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3610 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3611 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3612 \@namedef{bbl@ADJ@justify.arabic@on}{%
3613 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3614 \@namedef{bbl@ADJ@justify.arabic@off}{%
3615 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3616 %
3617 \def\bbl@adjust@layout#1{%
3618 \ifvmode
3619 #1%
3620 \expandafter\@gobble
3621 \fi
3622 {\bbl@error % The error is gobbled if everything went ok.
3623 {Currently, layout related features can be adjusted only\\%
3624 in vertical mode.}%
3625 {Maybe things change in the future, but this is what it is.}}
3626 \@namedef{bbl@ADJ@layout.tabular@on}{%
3627 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3628 \@namedef{bbl@ADJ@layout.tabular@off}{%
3629 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3630 \@namedef{bbl@ADJ@layout.lists@on}{%
3631 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3632 \@namedef{bbl@ADJ@layout.lists@off}{%
3633 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3634 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3635 \bbl@activateposthyphen}
3636 %
3637 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3638 \bbl@bcpallowedtrue}
3639 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3640 \bbl@bcpallowedfalse}
3641 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1{%
3642 \def\bbl@bcp@prefix{#1}}
3643 \def\bbl@bcp@prefix{bcp47-}
3644 \@namedef{bbl@ADJ@autoload.options#1{%
3645 \def\bbl@autoload@options{#1}}
3646 \let\bbl@autoload@bcptions\@empty
3647 \@namedef{bbl@ADJ@autoload.bcp47.options#1{%
3648 \def\bbl@autoload@bcptions{#1}}
3649 \newif\ifbbl@bcptoname
3650 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3651 \bbl@bcptonametrue
3652 \BabelEnsureInfo}
3653 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3654 \bbl@bcptonamefalse}
3655 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3656 \directlua{ Babel.ignore_pre_char = function(node)
3657 return (node.lang == \the\csname l@nohyphenation\endcsname)
3658 end }}
3659 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3660 \directlua{ Babel.ignore_pre_char = function(node)

```



```

3661     return false
3662   end }}
3663 \@namedef{bbl@ADJ@select.write@shift}{%
3664   \let\bbl@restorelastskip\relax
3665   \def\bbl@savelastskip{%
3666     \let\bbl@restorelastskip\relax
3667     \ifvmode
3668       \ifdim\lastskip=\z@
3669         \let\bbl@restorelastskip\nobreak
3670       \else
3671         \bbl@exp{%
3672           \def\\bbl@restorelastskip{%
3673             \skip=\the\lastskip
3674             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3675         \fi
3676       \fi}}
3677 \@namedef{bbl@ADJ@select.write@keep}{%
3678   \let\bbl@restorelastskip\relax
3679   \let\bbl@savelastskip\relax}
3680 \@namedef{bbl@ADJ@select.write@omit}{%
3681   \let\bbl@restorelastskip\relax
3682   \def\bbl@savelastskip##1\bbl@restorelastskip{}}

```

As the final task, load the code for lua. TODO: use babel name, override

```

3683 \ifx\directlua\undefined\else
3684   \ifx\bbl@luapatterns\undefined
3685     \input luababel.def
3686   \fi
3687 \fi

```

Continue with \LaTeX .

```

3688 </package | core>
3689 <*package>

```

8.1 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.

```

3690 <(*More package options)> ≡
3691 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3692 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3693 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3694 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3695 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3696 <(/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.

```

3697 \bbl@trace{Cross referencing macros}
3698 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3699   \def\@newl@bel#1#2#3{%
3700     {\@safe@activetrue
3701       \bbl@ifunset{#1@#2}%
3702       \relax

```

```

3703      {\gdef\@multiplelabels{%
3704        \@latex@warning@no@line{There were multiply-defined labels}}}%
3705        \@latex@warning@no@line{Label `#2' multiply defined}}}%
3706      \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.

```

3707  \CheckCommand*\@testdef[3]{%
3708    \def\reserved@a{#3}%
3709    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3710    \else
3711      \@tempswatrue
3712    \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.

```

3713  \def\@testdef#1#2#3{%  TODO. With @samestring?
3714    \@safe@activetrue
3715    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3716    \def\bbl@tempb{#3}%
3717    \@safe@activetrue
3718    \ifx\bbl@tempa\relax
3719    \else
3720      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3721    \fi
3722    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3723    \ifx\bbl@tempa\bbl@tempb
3724    \else
3725      \@tempswatrue
3726    \fi}
3727 \fi
```

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

```

3728 \bbl@xin@{R}\bbl@opt@safe
3729 \ifin@
3730   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3731   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3732   {\expandafter\strip@prefix\meaning\ref}%
3733   \ifin@
3734     \bbl@redefine\@kernel@ref#1{%
3735       \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3736     \bbl@redefine\@kernel@pageref#1{%
3737       \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3738     \bbl@redefine\@kernel@sref#1{%
3739       \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3740     \bbl@redefine\@kernel@spageref#1{%
3741       \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3742   \else
3743     \bbl@redefineroobust\ref#1{%
3744       \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3745     \bbl@redefineroobust\pageref#1{%
3746       \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3747   \fi
3748 \else
3749   \let\org@ref\ref
3750   \let\org@pageref\pageref
3751 \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.

```
3752 \bbl@xin@{B}\bbl@opt@safe
3753 \ifin@
3754 \bbl@redefine\@citex[#1]#2{%
3755 \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3756 \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.

```
3757 \AtBeginDocument{%
3758 \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.)

```
3759 \def\@citex[#1][#2]#3{%
3760 \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3761 \org@@citex[#1][#2]{\@tempa}}%
3762 }{}}
```

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

```
3763 \AtBeginDocument{%
3764 \ifpackageloaded{cite}{%
3765 \def\@citex[#1]#2{%
3766 \@safe@activestrue\org@@citex[#1][#2]\@safe@activesfalse}%
3767 }{}}
```

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

```
3768 \bbl@redefine\nocite#1{%
3769 \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3770 \bbl@redefine\bbl@cite{%
3771 \bbl@cite@choice
3772 \bbl@cite}
```

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

```
3773 \def\bbl@bbl@cite#1#2{%
3774 \org@bbl@cite{#1}{\@safe@activesfalse#2}}
```

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

```
3775 \def\bbl@cite@choice{%
3776 \global\let\bbl@cite\bbl@bbl@cite
3777 \ifpackageloaded{natbib}{\global\let\bbl@cite\org@bbl@cite}{%
3778 \ifpackageloaded{cite}{\global\let\bbl@cite\org@bbl@cite}{%
3779 \global\let\bbl@cite@choice\relax}
```

When a document is run for the first time, no .aux file is available, and \babcite will not yet be properly defined. In this case, this has to happen before the document starts.

```
3780 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3781 \bbl@redefine\@bibitem#1{%
3782   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3783 \else
3784   \let\org@nocite\nocite
3785   \let\org@@citex\@citex
3786   \let\org@babcite\babcite
3787   \let\org@@bibitem\@bibitem
3788 \fi
```

8.2 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.

```
3789 \bbl@trace{Marks}
3790 \IfBabelLayout{sectioning}
3791   {\ifx\bbl@opt@headfoot\@nnil
3792     \g@addto@macro\resetactivechars{%
3793       \set@typeset@protect
3794       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3795       \let\protect\noexpand
3796       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3797         \edef\thepage{%
3798           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3799       \fi}%
3800   \fi}
3801 {\ifbbl@single\else
3802   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3803     \markright#1{%
3804       \bbl@ifblank{#1}%
3805       {\org@markright{}}}%
3806       {\toks@{#1}%
3807        \bbl@exp{%
3808          \\\org@markright{\\\protect\\foreignlanguage{\language}%
3809            {\\\protect\\bbl@restore@actives\the\toks@}}}}}%
3810   \fi}
```

\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, \TeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```
3810 \ifx\@mkboth\markboth
3811   \def\bbl@tempc{\let\@mkboth\markboth}
3812 \else
3813   \def\bbl@tempc{}
3814 \fi
3815 \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3816   \markboth#1#2{%
3817     \protected@edef\bbl@tempb##1{%
3818       \protect\foreignlanguage
3819       {\language}\protect\bbl@restore@actives##1}}%
3820   \bbl@ifblank{#1}%
3821   {\toks@{}}%
```

```

3822      {\toks@\expandafter{\bbl@tempb{#1}}}%
3823      \bbl@ifblank{#2}%
3824      {\@temptokena{}}%
3825      {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3826      \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}%
3827      \bbl@tempc
3828      \fi} % end ifbbl@single, end \IfBabelLayout

```

8.3 Preventing clashes with other packages

8.3.1 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.

```

3829 \bbl@trace{Preventing clashes with other packages}
3830 \ifx\org@ref\undefined\else
3831   \bbl@xin@{R}\bbl@opt@safe
3832   \ifin@
3833     \AtBeginDocument{%
3834       \ifpackageloaded{ifthen}{%
3835         \bbl@redefine@long\ifthenelse#1#2#3{%
3836           \let\bbl@temp@pref\pageref
3837           \let\pageref\org@pageref
3838           \let\bbl@temp@ref\ref
3839           \let\ref\org@ref
3840           \@safe@activestrue
3841           \org@ifthenelse{#1}%
3842             {\let\pageref\bbl@temp@pref
3843              \let\ref\bbl@temp@ref
3844              \@safe@activesfalse
3845              #2}%
3846             {\let\pageref\bbl@temp@pref
3847              \let\ref\bbl@temp@ref
3848              \@safe@activesfalse
3849              #3}%
3850           }%
3851         }{}%
3852       }
3853 \fi

```

8.3.2 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`.

```

3854 \AtBeginDocument{%
3855   \ifpackageloaded{varioref}{%
3856     \bbl@redefine\@vpageref#1[#2]#3{%
3857       \@safe@activestrue

```

```

3858      \org@@@vpageref{#1}[#2]{#3}%
3859      \@safe@activesfalse}%
3860      \bbl@redefine\hrefpagenum#1#2{%
3861      \@safe@activestru
3862      \org@vrefpagenum{#1}{#2}%
3863      \@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.

```

3864      \expandafter\def\csname Ref \endcsname#1{%
3865      \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3866      }{}%
3867  }
3868 \fi

```

8.3.3 `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.

```

3869 \AtEndOfPackage{%
3870   \AtBeginDocument{%
3871     \@ifpackageloaded{hhline}%
3872     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3873       \else
3874         \makeatletter
3875         \def\@currname{hhline}\input{hhline.sty}\makeatother
3876         \fi}%
3877     {}}}

```

`\substitutefontfamily` Deprecated. Use the tools provided 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.

```

3878 \def\substitutefontfamily#1#2#3{%
3879   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3880   \immediate\write15{%
3881     \string\ProvidesFile{#1#2.fd}%
3882     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}]
3883     \space generated font description file]^^J
3884     \string\DeclareFontFamily{#1}{#2}{^^J
3885     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3886     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3887     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3888     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3889     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3890     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3891     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3892     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3893     }%
3894     \closeout15
3895   }
3896 \@onlypreamble\substitutefontfamily

```

8.4 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`

```

3897 \bbl@trace{Encoding and fonts}
3898 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3899 \newcommand\BabelNonText{TS1,T3,TS3}
3900 \let\org@TeX\TeX
3901 \let\org@LaTeX\LaTeX
3902 \let\ensureascii\@firstofone
3903 \AtBeginDocument{%
3904   \def\@elt#1{,#1,}%
3905   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3906   \let\@elt\relax
3907   \let\bbl@tempb\@empty
3908   \def\bbl@tempc{OT1}%
3909   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3910     \bbl@ifunset{T#1}{ }\def\bbl@tempb{#1}}}%
3911   \bbl@foreach\bbl@tempa{%
3912     \bbl@xin@{#1}{\BabelNonASCII}%
3913     \ifin@
3914       \def\bbl@tempb{#1}% Store last non-ascii
3915     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3916     \ifin@\else
3917       \def\bbl@tempc{#1}% Store last ascii
3918     \fi
3919     \fi}%
3920   \ifx\bbl@tempb\@empty\else
3921     \bbl@xin@,{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3922     \ifin@\else
3923       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3924     \fi
3925     \edef\ensureascii#1{%
3926       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3927     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3928     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3929   \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.

```

3930 \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.

```

3931 \AtBeginDocument{%
3932   \@ifpackageloaded{fontspec}%
3933   {\xdef\latinencoding{%
3934     \ifx\UTFencname\@undefined
3935       EU\ifcase\bbl@engine\or2\or1\fi
3936     \else
3937       \UTFencname
3938     \fi}}%
3939   {\gdef\latinencoding{OT1}%
3940     \ifx\cf@encoding\bbl@t@one
3941       \xdef\latinencoding{\bbl@t@one}%
3942     \else
3943       \def\@elt#1{,#1,}%

```

```

3944 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3945 \let\@elt\relax
3946 \bbl@xin@{,T1,}\bbl@tempa
3947 \ifin@
3948 \xdef\latinencoding{\bbl@t@one}%
3949 \fi
3950 \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.

```

3951 \DeclareRobustCommand{\latintext}{%
3952 \fontencoding{\latinencoding}\selectfont
3953 \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.

```

3954 \ifx\@undefined\DeclareTextFontCommand
3955 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3956 \else
3957 \DeclareTextFontCommand{\textlatin}{\latintext}
3958 \fi

```

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

```

3959 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

8.5 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.

```

3960 \bbl@trace{Loading basic (internal) bidi support}
3961 \ifodd\bbl@engine
3962 \else % TODO. Move to txtbabel
3963 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3964 \bbl@error
3965 {The bidi method 'basic' is available only in\\%
3966 luatex. I'll continue with 'bidi=default', so\\%
3967 expect wrong results}%
3968 {See the manual for further details.}%
3969 \let\bbl@beforeforeign\leavevmode
3970 \AtEndOfPackage{%
3971 \EnableBabelHook{babel-bidi}%
3972 \bbl@xebidipar}

```



```

3973 \fi\fi
3974 \def\bbl@loadxebidi#1{%
3975   \ifx\RTLfootnotetext\@undefined
3976     \AtEndOfPackage{%
3977       \EnableBabelHook{babel-bidi}%
3978       \ifx\fontspec\@undefined
3979         \bbl@loadfontspec % bidi needs fontspec
3980       \fi
3981       \usepackage#1{bidi}}%
3982   \fi}
3983 \ifnum\bbl@bidimode>200
3984   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3985     \bbl@tentative{bidi=bidi}
3986     \bbl@loadxebidi{}
3987   \or
3988     \bbl@loadxebidi{[rldocument]}
3989   \or
3990     \bbl@loadxebidi{}
3991   \fi
3992 \fi
3993 \fi
3994 % TODO? Separate:
3995 \ifnum\bbl@bidimode=\@ne
3996   \let\bbl@beforeforeign\leavevmode
3997   \ifodd\bbl@engine
3998     \newattribute\bbl@attr@dir
3999     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4000     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4001   \fi
4002   \AtEndOfPackage{%
4003     \EnableBabelHook{babel-bidi}%
4004     \ifodd\bbl@engine\else
4005       \bbl@xebidipar
4006     \fi}
4007 \fi

```

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

```

4008 \bbl@trace{Macros to switch the text direction}
4009 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4010 \def\bbl@rscripts{% TODO. Base on codes ??
4011   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4012   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
4013   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
4014   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4015   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4016   Old South Arabian,}%
4017 \def\bbl@provide@dirs#1{%
4018   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4019   \ifin@
4020     \global\bbl@csarg\chardef{wdir@#1}\@ne
4021     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4022     \ifin@
4023       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4024     \fi
4025   \else
4026     \global\bbl@csarg\chardef{wdir@#1}\z@
4027   \fi
4028   \ifodd\bbl@engine
4029     \bbl@csarg\ifcase{wdir@#1}%
4030     \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
4031   \or
4032     \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%

```

```

4033 \or
4034 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4035 \fi
4036 \fi}
4037 \def\bbl@switchdir{%
4038 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4039 \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4040 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4041 \def\bbl@setdirs#1{% TODO - math
4042 \ifcase\bbl@select@type % TODO - strictly, not the right test
4043 \bbl@bodydir{#1}%
4044 \bbl@pardir{#1}%
4045 \fi
4046 \bbl@texdir{#1}}
4047 % TODO. Only if \bbl@bidimode > 0?:
4048 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4049 \DisableBabelHook{babel-bidi}

```

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

```

4050 \ifodd\bbl@engine % luatex=1
4051 \else % pdftex=0, xetex=2
4052 \newcount\bbl@dirlevel
4053 \chardef\bbl@thetexdir\z@
4054 \chardef\bbl@thepardir\z@
4055 \def\bbl@texdir#1{%
4056 \ifcase#1\relax
4057 \chardef\bbl@thetexdir\z@
4058 \bbl@texdir@i\begin\endL
4059 \else
4060 \chardef\bbl@thetexdir@ne
4061 \bbl@texdir@i\beginR\endR
4062 \fi}
4063 \def\bbl@texdir@i#1#2{%
4064 \ifhmode
4065 \ifnum\currentgrouplevel>\z@
4066 \ifnum\currentgrouplevel=\bbl@dirlevel
4067 \bbl@error{Multiple bidi settings inside a group}%
4068 {I'll insert a new group, but expect wrong results.}%
4069 \bgroup\aftergroup#2\aftergroup\egroup
4070 \else
4071 \ifcase\currentgrouptype\or % 0 bottom
4072 \aftergroup#2% 1 simple {}
4073 \or
4074 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4075 \or
4076 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4077 \or\or\or % vbox vtop align
4078 \or
4079 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4080 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4081 \or
4082 \aftergroup#2% 14 \begingroup
4083 \else
4084 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4085 \fi
4086 \fi
4087 \bbl@dirlevel\currentgrouplevel
4088 \fi
4089 #1%
4090 \fi}
4091 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4092 \let\bbl@bodydir@gobble
4093 \let\bbl@pagedir@gobble

```

```
4094 \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).

```
4095 \def\bbl@xebidipar{%
4096   \let\bbl@xebidipar\relax
4097   \TeXeTstate\@ne
4098   \def\bbl@xeverypar{%
4099     \ifcase\bbl@thepardir
4100     \ifcase\bbl@thetextdir\else\beginR\fi
4101     \else
4102       {\setbox\z@\lastbox\beginR\box\z@}%
4103       \fi}%
4104   \let\bbl@severypar\everypar
4105   \newtoks\everypar
4106   \everypar=\bbl@severypar
4107   \bbl@severypar{\bbl@xeverypar\the\everypar}}
4108 \ifnum\bbl@bidimode>200
4109   \let\bbl@textdir@i\@gobbletwo
4110   \let\bbl@xebidipar\@empty
4111   \AddBabelHook{bidi}{foreign}{%
4112     \def\bbl@tempa{\def\BabelText####1}%
4113     \ifcase\bbl@thetextdir
4114       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4115     \else
4116       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4117     \fi}
4118   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4119 \fi
4120 \fi
```

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

```
4121 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4122 \AtBeginDocument{%
4123   \ifx\pdfstringdefDisableCommands\@undefined\else
4124     \ifx\pdfstringdefDisableCommands\relax\else
4125       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4126     \fi
4127   \fi}
```

8.6 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`.

```
4128 \bbl@trace{Local Language Configuration}
4129 \ifx\loadlocalcfg\@undefined
4130   \@ifpackagewith{babel}{noconfigs}%
4131   {\let\loadlocalcfg\@gobble}%
4132   {\def\loadlocalcfg#1{%
4133     \InputIfFileExists{#1.cfg}%
4134     {\typeout{*****^J%
4135               * Local config file #1.cfg used^^J%
4136               *}}%
4137     \@empty}}
4138 \fi
```

8.7 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).

```

4139 \bbl@trace{Language options}
4140 \let\bbl@afterlang\relax
4141 \let\BabelModifiers\relax
4142 \let\bbl@loaded\@empty
4143 \def\bbl@load@language#1{%
4144   \InputIfFileExists{#1.ldf}%
4145   {\edef\bbl@loaded{\CurrentOption
4146     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4147     \expandafter\let\expandafter\bbl@afterlang
4148       \csname\CurrentOption.ldf-h@k\endcsname
4149     \expandafter\let\expandafter\BabelModifiers
4150       \csname\bbl@mod@\CurrentOption\endcsname}%
4151   {\bbl@error{%
4152     Unknown option '\CurrentOption'. Either you misspelled it\\%
4153     or the language definition file \CurrentOption.ldf was not found}}%
4154     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4155     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4156     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.

```

4157 \def\bbl@try@load@lang#1#2#3{%
4158   \IfFileExists{\CurrentOption.ldf}%
4159   {\bbl@load@language{\CurrentOption}}%
4160   {#1\bbl@load@language{#2}#3}}
4161 %
4162 \DeclareOption{hebrew}{%
4163   \input{rlbabel.def}%
4164   \bbl@load@language{hebrew}}
4165 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4166 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4167 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4168 \DeclareOption{polutonikogreek}{%
4169   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4170 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4171 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4172 \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.

```

4173 \ifx\bbl@opt@config\@nnil
4174   \@ifpackagewith{babel}{noconfigs}{}%
4175   {\InputIfFileExists{bblopts.cfg}%
4176     {\typeout{*****^J%
4177       * Local config file bblopts.cfg used^^J%
4178       *}}%
4179     {}}%
4180 \else
4181   \InputIfFileExists{\bbl@opt@config.cfg}%
4182   {\typeout{*****^J%
4183     * Local config file \bbl@opt@config.cfg used^^J%
4184     *}}%
4185   {\bbl@error{%
4186     Local config file '\bbl@opt@config.cfg' not found}%
4187     Perhaps you misspelled it.}}%
4188 \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages. If not declared above, the names of the option

and the file are the same. We first pre-process the class and package options to determine the main language, which is processed in the third ‘main’ pass, *except* if all files are ldf *and* there is no main key. In the latter case (\bbl@opt@main is still \@nnil), the traditional way to set the main language is kept — the last loaded is the main language.

```

4189 \ifx\bbl@opt@main\@nnil
4190   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4191     \let\bbl@tempb\@empty
4192     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4193     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4194     \bbl@foreach\bbl@tempb{%      \bbl@tempb is a reversed list
4195       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4196         \ifodd\bbl@iniflag % = +=
4197           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4198         \else % n +=
4199           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4200         \fi
4201       \fi}%
4202   \fi
4203 \else
4204   \bbl@info{Main language set with 'main='. Except if you have\\%
4205     problems, prefer the default mechanism for setting\\%
4206     the main language. Reported}%
4207 \fi

```

A few languages are still defined explicitly. They are stored in case they are needed in the ‘main’ pass (the value can be \relax).

```

4208 \ifx\bbl@opt@main\@nnil\else
4209   \bbl@csarg\let{loadmain\expandafter}\csname ds@\bbl@opt@main\endcsname
4210   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4211 \fi

```

Now define the corresponding loaders. With package options, assume the language exists. With class options, check if the option is a language by checking if the correspondin file exists.

```

4212 \bbl@foreach\bbl@language@opts{%
4213   \def\bbl@tempa{#1}%
4214   \ifx\bbl@tempa\bbl@opt@main\else
4215     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4216       \bbl@ifunset{ds@#1}%
4217       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4218     {}%
4219   \else % + * (other = ini)
4220     \DeclareOption{#1}{%
4221       \bbl@ldfinit
4222       \babelprovide[import]{#1}%
4223       \bbl@afterldf{}}%
4224   \fi
4225 \fi}
4226 \bbl@foreach\@classoptionslist{%
4227   \def\bbl@tempa{#1}%
4228   \ifx\bbl@tempa\bbl@opt@main\else
4229     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4230       \bbl@ifunset{ds@#1}%
4231       {\IfFileExists{#1.ldf}%
4232        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4233        {}}%
4234     {}%
4235   \else % + * (other = ini)
4236     \IfFileExists{babel-#1.tex}%
4237     {\DeclareOption{#1}{%
4238       \bbl@ldfinit
4239       \babelprovide[import]{#1}%
4240       \bbl@afterldf{}}}%
4241   {}%

```

```

4242     \fi
4243 \fi}

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (but remember class options are processed before):

```

4244 \def\AfterBabelLanguage#1{%
4245   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4246 \DeclareOption*{}
4247 \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. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \babelprovide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

```

4248 \bbl@trace{Option 'main'}
4249 \ifx\bbl@opt@main\@nnil
4250   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4251   \let\bbl@tempc\@empty
4252   \bbl@for\bbl@tempb\bbl@tempa{%
4253     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4254     \ifin@ \edef\bbl@tempc{\bbl@tempb}\fi}
4255   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4256   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4257   \ifx\bbl@tempb\bbl@tempc\else
4258     \bbl@warning{%
4259       Last declared language option is '\bbl@tempc',\%
4260       but the last processed one was '\bbl@tempb'.\%
4261       The main language can't be set as both a global\%
4262       and a package option. Use 'main=\bbl@tempc' as\%
4263       option. Reported}
4264   \fi
4265 \else
4266   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4267     \bbl@ldfinit
4268     \let\CurrentOption\bbl@opt@main
4269     \bbl@exp{% \bbl@opt@provide = empty if *
4270       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4271     \bbl@afterldf{}
4272     \DeclareOption{\bbl@opt@main}{}
4273   \else % case 0,2 (main is ldf)
4274     \ifx\bbl@loadmain\relax
4275       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4276     \else
4277       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4278     \fi
4279     \ExecuteOptions{\bbl@opt@main}
4280     \@namedef{ds@\bbl@opt@main}{}%
4281   \fi
4282   \DeclareOption*{}
4283   \ProcessOptions*
4284 \fi
4285 \def\AfterBabelLanguage{%
4286   \bbl@error
4287   {Too late for \string\AfterBabelLanguage}%
4288   {Languages have been loaded, so I can do nothing}}
4289 \ifx\bbl@main@language\undefined
4290   \bbl@info{%

```

```

4291   You haven't specified a language. I'll use 'nil'\%
4292   as the main language. Reported}
4293   \bbl@load@language{nil}
4294 \fi
4295 \</package>

```

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

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

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

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

A proxy file for switch.def

```

4296 \<*kernel>
4297 \let\bbl@onlyswitch\@empty
4298 \input babel.def
4299 \let\bbl@onlyswitch\@undefined
4300 \</kernel>
4301 \<*patterns>

```

10 Loading hyphenation patterns

The following code is meant to be read by iniT_EX because it should instruct T_EX 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.

```

4302 \<<Make sure ProvidesFile is defined>>
4303 \ProvidesFile{hyphen.cfg}[\<<date>>] [\<<version>>] Babel hyphens]
4304 \xdef\bbl@format{\jobname}
4305 \def\bbl@version{\<<version>>}
4306 \def\bbl@date{\<<date>>}
4307 \ifx\AtBeginDocument\@undefined
4308   \def\@empty{}
4309 \fi
4310 \<<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.

```

4311 \def\process@line#1#2 #3 #4 {%
4312   \ifx=#1%
4313     \process@synonym{#2}%
4314   \else
4315     \process@language{#1#2}{#3}{#4}%
4316   \fi
4317   \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.

```

4318 \toks@{}
4319 \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.

```

4320 \def\process@synonym#1{%
4321   \ifnum\last@language=\m@ne
4322     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4323   \else
4324     \expandafter\chardef\csname l@#1\endcsname\last@language
4325     \wlog{\string\l@#1=\string\language\the\last@language}%
4326     \expandafter\let\csname #1hyphenmins\endcsname
4327       \csname\language\hyphenmins\endcsname
4328     \let\bbl@elt\relax
4329     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4330   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language.

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

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

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

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

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

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2

arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

4331 \def\process@language#1#2#3{%
4332   \expandafter\addlanguage\csname l@#1\endcsname
4333   \expandafter\language\csname l@#1\endcsname
4334   \edef\language{#1}%
4335   \bbl@hook@everylanguage{#1}%
4336   % > luatex
4337   \bbl@get@enc#1::\@@@
4338   \begingroup
4339     \lefthyphenmin\m@ne
4340     \bbl@hook@loadpatterns{#2}%
4341     % > luatex
4342     \ifnum\lefthyphenmin=\m@ne
4343     \else
4344       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4345         \the\lefthyphenmin\the\righthyphenmin}%
4346     \fi
4347   \endgroup
4348   \def\bbl@tempa{#3}%
4349   \ifx\bbl@tempa\@empty\else
4350     \bbl@hook@loadexceptions{#3}%
4351     % > luatex
4352   \fi
4353   \let\bbl@elt\relax

```



```

4354 \edef\bbl@languages{%
4355   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4356 \ifnum\the\language=\z@
4357   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4358     \set@hyphenmins\tw@\thr@\relax
4359   \else
4360     \expandafter\expandafter\expandafter\set@hyphenmins
4361       \csname #1hyphenmins\endcsname
4362   \fi
4363   \the\toks@
4364   \toks@{}}%
4365 \fi}

```

\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. It uses delimited arguments to achieve this.

```

4366 \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.

```

4367 \def\bbl@hook@everylanguage#1{%
4368   \def\bbl@hook@loadpatterns#1{\input #1\relax}
4369   \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4370   \def\bbl@hook@loadkernel#1{%
4371     \def\addlanguage{\csname newlanguage\endcsname}%
4372     \def\adddialect##1##2{%
4373       \global\chardef##1##2\relax
4374       \wlog{\string##1 = a dialect from \string\language##2}}%
4375     \def\iflanguage##1{%
4376       \expandafter\ifx\csname l@##1\endcsname\relax
4377         \@nolanerr{##1}%
4378       \else
4379         \ifnum\csname l@##1\endcsname=\language
4380           \expandafter\expandafter\expandafter\@firstoftwo
4381         \else
4382           \expandafter\expandafter\expandafter\@secondoftwo
4383         \fi
4384       \fi}%
4385     \def\providehyphenmins##1##2{%
4386       \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4387         \@namedef{##1hyphenmins}{##2}%
4388       \fi}%
4389     \def\set@hyphenmins##1##2{%
4390       \lefthyphenmin##1\relax
4391       \righthyphenmin##2\relax}%
4392     \def\selectlanguage{%
4393       \errhelp{Selecting a language requires a package supporting it}%
4394       \errmessage{Not loaded}}%
4395     \let\foreignlanguage\selectlanguage
4396     \let\otherlanguage\selectlanguage
4397     \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4398     \def\bbl@usehooks##1##2{% TODO. Temporary!!
4399     \def\setlocale{%
4400       \errhelp{Find an armchair, sit down and wait}%
4401       \errmessage{Not yet available}}%
4402     \let\uselocale\setlocale
4403     \let\locale\setlocale
4404     \let\selectlocale\setlocale
4405     \let\localename\setlocale
4406     \let\textlocale\setlocale
4407     \let\textlanguage\setlocale
4408     \let\languagetext\setlocale}
4409 \begingroup

```

```

4410 \def\AddBabelHook#1#2{%
4411 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4412 \def\next{\toks1}%
4413 \else
4414 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4415 \fi
4416 \next}
4417 \ifx\directlua\@undefined
4418 \ifx\XeTeXinputencoding\@undefined\else
4419 \input xebabel.def
4420 \fi
4421 \else
4422 \input luababel.def
4423 \fi
4424 \openin1 = babel-\bbl@format.cfg
4425 \ifeof1
4426 \else
4427 \input babel-\bbl@format.cfg\relax
4428 \fi
4429 \closein1
4430 \endgroup
4431 \bbl@hook@loadkernel{switch.def}

```

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

```

4432 \openin1 = language.dat

```

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

```

4433 \def\language{english}%
4434 \ifeof1
4435 \message{I couldn't find the file language.dat,\space
4436 I will try the file hyphen.tex}
4437 \input hyphen.tex\relax
4438 \chardef\l@english\z@
4439 \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.

```

4440 \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.

```

4441 \loop
4442 \endlinechar\m@ne
4443 \read1 to \bbl@line
4444 \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.

```

4445 \if T\ifeof1F\fi T\relax
4446 \ifx\bbl@line\@empty\else
4447 \edef\bbl@line{\bbl@line\space\space\space}%
4448 \expandafter\process@line\bbl@line\relax
4449 \fi
4450 \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.

```

4451 \begingroup
4452 \def\bbl@elt#1#2#3#4{%

```

```

4453 \global\language=#2\relax
4454 \gdef\language{#1}%
4455 \def\bbl@elt##1##2##3##4{}}%
4456 \bbl@languages
4457 \endgroup
4458 \fi
4459 \closein1

```

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

```

4460 \if/\the\toks@/\else
4461 \errhelp{language.dat loads no language, only synonyms}
4462 \errmessage{Orphan language synonym}
4463 \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.

```

4464 \let\bbl@line\@undefined
4465 \let\process@line\@undefined
4466 \let\process@synonym\@undefined
4467 \let\process@language\@undefined
4468 \let\bbl@get@enc\@undefined
4469 \let\bbl@hyph@enc\@undefined
4470 \let\bbl@tempa\@undefined
4471 \let\bbl@hook@loadkernel\@undefined
4472 \let\bbl@hook@everylanguage\@undefined
4473 \let\bbl@hook@loadpatterns\@undefined
4474 \let\bbl@hook@loadexceptions\@undefined
4475 </patterns>

```

Here the code for `iniTeX` ends.

11 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].

```

4476 <(*More package options)> ≡
4477 \chardef\bbl@bidimode\z@
4478 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4479 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4480 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4481 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4482 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4483 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4484 <(/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.

```

4485 <(*Font selection)> ≡
4486 \bbl@trace{Font handling with fontspec}
4487 \ifx\ExplSyntaxOn\@undefined\else
4488 \ExplSyntaxOn
4489 \catcode`\ =10
4490 \def\bbl@loadfontspec{%
4491 \usepackage{fontspec}% TODO. Apply patch always
4492 \expandafter
4493 \def\csname msg-text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4494 Font '\l_fontspec_fontname_tl' is using the\%
4495 default features for language '##1'.\\%

```

```

4496     That's usually fine, because many languages\\%
4497     require no specific features, but if the output is\\%
4498     not as expected, consider selecting another font.}
4499     \expandafter
4500     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%
4501         Font '\l_fontspec_fontname_tl' is using the\\%
4502         default features for script '##2'.\\%
4503         That's not always wrong, but if the output is\\%
4504         not as expected, consider selecting another font.}}
4505     \ExplSyntaxOff
4506 \fi
4507 \@onlypreamble\babelfont
4508 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4509     \bbl@foreach{#1}{%
4510         \expandafter\ifx\csname date##1\endcsname\relax
4511             \IfFileExists{babel-##1.tex}%
4512                 {\babelprovide{##1}}%
4513                 {}%
4514         \fi}%
4515     \edef\bbl@tempa{#1}%
4516     \def\bbl@tempb{#2}% Used by \bbl@bblfont
4517     \ifx\fontspec\undefined
4518         \bbl@loadfontspec
4519     \fi
4520     \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4521     \bbl@bblfont}
4522 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4523     \bbl@ifunset{\bbl@tempb family}%
4524         {\bbl@providefam{\bbl@tempb}}%
4525         {}%
4526     % For the default font, just in case:
4527     \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
4528     \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4529     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4530     \bbl@exp{%
4531         \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4532         \<\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4533         \<\bbl@tempb default>\<\bbl@tempb family>}}%
4534     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4535         \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:

```

4536 \def\bbl@providefam#1{%
4537     \bbl@exp{%
4538         \<\newcommand\<#1default>{}% Just define it
4539         \<\bbl@add@list\<\bbl@font@fams{#1}%
4540         \<\DeclareRobustCommand\<#1family>{}%
4541         \<\not@math@alphabet\<#1family>\relax
4542         % \<\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4543         \<\fontfamily\<#1default>%
4544         \<\ifx>\<\UseHooks\<\@undefined\<else>\<\UseHook{#1family}\<\fi>%
4545         \<\selectfont>%
4546         \<\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.

```

4547 \def\bbl@nostdfont#1{%
4548     \bbl@ifunset{\bbl@WFF@\f@family}%
4549     {\bbl@csarg\gdef{WFF@\f@family}}{}% Flag, to avoid dupl warns
4550     \bbl@infowarn{The current font is not a babel standard family:\\%
4551         #1%
4552         \fontname\font\\%
4553         There is nothing intrinsically wrong with this warning, and\\%
4554         you can ignore it altogether if you do not need these\\%

```

```

4555     families. But if they are used in the document, you should be\\%
4556     aware 'babel' will not set Script and Language for them, so\\%
4557     you may consider defining a new family with \string\babelfont.\\%
4558     See the manual for further details about \string\babelfont.\\%
4559     Reported}}
4560     {}}%
4561 \gdef\bbbl@switchfont{%
4562   \bbbl@ifunset{bbbl@lsys@\language}{\bbbl@provide@lsys{\language}}{}%
4563   \bbbl@exp{% eg Arabic -> arabic
4564     \lowercase{\edef\\bbbl@tempa{\bbbl@cl{sname}}}}%
4565   \bbbl@foreach\bbbl@font@fams{%
4566     \bbbl@ifunset{bbbl@##1dflt@\language}%      (1) language?
4567     {\bbbl@ifunset{bbbl@##1dflt@*\bbbl@tempa}%  (2) from script?
4568       {\bbbl@ifunset{bbbl@##1dflt@}%            2=F - (3) from generic?
4569         {}%                                     123=F - nothing!
4570         {\bbbl@exp{%                            3=T - from generic
4571           \global\let\<bbbl@##1dflt@\language>%
4572             \<bbbl@##1dflt@>}}}%
4573       {\bbbl@exp{%                               2=T - from script
4574         \global\let\<bbbl@##1dflt@\language>%
4575           \<bbbl@##1dflt@*\bbbl@tempa>}}}%
4576     {}}%                                         1=T - language, already defined
4577   \def\bbbl@tempa{\bbbl@nostdfont{}}%
4578   \bbbl@foreach\bbbl@font@fams{% don't gather with prev for
4579     \bbbl@ifunset{bbbl@##1dflt@\language}%
4580     {\bbbl@cs{famrst@##1}%
4581       \global\bbbl@csarg\let{famrst@##1}\relax}%
4582     {\bbbl@exp{% order is relevant. TODO: but sometimes wrong!
4583       \\bbbl@add\\originalTeX%
4584       \\bbbl@font@rst{\bbbl@cl{##1dflt}}%
4585       \<##1default>\<##1family>{##1}}%
4586       \\bbbl@font@set\<bbbl@##1dflt@\language>% the main part!
4587       \<##1default>\<##1family>}}}%
4588   \bbbl@ifrestoring{ }\bbbl@tempa}}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

4589 \ifx\fb@family\undefined\else % if latex
4590   \ifcase\bbbl@engine % if pdftex
4591     \let\bbbl@ckeckstdfonts\relax
4592   \else
4593     \def\bbbl@ckeckstdfonts{%
4594       \begingroup
4595       \global\let\bbbl@ckeckstdfonts\relax
4596       \let\bbbl@tempa\@empty
4597       \bbbl@foreach\bbbl@font@fams{%
4598         \bbbl@ifunset{bbbl@##1dflt@}%
4599         {\@nameuse{##1family}%
4600          \bbbl@csarg\gdef{WFF@\fb@family}}{}% Flag
4601          \bbbl@exp{\\bbbl@add\\bbbl@tempa{* \<##1family>= \fb@family\\%
4602            \space\space\fontname\font\\%
4603            \bbbl@csarg\xdef{##1dflt@}{\fb@family}%
4604            \expandafter\xdef\csname ##1default\endcsname{\fb@family}}%
4605          {}}%
4606       \ifx\bbbl@tempa\@empty\else
4607         \bbbl@infowarn{The following font families will use the default\\%
4608           settings for all or some languages:\\%
4609           \bbbl@tempa
4610           There is nothing intrinsically wrong with it, but\\%
4611           'babel' will no set Script and Language, which could\\%
4612           be relevant in some languages. If your document uses\\%
4613           these families, consider redefining them with \string\babelfont.\\%
4614           Reported}%

```

```

4615         \fi
4616     \endgroup}
4617 \fi
4618 \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.

4619 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4620     \bbl@xin@{<>}{#1}%
4621     \ifin@
4622         \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4623     \fi
4624     \bbl@exp{%
4625         \def\#2{#1}%          'Unprotected' macros return prev values
4626         \def\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4627         \bbl@ifsamestring{#2}{\f@family}%
4628         {\#3%
4629             \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}%
4630         \let\bbl@tempa\relax}%
4631     {}}
4632 %      TODO - next should be global?, but even local does its job. I'm
4633 %      still not sure -- must investigate:
4634 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4635     \let\bbl@tempe\bbl@mapselect
4636     \let\bbl@mapselect\relax
4637     \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4638     \let#4@empty %          Make sure \renewfontfamily is valid
4639     \bbl@exp{%
4640         \let\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4641         \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4642         {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4643         \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4644         {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4645         \renewfontfamily\#4%
4646         [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{.}{#3}
4647     }
4648     \begingroup
4649         #4%
4650         \xdef#1{\f@family}%    eg, \bbl@rmdflt@lang{FreeSerif(0)}
4651     \endgroup
4652     \let#4\bbl@temp@fam
4653     \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4654     \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.

```

4653 \def\bbl@font@rst#1#2#3#4{%
4654     \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}%

```

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

```

4655 \def\bbl@font@fams{rm,sf,tt}
4656 <</Font selection>>

```

12 Hooks for XeTeX and LuaTeX

12.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

```

4657 <<{*Footnote changes}>> ≡
4658 \bbl@trace{Bidi footnotes}
4659 \ifnum\bbl@bidimode>\z@

```

```

4660 \def\bbl@footnote#1#2#3{%
4661 \ifnextchar[%
4662 {\bbl@footnote@o{#1}{#2}{#3}}%
4663 {\bbl@footnote@x{#1}{#2}{#3}}}
4664 \long\def\bbl@footnote@x#1#2#3#4{%
4665 \bgroup
4666 \select@language@x{\bbl@main@language}%
4667 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4668 \egroup}
4669 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4670 \bgroup
4671 \select@language@x{\bbl@main@language}%
4672 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4673 \egroup}
4674 \def\bbl@footnotetext#1#2#3{%
4675 \ifnextchar[%
4676 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4677 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4678 \long\def\bbl@footnotetext@x#1#2#3#4{%
4679 \bgroup
4680 \select@language@x{\bbl@main@language}%
4681 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4682 \egroup}
4683 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4684 \bgroup
4685 \select@language@x{\bbl@main@language}%
4686 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4687 \egroup}
4688 \def\BabelFootnote#1#2#3#4{%
4689 \ifx\bbl@fn@footnote\undefined
4690 \let\bbl@fn@footnote\footnote
4691 \fi
4692 \ifx\bbl@fn@footnotetext\undefined
4693 \let\bbl@fn@footnotetext\footnotetext
4694 \fi
4695 \bbl@ifblank{#2}%
4696 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4697 \@namedef{\bbl@stripslash#1text}%
4698 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4699 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4700 \@namedef{\bbl@stripslash#1text}%
4701 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4702 \fi
4703 <</Footnote changes>>

```

Now, the code.

```

4704 <*xetex>
4705 \def\BabelStringsDefault{unicode}
4706 \let\xebbl@stop\relax
4707 \AddBabelHook{xetex}{encodedcommands}{%
4708 \def\bbl@tempa{#1}%
4709 \ifx\bbl@tempa\empty
4710 \XeTeXinputencoding"bytes"%
4711 \else
4712 \XeTeXinputencoding"#1"%
4713 \fi
4714 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4715 \AddBabelHook{xetex}{stopcommands}{%
4716 \xebbl@stop
4717 \let\xebbl@stop\relax}
4718 \def\bbl@intraspace#1 #2 #3\@{%
4719 \bbl@csarg\gdef{xisp@{language}}%
4720 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}

```

```

4721 \def\bbl@intrapenalty#1\@@{%
4722   \bbl@csarg\gdef{xeipn@\language}%
4723   {\XeTeXlinebreakpenalty #1\relax}}
4724 \def\bbl@provide@intraspace{%
4725   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4726   \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4727   \ifin@
4728     \bbl@ifunset{bbl@intsp@\language}{}%
4729     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4730       \ifx\bbl@KVP@intraspace\@nnil
4731         \bbl@exp{%
4732           \\bbl@intraspace\bbl@cl{intsp}\\@@}%
4733         \fi
4734         \ifx\bbl@KVP@intrapenalty\@nnil
4735           \bbl@intrapenalty0\@@
4736         \fi
4737       \fi
4738       \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4739         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4740       \fi
4741       \ifx\bbl@KVP@intrapenalty\@nnil\else
4742         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4743       \fi
4744       \bbl@exp{%
4745         % TODO. Execute only once (but redundant):
4746         \\bbl@add\<extras\language>{%
4747           \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4748           \<bbl@xeisp@\language>%
4749           \<bbl@xeipn@\language>}%
4750         \\bbl@tglobal\<extras\language>%
4751         \\bbl@add\<noextras\language>{%
4752           \XeTeXlinebreaklocale "en"%
4753           \\bbl@tglobal\<noextras\language>}%
4754         \ifx\bbl@ispacesize\@undefined
4755           \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4756         \ifx\AtBeginDocument\@notprerr
4757           \expandafter\@secondoftwo % to execute right now
4758         \fi
4759         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4760       \fi}%
4761   \fi}
4762 \ifx\DisableBabelHook\@undefined\endinput\fi
4763 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4764 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4765 \DisableBabelHook{babel-fontspec}
4766 <(Font selection)>
4767 \input txtbabel.def
4768 </xetex>

```

12.2 Layout

In progress.

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

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

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

```

4769 <*texxt>
4770 \providecommand\bbl@provide@intraspace{}
4771 \bbl@trace{Redefinitions for bidi layout}
4772 \def\bbl@sspre@caption{%
4773   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}

```



```

4774 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4775 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4776 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4777 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4778   \def\@hangfrom#1{%
4779     \setbox\@tempboxa\hbox{#1}}%
4780   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4781   \noindent\box\@tempboxa}
4782 \def\raggedright{%
4783   \let\@centercr
4784   \bbl@startskip\z@skip
4785   \@rightskip\@flushglue
4786   \bbl@endskip\@rightskip
4787   \parindent\z@
4788   \parfillskip\bbl@startskip}
4789 \def\raggedleft{%
4790   \let\@centercr
4791   \bbl@startskip\@flushglue
4792   \bbl@endskip\z@skip
4793   \parindent\z@
4794   \parfillskip\bbl@endskip}
4795 \fi
4796 \IfBabelLayout{lists}
4797   {\bbl@sreplace\list
4798     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4799     \def\bbl@listleftmargin{%
4800       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4801     \ifcase\bbl@engine
4802       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4803       \def\p@enumiii{\p@enumii}\theenumii{}\fi
4804       \bbl@sreplace\verbatim
4805       {\leftskip\@totalleftmargin}%
4806       {\bbl@startskip\textwidth
4807         \advance\bbl@startskip-\linewidth}%
4808       \bbl@sreplace\verbatim
4809       {\rightskip\z@skip}%
4810       {\bbl@endskip\z@skip}}%
4811   {}
4812 {}
4813 \IfBabelLayout{contents}
4814   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4815     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4816 {}
4817 \IfBabelLayout{columns}
4818   {\bbl@sreplace\outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4819     \def\bbl@outputbox#1{%
4820       \hb@xt@\textwidth{%
4821         \hskip\columnwidth
4822         \hfil
4823         {\normalcolor\vrule \@width\columnseprule}%
4824         \hfil
4825         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4826         \hskip-\textwidth
4827         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4828         \hskip\columnsep
4829         \hskip\columnwidth}}}%
4830   {}
4831 \langle Footnote changes \rangle
4832 \IfBabelLayout{footnotes}%
4833   {\BabelFootnote\footnote\languagename{}\{}}%
4834   {\BabelFootnote\localfootnote\languagename{}\{}}%
4835   {\BabelFootnote\mainfootnote{}\{}}%
4836 {}

```

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.

```

4837 \IfBabelLayout{counters}%
4838   {\let\bbl@latinarabic=\@arabic
4839    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4840   \let\bbl@asciroman=\@roman
4841   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4842   \let\bbl@asciiRoman=\@Roman
4843   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
4844 \end{texet}

```

12.3 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).

```

4845 \begin{luatex}
4846 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4847 \bbl@trace{Read language.dat}
4848 \ifx\bbl@readstream\@undefined
4849   \csname newread\endcsname\bbl@readstream
4850 \fi
4851 \begin{group}
4852   \toks@{}
4853   \count@ \z@ % 0=start, 1=0th, 2=normal
4854   \def\bbl@process@line#1#2 #3 #4 {%
4855     \ifx=#1%
4856       \bbl@process@synonym{#2}%
4857     \else
4858       \bbl@process@language{#1#2}{#3}{#4}%
4859     \fi
4860     \ignorespaces}
4861   \def\bbl@manylang{%

```

```

4862 \ifnum\bbl@last>\@ne
4863 \bbl@info{Non-standard hyphenation setup}%
4864 \fi
4865 \let\bbl@manylang\relax}
4866 \def\bbl@process@language#1#2#3{%
4867 \ifcase\count@
4868 \ifundefined{zth@#1}{\count@tw@}{\count@ne}%
4869 \or
4870 \count@tw@
4871 \fi
4872 \ifnum\count@=tw@
4873 \expandafter\addlanguage\csname l@#1\endcsname
4874 \language\allocationnumber
4875 \chardef\bbl@last\allocationnumber
4876 \bbl@manylang
4877 \let\bbl@elt\relax
4878 \xdef\bbl@languages{%
4879 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4880 \fi
4881 \the\toks@
4882 \toks@{}}
4883 \def\bbl@process@synonym@aux#1#2{%
4884 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4885 \let\bbl@elt\relax
4886 \xdef\bbl@languages{%
4887 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4888 \def\bbl@process@synonym#1{%
4889 \ifcase\count@
4890 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4891 \or
4892 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4893 \else
4894 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4895 \fi}
4896 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4897 \chardef\l@english\z@
4898 \chardef\l@USenglish\z@
4899 \chardef\bbl@last\z@
4900 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4901 \gdef\bbl@languages{%
4902 \bbl@elt{english}{0}{hyphen.tex}}%
4903 \bbl@elt{USenglish}{0}{}}
4904 \else
4905 \global\let\bbl@languages@format\bbl@languages
4906 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4907 \ifnum#2>\z@\else
4908 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4909 \fi}%
4910 \xdef\bbl@languages{\bbl@languages}%
4911 \fi
4912 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4913 \bbl@languages
4914 \openin\bbl@readstream=language.dat
4915 \ifeof\bbl@readstream
4916 \bbl@warning{I couldn't find language.dat. No additional\%
4917 patterns loaded. Reported}%
4918 \else
4919 \loop
4920 \endlinechar\m@ne
4921 \read\bbl@readstream to \bbl@line
4922 \endlinechar\^^M
4923 \if T\ifeof\bbl@readstream F\fi T\relax
4924 \ifx\bbl@line\empty\else

```

```

4925         \edef\bbl@line{\bbl@line\space\space\space}%
4926         \expandafter\bbl@process@line\bbl@line\relax
4927     \fi
4928     \repeat
4929 \fi
4930 \endgroup
4931 \bbl@trace{Macros for reading patterns files}
4932 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4933 \ifx\babelcatcodetablenum\undefined
4934     \ifx\newcatcodetable\undefined
4935         \def\babelcatcodetablenum{5211}
4936         \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4937     \else
4938         \newcatcodetable\babelcatcodetablenum
4939         \newcatcodetable\bbl@pattcodes
4940     \fi
4941 \else
4942     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4943 \fi
4944 \def\bbl@luapatterns#1#2{%
4945     \bbl@get@enc#1::\@@@
4946     \setbox\z@\hbox\bgroup
4947     \begingroup
4948         \savecatcodetable\babelcatcodetablenum\relax
4949         \initcatcodetable\bbl@pattcodes\relax
4950         \catcodetable\bbl@pattcodes\relax
4951         \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4952         \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\-=13
4953         \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4954         \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4955         \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4956         \catcode\`=12 \catcode\'=12 \catcode\"=12
4957         \input #1\relax
4958         \catcodetable\babelcatcodetablenum\relax
4959     \endgroup
4960     \def\bbl@tempa{#2}%
4961     \ifx\bbl@tempa\empty\else
4962         \input #2\relax
4963     \fi
4964     \egroup}%
4965 \def\bbl@patterns@lua#1{%
4966     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4967         \csname l@#1\endcsname
4968         \edef\bbl@tempa{#1}%
4969     \else
4970         \csname l@#1:\f@encoding\endcsname
4971         \edef\bbl@tempa{#1:\f@encoding}%
4972     \fi\relax
4973     \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4974     \@ifundefined{bbl@hyphendata@the\language}%
4975     {\def\bbl@elt##1##2##3##4{%
4976         \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4977         \def\bbl@tempb{##3}%
4978         \ifx\bbl@tempb\empty\else % if not a synonymous
4979             \def\bbl@tempc{##3}{##4}%
4980         \fi
4981         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4982     \fi}%
4983     \bbl@languages
4984     \@ifundefined{bbl@hyphendata@the\language}%
4985     {\bbl@info{No hyphenation patterns were set for\\
4986         language '\bbl@tempa'. Reported}}%
4987     {\expandafter\expandafter\expandafter\bbl@luapatterns

```

```

4988         \csname bbl@hyphendata@the\language\endcsname}}{}
4989 \endinput\fi
4990 % Here ends \ifx\AddBabelHook\undefined
4991 % A few lines are only read by hyphen.cfg
4992 \ifx\DisableBabelHook\undefined
4993   \AddBabelHook{luatex}{everylanguage}{%
4994     \def\process@language##1##2##3{%
4995       \def\process@line####1####2 ####3 ####4 {}}}
4996   \AddBabelHook{luatex}{loadpatterns}{%
4997     \input #1\relax
4998     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4999       {{#1}}}}
5000   \AddBabelHook{luatex}{loadexceptions}{%
5001     \input #1\relax
5002     \def\bbl@tempb##1##2{{##1}{#1}}%
5003     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5004       {\expandafter\expandafter\expandafter\bbl@tempb
5005         \csname bbl@hyphendata@the\language\endcsname}}
5006 \endinput\fi
5007 % Here stops reading code for hyphen.cfg
5008 % The following is read the 2nd time it's loaded
5009 \begingroup % TODO - to a lua file
5010 \catcode`\%=12
5011 \catcode`\'=12
5012 \catcode`\|=12
5013 \catcode`\:=12
5014 \directlua{
5015   Babel = Babel or {}
5016   function Babel.bytes(line)
5017     return line:gsub("(.)",
5018       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5019   end
5020   function Babel.begin_process_input()
5021     if luatexbase and luatexbase.add_to_callback then
5022       luatexbase.add_to_callback('process_input_buffer',
5023         Babel.bytes, 'Babel.bytes')
5024     else
5025       Babel.callback = callback.find('process_input_buffer')
5026       callback.register('process_input_buffer', Babel.bytes)
5027     end
5028   end
5029   function Babel.end_process_input ()
5030     if luatexbase and luatexbase.remove_from_callback then
5031       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5032     else
5033       callback.register('process_input_buffer', Babel.callback)
5034     end
5035   end
5036   function Babel.addpatterns(pp, lg)
5037     local lg = lang.new(lg)
5038     local pats = lang.patterns(lg) or ''
5039     lang.clear_patterns(lg)
5040     for p in pp:gmatch('[^%s]+') do
5041       ss = ''
5042       for i in string.utfcharacters(p:gsub('%d', '')) do
5043         ss = ss .. '%d?' .. i
5044       end
5045       ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5046       ss = ss:gsub('%.%d%?$', '%%.')
5047       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5048       if n == 0 then
5049         tex.sprint(
5050           [[\string\csname\space bbl@info\endcsname{New pattern: }]]

```

```

5051         .. p .. [[]])
5052         pats = pats .. ' ' .. p
5053     else
5054         tex.sprint(
5055             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5056             .. p .. [[]])
5057     end
5058 end
5059 lang.patterns(lg, pats)
5060 end
5061 Babel.characters = Babel.characters or {}
5062 Babel.ranges = Babel.ranges or {}
5063 function Babel.hlist_has_bidi(head)
5064     local has_bidi = false
5065     local ranges = Babel.ranges
5066     for item in node.traverse(head) do
5067         if item.id == node.id'glyph' then
5068             local itemchar = item.char
5069             local chardata = Babel.characters[itemchar]
5070             local dir = chardata and chardata.d or nil
5071             if not dir then
5072                 for nn, et in ipairs(ranges) do
5073                     if itemchar < et[1] then
5074                         break
5075                     elseif itemchar <= et[2] then
5076                         dir = et[3]
5077                         break
5078                     end
5079                 end
5080             end
5081             if dir and (dir == 'al' or dir == 'r') then
5082                 has_bidi = true
5083             end
5084         end
5085     end
5086     return has_bidi
5087 end
5088 function Babel.set_chranges_b (script, chrng)
5089     if chrng == '' then return end
5090     texio.write('Replacing ' .. script .. ' script ranges')
5091     Babel.script_blocks[script] = {}
5092     for s, e in string.gmatch(chrng..' ', '(-)%.%(-)%s') do
5093         table.insert(
5094             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5095     end
5096 end
5097 }
5098 \endgroup
5099 \ifx\newattribute\@undefined\else
5100     \newattribute\bbl@attr@locale
5101     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5102     \AddBabelHook{luatex}{beforeextras}{%
5103         \setattribute\bbl@attr@locale\localeid}
5104 \fi
5105 \def\BabelStringsDefault{unicode}
5106 \let\luabb1@stop\relax
5107 \AddBabelHook{luatex}{encodedcommands}{%
5108     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5109     \ifx\bbl@tempa\bbl@tempb\else
5110         \directlua{Babel.begin_process_input()}%
5111         \def\luabb1@stop{%
5112             \directlua{Babel.end_process_input()}}%
5113     \fi}%

```

```

5114 \AddBabelHook{luatex}{stopcommands}{%
5115   \luabbl@stop
5116   \let\luabbl@stop\relax}
5117 \AddBabelHook{luatex}{patterns}{%
5118   \@ifundefined{bbl@hyphendata@the\language}%
5119   {\def\bbl@elt##1##2##3##4{%
5120     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5121     \def\bbl@tempb{##3}%
5122     \ifx\bbl@tempb\@empty\else % if not a synonymous
5123       \def\bbl@tempc{##3}{##4}}%
5124     \fi
5125     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5126     \fi}%
5127   \bbl@languages
5128   \@ifundefined{bbl@hyphendata@the\language}%
5129   {\bbl@info{No hyphenation patterns were set for\%
5130     language '#2'. Reported}}%
5131   {\expandafter\expandafter\expandafter\bbl@luapatterns
5132     \csname bbl@hyphendata@the\language\endcsname}}}%
5133   \@ifundefined{bbl@patterns@}{}%
5134   \begingroup
5135     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5136     \ifin@ \else
5137       \ifx\bbl@patterns@\@empty\else
5138         \directlua{ Babel.addpatterns(
5139           [[\bbl@patterns@]], \number\language) }%
5140         \fi
5141         \@ifundefined{bbl@patterns@#1}%
5142         \@empty
5143         {\directlua{ Babel.addpatterns(
5144           [[\space\csname bbl@patterns@#1\endcsname]],
5145           \number\language) }}%
5146         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5147         \fi
5148     \endgroup}%
5149   \bbl@exp{%
5150     \bbl@ifunset{bbl@prehc@languagename}{}%
5151     {\bbl@ifblank{\bbl@cs{prehc@languagename}}{}}%
5152     {\prehyphenchar=\bbl@c{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.

```

5153 \@onlypreamble\babelpatterns
5154 \AtEndOfPackage{%
5155   \newcommand\babelpatterns[2][\@empty]{%
5156     \ifx\bbl@patterns@\relax
5157       \let\bbl@patterns@\@empty
5158     \fi
5159     \ifx\bbl@pttnlist@\@empty\else
5160       \bbl@warning{%
5161         You must not intermingle \string\selectlanguage\space and\%
5162         \string\babelpatterns\space or some patterns will not\%
5163         be taken into account. Reported}%
5164       \fi
5165       \ifx\@empty#1%
5166         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5167       \else
5168         \edef\bbl@tempb{\zap@space#1 \@empty}%
5169         \bbl@for\bbl@tempa\bbl@tempb{%
5170           \bbl@fixname\bbl@tempa
5171           \bbl@iflanguage\bbl@tempa{%
5172             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%

```

```

5173      \@ifundefined{bbl@patterns@bbl@tempa}%
5174      \@empty
5175      {\csname bbl@patterns@bbl@tempa\endcsname\space}%
5176      #2}}}%
5177      \fi}}

```

12.4 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.

```

5178 % TODO - to a lua file
5179 \directlua{
5180   Babel = Babel or {}
5181   Babel.linebreaking = Babel.linebreaking or {}
5182   Babel.linebreaking.before = {}
5183   Babel.linebreaking.after = {}
5184   Babel.locale = {} % Free to use, indexed by \localeid
5185   function Babel.linebreaking.add_before(func)
5186     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5187     table.insert(Babel.linebreaking.before, func)
5188   end
5189   function Babel.linebreaking.add_after(func)
5190     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5191     table.insert(Babel.linebreaking.after, func)
5192   end
5193 }
5194 \def\bbl@intraspace#1 #2 #3\@{%
5195   \directlua{
5196     Babel = Babel or {}
5197     Babel.intraspaces = Babel.intraspaces or {}
5198     Babel.intraspaces['\csname bbl@sbc@languagename\endcsname'] = %
5199       {b = #1, p = #2, m = #3}
5200     Babel.locale_props[\the\localeid].intraspace = %
5201       {b = #1, p = #2, m = #3}
5202   }}
5203 \def\bbl@intrapenalty#1\@{%
5204   \directlua{
5205     Babel = Babel or {}
5206     Babel.intrapenalties = Babel.intrapenalties or {}
5207     Babel.intrapenalties['\csname bbl@sbc@languagename\endcsname'] = #1
5208     Babel.locale_props[\the\localeid].intrapenalty = #1
5209   }}
5210 \begingroup
5211 \catcode`\%=12
5212 \catcode`\^=14
5213 \catcode`\'=12
5214 \catcode`\~=12
5215 \gdef\bbl@seaintraspace{^
5216   \let\bbl@seaintraspace\relax
5217   \directlua{
5218     Babel = Babel or {}
5219     Babel.sea_enabled = true
5220     Babel.sea_ranges = Babel.sea_ranges or {}
5221     function Babel.set_chrngs (script, chrng)
5222       local c = 0
5223       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5224         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5225         c = c + 1
5226       end
5227     end
5228     function Babel.sea_disc_to_space (head)

```



```

5229     local sea_ranges = Babel.sea_ranges
5230     local last_char = nil
5231     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5232     for item in node.traverse(head) do
5233         local i = item.id
5234         if i == node.id'glyph' then
5235             last_char = item
5236         elseif i == 7 and item.subtype == 3 and last_char
5237             and last_char.char > 0x0C99 then
5238             quad = font.getfont(last_char.font).size
5239             for lg, rg in pairs(sea_ranges) do
5240                 if last_char.char > rg[1] and last_char.char < rg[2] then
5241                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5242                     local intraspace = Babel.intraspaces[lg]
5243                     local intrapenalty = Babel.intrapenalties[lg]
5244                     local n
5245                     if intrapenalty ~= 0 then
5246                         n = node.new(14, 0)      ^% penalty
5247                         n.penalty = intrapenalty
5248                         node.insert_before(head, item, n)
5249                     end
5250                     n = node.new(12, 13)      ^% (glue, spaceskip)
5251                     node.setglue(n, intraspace.b * quad,
5252                                   intraspace.p * quad,
5253                                   intraspace.m * quad)
5254                     node.insert_before(head, item, n)
5255                     node.remove(head, item)
5256                 end
5257             end
5258         end
5259     end
5260 end
5261 }^^
5262 \bbl@luahyphenate}

```

12.5 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.

```

5263 \catcode`\%=14
5264 \gdef\bbl@cjkintraspacespace{%
5265   \let\bbl@cjkintraspacespace\relax
5266   \directlua{
5267     Babel = Babel or {}
5268     require('babel-data-cjk.lua')
5269     Babel.cjk_enabled = true
5270     function Babel.cjk_linebreak(head)
5271         local GLYPH = node.id'glyph'
5272         local last_char = nil
5273         local quad = 655360      % 10 pt = 655360 = 10 * 65536
5274         local last_class = nil
5275         local last_lang = nil
5276
5277         for item in node.traverse(head) do
5278             if item.id == GLYPH then
5279
5280                 local lang = item.lang
5281
5282                 local LOCALE = node.get_attribute(item,

```

```

5283         Babel.attr_locale)
5284     local props = Babel.locale_props[LOCALE]
5285
5286     local class = Babel.cjk_class[item.char].c
5287
5288     if props.cjk_quotes and props.cjk_quotes[item.char] then
5289         class = props.cjk_quotes[item.char]
5290     end
5291
5292     if class == 'cp' then class = 'cl' end % ]] as CL
5293     if class == 'id' then class = 'I' end
5294
5295     local br = 0
5296     if class and last_class and Babel.cjk_breaks[last_class][class] then
5297         br = Babel.cjk_breaks[last_class][class]
5298     end
5299
5300     if br == 1 and props.linebreak == 'c' and
5301         lang ~= \the\l@nohyphenation\space and
5302         last_lang ~= \the\l@nohyphenation then
5303         local intrapenalty = props.intrapenalty
5304         if intrapenalty ~= 0 then
5305             local n = node.new(14, 0) % penalty
5306             n.penalty = intrapenalty
5307             node.insert_before(head, item, n)
5308         end
5309         local intraspace = props.intraspace
5310         local n = node.new(12, 13) % (glue, spaceskip)
5311         node.setglue(n, intraspace.b * quad,
5312             intraspace.p * quad,
5313             intraspace.m * quad)
5314         node.insert_before(head, item, n)
5315     end
5316
5317     if font.getfont(item.font) then
5318         quad = font.getfont(item.font).size
5319     end
5320     last_class = class
5321     last_lang = lang
5322     else % if penalty, glue or anything else
5323         last_class = nil
5324     end
5325 end
5326 lang.hyphenate(head)
5327 end
5328 }%
5329 \bbl@luahyphenate}
5330 \gdef\bbl@luahyphenate{%
5331 \let\bbl@luahyphenate\relax
5332 \directlua{
5333     luatexbase.add_to_callback('hyphenate',
5334         function (head, tail)
5335             if Babel.linebreaking.before then
5336                 for k, func in ipairs(Babel.linebreaking.before) do
5337                     func(head)
5338                 end
5339             end
5340             if Babel.cjk_enabled then
5341                 Babel.cjk_linebreak(head)
5342             end
5343             lang.hyphenate(head)
5344             if Babel.linebreaking.after then
5345                 for k, func in ipairs(Babel.linebreaking.after) do

```

```

5346         func(head)
5347     end
5348 end
5349 if Babel.sea_enabled then
5350     Babel.sea_disc_to_space(head)
5351 end
5352 end,
5353 'Babel.hyphenate')
5354 }
5355 }
5356 \endgroup
5357 \def\bbl@provide@intraspace{%
5358     \bbl@ifunset{bbl@intsp@language\language}{}%
5359     {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5360         \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5361         \ifin@           % cjk
5362         \bbl@cjk@intraspace
5363         \directlua{
5364             Babel = Babel or {}
5365             Babel.locale_props = Babel.locale_props or {}
5366             Babel.locale_props[\the\localeid].linebreak = 'c'
5367         }%
5368         \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5369         \ifx\bbl@KVP@intrapenalty\@nnil
5370             \bbl@intrapenalty0@@
5371         \fi
5372     \else           % sea
5373         \bbl@sea@intraspace
5374         \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5375         \directlua{
5376             Babel = Babel or {}
5377             Babel.sea_ranges = Babel.sea_ranges or {}
5378             Babel.set_chranges('\bbl@cl{sbcpr}',
5379                               '\bbl@cl{chrng}')
5380         }%
5381         \ifx\bbl@KVP@intrapenalty\@nnil
5382             \bbl@intrapenalty0@@
5383         \fi
5384     \fi
5385 \fi
5386 \ifx\bbl@KVP@intrapenalty\@nnil\else
5387     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
5388 \fi}}

```

12.6 Arabic justification

```

5389 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5390 \def\bblar@chars{%
5391     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5392     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5393     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5394 \def\bblar@elongated{%
5395     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5396     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5397     0649,064A}
5398 \begin{group}
5399     \catcode`\_ =11 \catcode`\:=11
5400     \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5401 \end{group}
5402 \gdef\bbl@arabicjust{%
5403     \let\bbl@arabicjust\relax
5404     \newattribute\bblar@kashida
5405     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%

```

```

5406 \bblar@kashida=\z@
5407 \bbl@patchfont{\bbl@parsejalt}}%
5408 \directlua{
5409   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5410   Babel.arabic.elong_map[\the\localeid] = {}
5411   luatexbase.add_to_callback('post_linebreak_filter',
5412     Babel.arabic.justify, 'Babel.arabic.justify')
5413   luatexbase.add_to_callback('hpack_filter',
5414     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5415 }%
5416 % Save both node lists to make replacement. TODO. Save also widths to
5417 % make computations
5418 \def\bblar@fetchjalt#1#2#3#4{%
5419   \bbl@exp{\bbl@foreach{#1}}{%
5420     \bbl@ifunset\bblar@JE@##1}%
5421     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5422     {\setbox\z@\hbox{^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5423   \directlua{%
5424     local last = nil
5425     for item in node.traverse(tex.box[0].head) do
5426       if item.id == node.id'glyph' and item.char > 0x600 and
5427         not (item.char == 0x200D) then
5428         last = item
5429       end
5430     end
5431     Babel.arabic.#3['##1#4'] = last.char
5432   }}
5433 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5434 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5435 % positioning?
5436 \gdef\bbl@parsejalt{%
5437   \ifx\addfontfeature\@undefined\else
5438     \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
5439     \ifin@
5440       \directlua{%
5441         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5442           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5443           tex.print([\string\csname\space\bbl@parsejalti\endcsname])
5444         end
5445       }%
5446     \fi
5447   \fi}
5448 \gdef\bbl@parsejalti{%
5449   \begingroup
5450   \let\bbl@parsejalt\relax % To avoid infinite loop
5451   \edef\bbl@tempb{\fontid\font}%
5452   \bblar@nofswarn
5453   \bblar@fetchjalt\bblar@elongated{}{from}{}%
5454   \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5455   \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5456   \addfontfeature{RawFeature+=jalt}%
5457   % \@namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5458   \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5459   \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5460   \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5461   \directlua{%
5462     for k, v in pairs(Babel.arabic.from) do
5463       if Babel.arabic.dest[k] and
5464         not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5465         Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5466           [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5467       end
5468     end

```

```

5469     }%
5470 \endgroup}
5471 %
5472 \begingroup
5473 \catcode`#=11
5474 \catcode`~=11
5475 \directlua{
5476
5477 Babel.arabic = Babel.arabic or {}
5478 Babel.arabic.from = {}
5479 Babel.arabic.dest = {}
5480 Babel.arabic.justify_factor = 0.95
5481 Babel.arabic.justify_enabled = true
5482
5483 function Babel.arabic.justify(head)
5484   if not Babel.arabic.justify_enabled then return head end
5485   for line in node.traverse_id(node.id'hlist', head) do
5486     Babel.arabic.justify_hlist(head, line)
5487   end
5488   return head
5489 end
5490
5491 function Babel.arabic.justify_hbox(head, gc, size, pack)
5492   local has_inf = false
5493   if Babel.arabic.justify_enabled and pack == 'exactly' then
5494     for n in node.traverse_id(12, head) do
5495       if n.stretch_order > 0 then has_inf = true end
5496     end
5497     if not has_inf then
5498       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5499     end
5500   end
5501   return head
5502 end
5503
5504 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5505   local d, new
5506   local k_list, k_item, pos_inline
5507   local width, width_new, full, k_curr, wt_pos, goal, shift
5508   local subst_done = false
5509   local elong_map = Babel.arabic.elong_map
5510   local last_line
5511   local GLYPH = node.id'glyph'
5512   local KASHIDA = Babel.attr_kashida
5513   local LOCALE = Babel.attr_locale
5514
5515   if line == nil then
5516     line = {}
5517     line.glue_sign = 1
5518     line.glue_order = 0
5519     line.head = head
5520     line.shift = 0
5521     line.width = size
5522   end
5523
5524   % Exclude last line. todo. But-- it discards one-word lines, too!
5525   % ? Look for glue = 12:15
5526   if (line.glue_sign == 1 and line.glue_order == 0) then
5527     elongs = {}      % Stores elongated candidates of each line
5528     k_list = {}      % And all letters with kashida
5529     pos_inline = 0   % Not yet used
5530
5531     for n in node.traverse_id(GLYPH, line.head) do

```

```

5532     pos_inline = pos_inline + 1 % To find where it is. Not used.
5533
5534     % Elongated glyphs
5535     if elong_map then
5536         local locale = node.get_attribute(n, LOCALE)
5537         if elong_map[locale] and elong_map[locale][n.font] and
5538             elong_map[locale][n.font][n.char] then
5539             table.insert(elongs, {node = n, locale = locale} )
5540             node.set_attribute(n.prev, KASHIDA, 0)
5541         end
5542     end
5543
5544     % Tatwil
5545     if Babel.kashida_wts then
5546         local k_wt = node.get_attribute(n, KASHIDA)
5547         if k_wt > 0 then % todo. parameter for multi inserts
5548             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5549         end
5550     end
5551
5552 end % of node.traverse_id
5553
5554 if #elongs == 0 and #k_list == 0 then goto next_line end
5555 full = line.width
5556 shift = line.shift
5557 goal = full * Babel.arabic.justify_factor % A bit crude
5558 width = node.dimensions(line.head) % The 'natural' width
5559
5560 % == Elongated ==
5561 % Original idea taken from 'chickenize'
5562 while (#elongs > 0 and width < goal) do
5563     subst_done = true
5564     local x = #elongs
5565     local curr = elongs[x].node
5566     local oldchar = curr.char
5567     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5568     width = node.dimensions(line.head) % Check if the line is too wide
5569     % Substitute back if the line would be too wide and break:
5570     if width > goal then
5571         curr.char = oldchar
5572         break
5573     end
5574     % If continue, pop the just substituted node from the list:
5575     table.remove(elongs, x)
5576 end
5577
5578 % == Tatwil ==
5579 if #k_list == 0 then goto next_line end
5580
5581 width = node.dimensions(line.head) % The 'natural' width
5582 k_curr = #k_list
5583 wt_pos = 1
5584
5585 while width < goal do
5586     subst_done = true
5587     k_item = k_list[k_curr].node
5588     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5589         d = node.copy(k_item)
5590         d.char = 0x0640
5591         line.head, new = node.insert_after(line.head, k_item, d)
5592         width_new = node.dimensions(line.head)
5593         if width > goal or width == width_new then
5594             node.remove(line.head, new) % Better compute before

```

```

5595         break
5596     end
5597     width = width_new
5598 end
5599 if k_curr == 1 then
5600     k_curr = #k_list
5601     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5602 else
5603     k_curr = k_curr - 1
5604 end
5605 end
5606
5607 ::next_line::
5608
5609 % Must take into account marks and ins, see luatex manual.
5610 % Have to be executed only if there are changes. Investigate
5611 % what's going on exactly.
5612 if subst_done and not gc then
5613     d = node.hpack(line.head, full, 'exactly')
5614     d.shift = shift
5615     node.insert_before(head, line, d)
5616     node.remove(head, line)
5617 end
5618 end % if process line
5619 end
5620 }
5621 \endgroup
5622 \fi\fi % Arabic just block

```

12.7 Common stuff

```

5623 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5624 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5625 \DisableBabelHook{babel-fontspec}
5626 <<Font selection>>

```

12.8 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.

```

5627 % TODO - to a lua file
5628 \directlua{
5629 Babel.script_blocks = {
5630   ['dflt'] = {},
5631   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5632              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5633   ['Armn'] = {{0x0530, 0x058F}},
5634   ['Beng'] = {{0x0980, 0x09FF}},
5635   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5636   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5637   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5638              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5639   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5640   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5641              {0xAB00, 0xAB2F}},
5642   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5643   % Don't follow strictly Unicode, which places some Coptic letters in
5644   % the 'Greek and Coptic' block
5645   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},

```

```

5646 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5647             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5648             {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5649             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5650             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5651             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5652 ['Hebr'] = {{0x0590, 0x05FF}},
5653 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5654             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5655 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5656 ['Knda'] = {{0x0C80, 0x0CFF}},
5657 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5658             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5659             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5660 ['Lao'] = {{0x0E80, 0x0EFF}},
5661 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5662             {0x0180, 0x024F}, {0x01E0, 0x01FF}, {0x2C60, 0x2C7F},
5663             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5664 ['Mahj'] = {{0x11150, 0x1117F}},
5665 ['Mlym'] = {{0x0D00, 0x0D7F}},
5666 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5667 ['Orya'] = {{0x0B00, 0x0B7F}},
5668 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5669 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5670 ['Taml'] = {{0x0B80, 0x0BFF}},
5671 ['Telu'] = {{0x0C00, 0x0C7F}},
5672 ['Tfng'] = {{0x2D30, 0x2D7F}},
5673 ['Thai'] = {{0x0E00, 0x0E7F}},
5674 ['Tibt'] = {{0x0F00, 0x0FFF}},
5675 ['Vaii'] = {{0xA500, 0xA63F}},
5676 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5677 }
5678
5679 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5680 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5681 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5682
5683 function Babel.locale_map(head)
5684   if not Babel.locale_mapped then return head end
5685
5686   local LOCALE = Babel.attr_locale
5687   local GLYPH = node.id('glyph')
5688   local inmath = false
5689   local toloc_save
5690   for item in node.traverse(head) do
5691     local toloc
5692     if not inmath and item.id == GLYPH then
5693       % Optimization: build a table with the chars found
5694       if Babel.chr_to_loc[item.char] then
5695         toloc = Babel.chr_to_loc[item.char]
5696       else
5697         for lc, maps in pairs(Babel.loc_to_scr) do
5698           for _, rg in pairs(maps) do
5699             if item.char >= rg[1] and item.char <= rg[2] then
5700               Babel.chr_to_loc[item.char] = lc
5701               toloc = lc
5702               break
5703             end
5704           end
5705         end
5706       end
5707       % Now, take action, but treat composite chars in a different
5708       % fashion, because they 'inherit' the previous locale. Not yet

```



```

5709 % optimized.
5710 if not toloc and
5711     (item.char >= 0x0300 and item.char <= 0x036F) or
5712     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5713     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5714     toloc = toloc_save
5715 end
5716 if toloc and toloc > -1 then
5717     if Babel.locale_props[toloc].lg then
5718         item.lang = Babel.locale_props[toloc].lg
5719         node.set_attribute(item, LOCALE, toloc)
5720     end
5721     if Babel.locale_props[toloc]['/'..item.font] then
5722         item.font = Babel.locale_props[toloc]['/'..item.font]
5723     end
5724     toloc_save = toloc
5725 end
5726 elseif not inmath and item.id == 7 then
5727     item.replace = item.replace and Babel.locale_map(item.replace)
5728     item.pre      = item.pre and Babel.locale_map(item.pre)
5729     item.post     = item.post and Babel.locale_map(item.post)
5730 elseif item.id == node.id'math' then
5731     inmath = (item.subtype == 0)
5732 end
5733 end
5734 return head
5735 end
5736 }

```

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

```

5737 \newcommand\babelcharproperty[1]{%
5738   \count@=#1\relax
5739   \ifvmode
5740     \expandafter\bbl@chprop
5741   \else
5742     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5743               vertical mode (preamble or between paragraphs)}%
5744     {See the manual for futher info}%
5745   \fi}
5746 \newcommand\bbl@chprop[3][\the\count@]{%
5747   \@tempcnta=#1\relax
5748   \bbl@ifunset{\bbl@chprop@#2}%
5749   {\bbl@error{No property named '#2'. Allowed values are\\%
5750             direction (bc), mirror (bmg), and linebreak (lb)}%
5751    {See the manual for futher info}}%
5752   {}%
5753   \loop
5754     \bbl@cs{chprop@#2}{#3}%
5755     \ifnum\count@<\@tempcnta
5756       \advance\count@\@ne
5757     \repeat}
5758 \def\bbl@chprop@direction#1{%
5759   \directlua{
5760     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5761     Babel.characters[\the\count@]['d'] = '#1'
5762   }}
5763 \let\bbl@chprop@bc\bbl@chprop@direction
5764 \def\bbl@chprop@mirror#1{%
5765   \directlua{
5766     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5767     Babel.characters[\the\count@]['m'] = '\number#1'
5768   }}

```

```

5769 \let\bbl@chprop@bmg\bbl@chprop@mirror
5770 \def\bbl@chprop@linebreak#1{%
5771   \directlua{
5772     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5773     Babel.cjk_characters[\the\count@]['c'] = '#1'
5774   }}
5775 \let\bbl@chprop@lb\bbl@chprop@linebreak
5776 \def\bbl@chprop@locale#1{%
5777   \directlua{
5778     Babel.chr_to_loc = Babel.chr_to_loc or {}
5779     Babel.chr_to_loc[\the\count@] =
5780       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5781   }}

```

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.

```

5782 \directlua{
5783   Babel.nohyphenation = \the\l@nohyphenation
5784 }

```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ - becomes $\text{function}(m) \text{ return } m[1]..m[1]..'-' \text{ end}$, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m) \text{ return } \text{Babel.capt_map}(m[1],1) \text{ end}$, 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 $\text{lua load} - \text{save the code as string in a TeX macro, and expand this macro at the appropriate place}$. 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).

```

5785 \begingroup
5786 \catcode`\~ = 12
5787 \catcode`\% = 12
5788 \catcode`\& = 14
5789 \catcode`\| = 12
5790 \gdef\babelprehyphenation{&&
5791   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
5792 \gdef\babelposthyphenation{&&
5793   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
5794 \gdef\bbl@settransform#1[#2]#3#4#5{&&
5795   \ifcase#1
5796     \bbl@activateprehyphen
5797   \else
5798     \bbl@activateposthyphen
5799   \fi
5800   \begingroup
5801     \def\babeltempa{\bbl@add@list\babeltempb}&&
5802     \let\babeltempb\@empty
5803     \def\bbl@tempa{#5}&&
5804     \bbl@replace\bbl@tempa{,}{,}&& TODO. Ugly trick to preserve {}
5805     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&&
5806       \bbl@ifsamestring{##1}{remove}&&
5807       {\bbl@add@list\babeltempb{nil}}&&
5808       {\directlua{
5809         local rep = {[##1]=}
5810         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5811         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5812         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5813         if #1 == 0 then
5814           rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5815             'space = {' .. '%2, %3, %4' .. '}')
5816           rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5817             'spacefactor = {' .. '%2, %3, %4' .. '}')
5818           rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5819         else

```

```

5820         rep = rep:gsub(    '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5821         rep = rep:gsub(    '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5822         rep = rep:gsub(    '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5823     end
5824     tex.print([[\\string\\babeltempa{[]] .. rep .. [[]]])
5825     }}&%
5826 \\let\\bbl@kv@attribute\\relax
5827 \\let\\bbl@kv@label\\relax
5828 \\bbl@forkv{#2}{\\bbl@csarg\\edef{kv@##1}{##2}}&%
5829 \\ifx\\bbl@kv@attribute\\relax\\else
5830     \\edef\\bbl@kv@attribute{\\expandafter\\bbl@stripslash\\bbl@kv@attribute}&%
5831 \\fi
5832 \\directlua{
5833     local lbkr = Babel.linebreaking.replacements[#1]
5834     local u = unicode.utf8
5835     local id, attr, label
5836     if #1 == 0 then
5837         id = \\the\\csname bbl@id@@#3\\endcsname\\space
5838     else
5839         id = \\the\\csname l@#3\\endcsname\\space
5840     end
5841     \\ifx\\bbl@kv@attribute\\relax
5842         attr = -1
5843     \\else
5844         attr = luatexbase.registernumber'\\bbl@kv@attribute'
5845     \\fi
5846     \\ifx\\bbl@kv@label\\relax\\else    &% Same refs:
5847         label = [==[\\bbl@kv@label]==]
5848     \\fi
5849     &% Convert pattern:
5850     local patt = string.gsub([==[#4]==], '%s', '')
5851     if #1 == 0 then
5852         patt = string.gsub(patt, '|', ' ')
5853     end
5854     if not u.find(patt, '()', nil, true) then
5855         patt = '()' .. patt .. '()'
5856     end
5857     if #1 == 1 then
5858         patt = string.gsub(patt, '%(%)^', '^()')
5859         patt = string.gsub(patt, '%$%(%)', '()$')
5860     end
5861     patt = u.gsub(patt, '{(.)}',
5862         function (n)
5863             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5864         end)
5865     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5866         function (n)
5867             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5868         end)
5869     lbkr[id] = lbkr[id] or {}
5870     table.insert(lbkr[id],
5871         { label=label, attr=attr, pattern=patt, replace=\\babeltempb })
5872     }&%
5873 \\endgroup}
5874 \\endgroup
5875 \\def\\bbl@activateposthyphen{%
5876     \\let\\bbl@activateposthyphen\\relax
5877     \\directlua{
5878         require('babel-transforms.lua')
5879         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5880     }}
5881 \\def\\bbl@activateprehyphen{%
5882     \\let\\bbl@activateprehyphen\\relax

```

```

5883 \directlua{
5884   require('babel-transforms.lua')
5885   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5886 }

```

12.9 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 L^AT_EX. Just in case, consider the possibility it has not been loaded.

```

5887 \def\bbl@activate@preotf{%
5888   \let\bbl@activate@preotf\relax % only once
5889   \directlua{
5890     Babel = Babel or {}
5891     %
5892     function Babel.pre_otfload_v(head)
5893       if Babel.numbers and Babel.digits_mapped then
5894         head = Babel.numbers(head)
5895       end
5896       if Babel.bidi_enabled then
5897         head = Babel.bidi(head, false, dir)
5898       end
5899       return head
5900     end
5901     %
5902     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5903       if Babel.numbers and Babel.digits_mapped then
5904         head = Babel.numbers(head)
5905       end
5906       if Babel.bidi_enabled then
5907         head = Babel.bidi(head, false, dir)
5908       end
5909       return head
5910     end
5911     %
5912     luatexbase.add_to_callback('pre_linebreak_filter',
5913       Babel.pre_otfload_v,
5914       'Babel.pre_otfload_v',
5915       luatexbase.priority_in_callback('pre_linebreak_filter',
5916         'luaotfload.node_processor') or nil)
5917     %
5918     luatexbase.add_to_callback('hpack_filter',
5919       Babel.pre_otfload_h,
5920       'Babel.pre_otfload_h',
5921       luatexbase.priority_in_callback('hpack_filter',
5922         'luaotfload.node_processor') or nil)
5923   }

```

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=`.

```

5924 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5925   \let\bbl@beforeforeign\leavevmode
5926   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5927   \RequirePackage{luatexbase}
5928   \bbl@activate@preotf
5929   \directlua{
5930     require('babel-data-bidi.lua')
5931     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5932       require('babel-bidi-basic.lua')
5933     \or
5934       require('babel-bidi-basic-r.lua')
5935     \fi}

```

```

5936 % TODO - to locale_props, not as separate attribute
5937 \newattribute\bbl@attr@dir
5938 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5939 % TODO. I don't like it, hackish:
5940 \bbl@exp{\output{\bodydir\pagedir\the\output}}
5941 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5942 \fi\fi
5943 \chardef\bbl@thetextdir\z@
5944 \chardef\bbl@thepardir\z@
5945 \def\bbl@getluadir#1{%
5946   \directlua{
5947     if tex.#1dir == 'TLT' then
5948       tex.sprint('0')
5949     elseif tex.#1dir == 'TRT' then
5950       tex.sprint('1')
5951     end}}
5952 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5953   \ifcase#3\relax
5954     \ifcase\bbl@getluadir{#1}\relax\else
5955       #2 TLT\relax
5956     \fi
5957   \else
5958     \ifcase\bbl@getluadir{#1}\relax
5959       #2 TRT\relax
5960     \fi
5961   \fi}
5962 \def\bbl@thedir{0}
5963 \def\bbl@textdir#1{%
5964   \bbl@setluadir{text}\textdir{#1}%
5965   \chardef\bbl@thetextdir#1\relax
5966   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5967   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5968 \def\bbl@pardir#1{%
5969   \bbl@setluadir{par}\pardir{#1}%
5970   \chardef\bbl@thepardir#1\relax}
5971 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5972 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5973 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
5974 %
5975 \ifnum\bbl@bidimode>\z@
5976   \def\bbl@insidemath{0}%
5977   \def\bbl@everymath{\def\bbl@insidemath{1}}
5978   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5979   \frozen@everymath\expandafter{%
5980     \expandafter\bbl@everymath\the\frozen@everymath}
5981   \frozen@everydisplay\expandafter{%
5982     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
5983   \AtBeginDocument{
5984     \directlua{
5985       function Babel.math_box_dir(head)
5986         if not (token.get_macro('bbl@insidemath') == '0') then
5987           if Babel.hlist_has_bidi(head) then
5988             local d = node.new(node.id'dir')
5989             d.dir = '+TRT'
5990             node.insert_before(head, node.has_glyph(head), d)
5991             for item in node.traverse(head) do
5992               node.set_attribute(item,
5993                 Babel.attr_dir, token.get_macro('bbl@thedir'))
5994             end
5995           end
5996         end
5997         return head
5998       end

```

```

5999     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6000     "Babel.math_box_dir", 0)
6001   }}%
6002 \fi

```

12.10 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`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolum` still fails.

```

6003 \bbl@trace{Redefinitions for bidi layout}
6004 %
6005 <(*More package options)> ≡
6006 \chardef\bbl@eqnpos\z@
6007 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6008 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6009 <(/More package options)>
6010 %
6011 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6012 \ifnum\bbl@bidimode>\z@
6013   \ifx\matheqdirmode\undefined\else
6014     \matheqdirmode\@ne
6015   \fi
6016   \let\bbl@eqnodir\relax
6017   \def\bbl@eqdel{()}
6018   \def\bbl@eqnum{%
6019     {\normalfont\normalcolor
6020     \expandafter\@firstoftwo\bbl@eqdel
6021     \theequation
6022     \expandafter\@secondoftwo\bbl@eqdel}}
6023   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6024   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6025   \def\bbl@eqno@flip#1{%
6026     \ifdim\predisplaysize=-\maxdimen
6027       \eqno
6028       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6029     \else
6030       \leqno\hbox{#1}%
6031     \fi}
6032   \def\bbl@leqno@flip#1{%
6033     \ifdim\predisplaysize=-\maxdimen
6034       \leqno
6035       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6036     \else
6037       \eqno\hbox{#1}%
6038     \fi}
6039   \AtBeginDocument{%
6040     \ifx\maketag@@@\undefined % Normal equation, eqnarray
6041       \AddToHook{env/equation/begin}{%
6042         \ifnum\bbl@thetextdir>\z@
6043           \let\@eqnnum\bbl@eqnum
6044           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6045           \chardef\bbl@thetextdir\z@
6046           \bbl@add\normalfont{\bbl@eqnodir}%

```

```

6047         \ifcase\bb1@eqnpos
6048             \let\bb1@puteqno\bb1@eqno@flip
6049         \or
6050             \let\bb1@puteqno\bb1@leqno@flip
6051         \fi
6052     \fi}%
6053 \ifnum\bb1@eqnpos=\tw@%else
6054     \def\endequation{\bb1@puteqno{\@eqnnum}$$\@ignoretrue}%
6055 \fi
6056 \AddToHook{env/eqnarray/begin}{%
6057     \ifnum\bb1@thetextdir>\z@
6058         \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6059         \chardef\bb1@thetextdir\z@
6060         \bb1@add\normalfont{\bb1@eqnodir}%
6061         \ifnum\bb1@eqnpos=\@ne
6062             \def\@eqnnum{%
6063                 \setbox\z@\hbox{\bb1@eqnum}%
6064                 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6065             \else
6066                 \let\@eqnnum\bb1@eqnum
6067             \fi
6068         \fi}
6069 % Hack. YA luatex bug?:
6070 \expandafter\bb1@sreplace\csname] \endcsname{${$}{\eqno\kern.001pt$}}%
6071 \else % amstex
6072     \ifx\bb1@noamsmath\@undefined
6073         \ifnum\bb1@eqnpos=\@ne
6074             \let\bb1@ams@lap\hbox
6075         \else
6076             \let\bb1@ams@lap\llap
6077         \fi
6078         \ExplSyntaxOn
6079         \bb1@sreplace\intertext@{\normalbaselines}%
6080         {\normalbaselines
6081             \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6082         \ExplSyntaxOff
6083         \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6084         \ifx\bb1@ams@lap\hbox % leqno
6085             \def\bb1@ams@flip#1{%
6086                 \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6087         \else % eqno
6088             \def\bb1@ams@flip#1{%
6089                 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6090         \fi
6091         \def\bb1@ams@preset#1{%
6092             \ifnum\bb1@thetextdir>\z@
6093                 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6094                 \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6095                 \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6096             \fi}%
6097     \ifnum\bb1@eqnpos=\tw@%else
6098         \def\bb1@ams@equation{%
6099             \ifnum\bb1@thetextdir>\z@
6100                 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6101                 \chardef\bb1@thetextdir\z@
6102                 \bb1@add\normalfont{\bb1@eqnodir}%
6103                 \ifcase\bb1@eqnpos
6104                     \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6105                 \or
6106                     \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6107                 \fi
6108             \fi}%
6109     \AddToHook{env/equation/begin}{\bb1@ams@equation}%

```

```

6110      \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6111      \fi
6112      \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6113      \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6114      \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6115      \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6116      \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6117      \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6118      \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6119      % Hackish, for proper alignment. Don't ask me why it works!:
6120      \bbl@exp{% Avoid a 'visible' conditional
6121        \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}<fi>}}%
6122      \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6123      \AddToHook{env/split/before}{%
6124        \ifnum\bbl@thetextdir>\z@
6125          \bbl@ifsamestring@currentenv{equation}%
6126          {\ifx\bbl@ams@lap\hbox % leqno
6127            \def\bbl@ams@flip#1{%
6128              \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6129            \else
6130              \def\bbl@ams@flip#1{%
6131                \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6132              \fi}%
6133            }%
6134          \fi}%
6135      \fi
6136      \fi}
6137 \fi
6138 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
6139 \ifnum\bbl@bidimode>\z@
6140   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6141     \bbl@exp{%
6142       \def\\bbl@insidemath{0}%
6143       \mathdir\the\bodydir
6144       #1%           Once entered in math, set boxes to restore values
6145       \<ifmmode>%
6146         \everyvbox{%
6147           \the\everyvbox
6148           \bodydir\the\bodydir
6149           \mathdir\the\mathdir
6150           \everyhbox{\the\everyhbox}%
6151           \everyvbox{\the\everyvbox}}%
6152         \everyhbox{%
6153           \the\everyhbox
6154           \bodydir\the\bodydir
6155           \mathdir\the\mathdir
6156           \everyhbox{\the\everyhbox}%
6157           \everyvbox{\the\everyvbox}}%
6158       \<fi>}}%
6159   \def\@hangfrom#1{%
6160     \setbox\@tempboxa\hbox{#{1}}%
6161     \hangindent\wd\@tempboxa
6162     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6163       \shapemode\@ne
6164     \fi
6165     \noindent\box\@tempboxa}
6166 \fi
6167 \IfBabelLayout{tabular}
6168 {\let\bbl@OL@@tabular\@tabular
6169  \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6170  \let\bbl@NL@@tabular\@tabular
6171  \AtBeginDocument{%
6172    \ifx\bbl@NL@@tabular\@tabular\else

```



```

6173      \bbl@replace\@tabular{$}\bbl@nextfake$}%
6174      \let\bbl@NL\@tabular\@tabular
6175      \fi}}
6176      {}
6177 \IfBabelLayout{lists}
6178   {\let\bbl@OL\list\list
6179     \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6180     \let\bbl@NL\list\list
6181     \def\bbl@listparshape#1#2#3{%
6182       \parshape #1 #2 #3 %
6183       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6184         \shapemode\tw@
6185       \fi}}
6186   {}
6187 \IfBabelLayout{graphics}
6188   {\let\bbl@pictresetdir\relax
6189     \def\bbl@pictsetdir#1{%
6190       \ifcase\bbl@thetextdir
6191         \let\bbl@pictresetdir\relax
6192       \else
6193         \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6194           \or\textdir TLT
6195           \else\bodydir TLT \textdir TLT
6196         \fi
6197         % \(\text|par)dir required in pgf:
6198         \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6199       \fi}%
6200   \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6201   \directlua{
6202     Babel.get_picture_dir = true
6203     Babel.picture_has_bidi = 0
6204     %
6205     function Babel.picture_dir (head)
6206       if not Babel.get_picture_dir then return head end
6207       if Babel.hlist_has_bidi(head) then
6208         Babel.picture_has_bidi = 1
6209       end
6210       return head
6211     end
6212     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6213       "Babel.picture_dir")
6214   }%
6215 \AtBeginDocument{%
6216   \long\def\put(#1,#2)#3{%
6217     \@killglue
6218     % Try:
6219     \ifx\bbl@pictresetdir\relax
6220       \def\bbl@tempc{0}%
6221     \else
6222       \directlua{
6223         Babel.get_picture_dir = true
6224         Babel.picture_has_bidi = 0
6225       }%
6226       \setbox\z@\hb@xt@\z@{%
6227         \@defaultunitsset\@tempdimc{#1}\unitlength
6228         \kern\@tempdimc
6229         #3\hss}% TODO: #3 executed twice (below). That's bad.
6230       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6231     \fi
6232     % Do:
6233     \@defaultunitsset\@tempdimc{#2}\unitlength
6234     \raise\@tempdimc\hb@xt@\z@{%
6235       \@defaultunitsset\@tempdimc{#1}\unitlength

```

```

6236      \kern\@tempdimc
6237      {\ifnum\bb1@tempc>\z@\bb1@pictresetdir\fi#3}\hss}%
6238      \ignorespaces}%
6239      \MakeRobust\put}%
6240      \AtBeginDocument
6241      {\AddToHook{cmd/diagbox@pict/before}{\let\bb1@pictsetdir\@gobble}%
6242      \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6243      \AddToHook{env/pgfpicture/begin}{\bb1@pictsetdir\@ne}%
6244      \bb1@add\pgfinterruptpicture{\bb1@pictresetdir}%
6245      \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir\z@}%
6246      \fi
6247      \ifx\tikzpicture\@undefined\else
6248      \AddToHook{env/tikzpicture/begin}{\bb1@pictsetdir\z@}%
6249      \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
6250      \bb1@sreplace\tikz{\begingroup}{\begingroup\bb1@pictsetdir\tw@}%
6251      \fi
6252      \ifx\tcolorbox\@undefined\else
6253      \def\tcb@drawing@env@begin{%
6254      \csname tcb@before@tcb@split@state\endcsname
6255      \bb1@pictsetdir\tw@
6256      \begin{\kv tcb@graphenv}%
6257      \tcb@bbdraw%
6258      \tcb@apply@graph@patches
6259      }%
6260      \def\tcb@drawing@env@end{%
6261      \end{\kv tcb@graphenv}%
6262      \bb1@pictresetdir
6263      \csname tcb@after@tcb@split@state\endcsname
6264      }%
6265      \fi
6266      }}
6267      {}

```

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`.

```

6268 \IfBabelLayout{counters}%
6269 {\let\bb1@OL@textsuperscript\@textsuperscript
6270 \bb1@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6271 \let\bb1@latinarabic=\@arabic
6272 \let\bb1@OL@@arabic\@arabic
6273 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
6274 \ifpackagewith{babel}{bidi=default}%
6275 {\let\bb1@asciroman=\@roman
6276 \let\bb1@OL@@roman\@roman
6277 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
6278 \let\bb1@asciiRoman=\@Roman
6279 \let\bb1@OL@@roman\@Roman
6280 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}%
6281 \let\bb1@OL@labelenumii\labelenumii
6282 \def\labelenumii{\theenumii}%
6283 \let\bb1@OL@p@enumiii\p@enumiii
6284 \def\p@enumiii{\p@enumii}\theenumii{}}{}{}{}
6285 <<Footnote changes>>
6286 \IfBabelLayout{footnotes}%
6287 {\let\bb1@OL@footnote\footnote
6288 \BabelFootnote\footnote\language{}{}}%
6289 \BabelFootnote\localfootnote\language{}{}}%
6290 \BabelFootnote\mainfootnote{}{}}{}
6291 {}

```

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

```

6292 \IfBabelLayout{extras}%

```

```

6293 {\let\bbl@OL@underline\underline
6294 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6295 \let\bbl@OL@LaTeX2e\LaTeX2e
6296 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6297 \if b\expandafter\@car\f@series\@nil\boldmath\fi
6298 \babelsublr{%
6299 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}}}}
6300 {}
6301 \</luatex>

```

12.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex manual`), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

6302 (*transforms)
6303 Babel.linebreaking.replacements = {}
6304 Babel.linebreaking.replacements[0] = {} -- pre
6305 Babel.linebreaking.replacements[1] = {} -- post
6306
6307 -- Discretionaries contain strings as nodes
6308 function Babel.str_to_nodes(fn, matches, base)
6309     local n, head, last
6310     if fn == nil then return nil end
6311     for s in string.utfvalues(fn(matches)) do
6312         if base.id == 7 then
6313             base = base.replace
6314         end
6315         n = node.copy(base)
6316         n.char = s
6317         if not head then
6318             head = n
6319         else
6320             last.next = n
6321         end
6322         last = n
6323     end
6324     return head
6325 end
6326
6327 Babel.fetch_subtext = {}
6328
6329 Babel.ignore_pre_char = function(node)
6330     return (node.lang == Babel.nohyphenation)
6331 end
6332
6333 -- Merging both functions doesn't seem feasible, because there are too
6334 -- many differences.
6335 Babel.fetch_subtext[0] = function(head)
6336     local word_string = ''
6337     local word_nodes = {}
6338     local lang
6339     local item = head
6340     local inmath = false
6341

```

```

6342 while item do
6343
6344     if item.id == 11 then
6345         inmath = (item.subtype == 0)
6346     end
6347
6348     if inmath then
6349         -- pass
6350
6351     elseif item.id == 29 then
6352         local locale = node.get_attribute(item, Babel.attr_locale)
6353
6354         if lang == locale or lang == nil then
6355             lang = lang or locale
6356             if Babel.ignore_pre_char(item) then
6357                 word_string = word_string .. Babel.us_char
6358             else
6359                 word_string = word_string .. unicode.utf8.char(item.char)
6360             end
6361             word_nodes[#word_nodes+1] = item
6362         else
6363             break
6364         end
6365
6366     elseif item.id == 12 and item.subtype == 13 then
6367         word_string = word_string .. ' '
6368         word_nodes[#word_nodes+1] = item
6369
6370     -- Ignore leading unrecognized nodes, too.
6371     elseif word_string ~= '' then
6372         word_string = word_string .. Babel.us_char
6373         word_nodes[#word_nodes+1] = item -- Will be ignored
6374     end
6375
6376     item = item.next
6377 end
6378
6379 -- Here and above we remove some trailing chars but not the
6380 -- corresponding nodes. But they aren't accessed.
6381 if word_string:sub(-1) == ' ' then
6382     word_string = word_string:sub(1,-2)
6383 end
6384 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6385 return word_string, word_nodes, item, lang
6386 end
6387
6388 Babel.fetch_subtext[1] = function(head)
6389     local word_string = ''
6390     local word_nodes = {}
6391     local lang
6392     local item = head
6393     local inmath = false
6394
6395     while item do
6396
6397         if item.id == 11 then
6398             inmath = (item.subtype == 0)
6399         end
6400
6401         if inmath then
6402             -- pass
6403
6404         elseif item.id == 29 then

```

```

6405     if item.lang == lang or lang == nil then
6406         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6407             lang = lang or item.lang
6408             word_string = word_string .. unicode.utf8.char(item.char)
6409             word_nodes[#word_nodes+1] = item
6410         end
6411     else
6412         break
6413     end
6414
6415     elseif item.id == 7 and item.subtype == 2 then
6416         word_string = word_string .. '='
6417         word_nodes[#word_nodes+1] = item
6418
6419     elseif item.id == 7 and item.subtype == 3 then
6420         word_string = word_string .. '|'
6421         word_nodes[#word_nodes+1] = item
6422
6423     -- (1) Go to next word if nothing was found, and (2) implicitly
6424     -- remove leading USs.
6425     elseif word_string == '' then
6426         -- pass
6427
6428     -- This is the responsible for splitting by words.
6429     elseif (item.id == 12 and item.subtype == 13) then
6430         break
6431
6432     else
6433         word_string = word_string .. Babel.us_char
6434         word_nodes[#word_nodes+1] = item -- Will be ignored
6435     end
6436
6437     item = item.next
6438 end
6439
6440 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6441 return word_string, word_nodes, item, lang
6442 end
6443
6444 function Babel.pre_hyphenate_replace(head)
6445     Babel.hyphenate_replace(head, 0)
6446 end
6447
6448 function Babel.post_hyphenate_replace(head)
6449     Babel.hyphenate_replace(head, 1)
6450 end
6451
6452 Babel.us_char = string.char(31)
6453
6454 function Babel.hyphenate_replace(head, mode)
6455     local u = unicode.utf8
6456     local lbkr = Babel.linebreaking.replacements[mode]
6457
6458     local word_head = head
6459
6460     while true do -- for each subtext block
6461
6462         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6463
6464         if Babel.debug then
6465             print()
6466             print((mode == 0) and '@@@<' or '@@@>', w)
6467         end

```

```

6468
6469     if nw == nil and w == '' then break end
6470
6471     if not lang then goto next end
6472     if not lbkr[lang] then goto next end
6473
6474     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6475     -- loops are nested.
6476     for k=1, #lbkr[lang] do
6477         local p = lbkr[lang][k].pattern
6478         local r = lbkr[lang][k].replace
6479         local attr = lbkr[lang][k].attr or -1
6480
6481         if Babel.debug then
6482             print('*****', p, mode)
6483         end
6484
6485         -- This variable is set in some cases below to the first *byte*
6486         -- after the match, either as found by u.match (faster) or the
6487         -- computed position based on sc if w has changed.
6488         local last_match = 0
6489         local step = 0
6490
6491         -- For every match.
6492         while true do
6493             if Babel.debug then
6494                 print('====')
6495             end
6496             local new -- used when inserting and removing nodes
6497
6498             local matches = { u.match(w, p, last_match) }
6499
6500             if #matches < 2 then break end
6501
6502             -- Get and remove empty captures (with ()'s, which return a
6503             -- number with the position), and keep actual captures
6504             -- (from (...)), if any, in matches.
6505             local first = table.remove(matches, 1)
6506             local last = table.remove(matches, #matches)
6507             -- Non re-fetched substrings may contain \31, which separates
6508             -- substrings.
6509             if string.find(w:sub(first, last-1), Babel.us_char) then break end
6510
6511             local save_last = last -- with A()BC()D, points to D
6512
6513             -- Fix offsets, from bytes to unicode. Explained above.
6514             first = u.len(w:sub(1, first-1)) + 1
6515             last = u.len(w:sub(1, last-1)) -- now last points to C
6516
6517             -- This loop stores in a small table the nodes
6518             -- corresponding to the pattern. Used by 'data' to provide a
6519             -- predictable behavior with 'insert' (w_nodes is modified on
6520             -- the fly), and also access to 'remove'd nodes.
6521             local sc = first-1 -- Used below, too
6522             local data_nodes = {}
6523
6524             local enabled = true
6525             for q = 1, last-first+1 do
6526                 data_nodes[q] = w_nodes[sc+q]
6527                 if enabled
6528                     and attr > -1
6529                     and not node.has_attribute(data_nodes[q], attr)
6530                 then

```

```

6531         enabled = false
6532     end
6533 end
6534
6535 -- This loop traverses the matched substring and takes the
6536 -- corresponding action stored in the replacement list.
6537 -- sc = the position in substr nodes / string
6538 -- rc = the replacement table index
6539 local rc = 0
6540
6541 while rc < last-first+1 do -- for each replacement
6542     if Babel.debug then
6543         print('.....', rc + 1)
6544     end
6545     sc = sc + 1
6546     rc = rc + 1
6547
6548     if Babel.debug then
6549         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6550         local ss = ''
6551         for itt in node.traverse(head) do
6552             if itt.id == 29 then
6553                 ss = ss .. unicode.utf8.char(itt.char)
6554             else
6555                 ss = ss .. '{' .. itt.id .. '}'
6556             end
6557         end
6558         print('*****', ss)
6559     end
6560
6561     local crep = r[rc]
6562     local item = w_nodes[sc]
6563     local item_base = item
6564     local placeholder = Babel.us_char
6565     local d
6566
6567     if crep and crep.data then
6568         item_base = data_nodes[crep.data]
6569     end
6570
6571     if crep then
6572         step = crep.step or 0
6573     end
6574
6575     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6576         last_match = save_last -- Optimization
6577         goto next
6578     end
6579
6580     elseif crep == nil or crep.remove then
6581         node.remove(head, item)
6582         table.remove(w_nodes, sc)
6583         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6584         sc = sc - 1 -- Nothing has been inserted.
6585         last_match = utf8.offset(w, sc+1+step)
6586         goto next
6587
6588     elseif crep and crep.kashida then -- Experimental
6589         node.set_attribute(item,
6590             Babel.attr_kashida,
6591             crep.kashida)
6592         last_match = utf8.offset(w, sc+1+step)
6593         goto next

```

```

6594
6595 elseif crep and crep.string then
6596   local str = crep.string(matches)
6597   if str == '' then -- Gather with nil
6598     node.remove(head, item)
6599     table.remove(w_nodes, sc)
6600     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6601     sc = sc - 1 -- Nothing has been inserted.
6602   else
6603     local loop_first = true
6604     for s in string.utfvalues(str) do
6605       d = node.copy(item_base)
6606       d.char = s
6607       if loop_first then
6608         loop_first = false
6609         head, new = node.insert_before(head, item, d)
6610         if sc == 1 then
6611           word_head = head
6612         end
6613         w_nodes[sc] = d
6614         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6615       else
6616         sc = sc + 1
6617         head, new = node.insert_before(head, item, d)
6618         table.insert(w_nodes, sc, new)
6619         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6620       end
6621       if Babel.debug then
6622         print('.....', 'str')
6623         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6624       end
6625     end -- for
6626     node.remove(head, item)
6627   end -- if ''
6628   last_match = utf8.offset(w, sc+1+step)
6629   goto next
6630
6631 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6632   d = node.new(7, 0) -- (disc, discretionary)
6633   d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6634   d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6635   d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6636   d.attr = item_base.attr
6637   if crep.pre == nil then -- TeXbook p96
6638     d.penalty = crep.penalty or tex.hyphenpenalty
6639   else
6640     d.penalty = crep.penalty or tex.exhyphenpenalty
6641   end
6642   placeholder = '|'
6643   head, new = node.insert_before(head, item, d)
6644
6645 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6646   -- ERROR
6647
6648 elseif crep and crep.penalty then
6649   d = node.new(14, 0) -- (penalty, userpenalty)
6650   d.attr = item_base.attr
6651   d.penalty = crep.penalty
6652   head, new = node.insert_before(head, item, d)
6653
6654 elseif crep and crep.space then
6655   -- 655360 = 10 pt = 10 * 65536 sp
6656   d = node.new(12, 13) -- (glue, spaceskip)

```



```

6657         local quad = font.getfont(item_base.font).size or 655360
6658         node.setglue(d, crep.space[1] * quad,
6659                     crep.space[2] * quad,
6660                     crep.space[3] * quad)
6661         if mode == 0 then
6662             placeholder = ' '
6663         end
6664         head, new = node.insert_before(head, item, d)
6665
6666     elseif crep and crep.spacefactor then
6667         d = node.new(12, 13) -- (glue, spaceskip)
6668         local base_font = font.getfont(item_base.font)
6669         node.setglue(d,
6670                     crep.spacefactor[1] * base_font.parameters['space'],
6671                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
6672                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
6673         if mode == 0 then
6674             placeholder = ' '
6675         end
6676         head, new = node.insert_before(head, item, d)
6677
6678     elseif mode == 0 and crep and crep.space then
6679         -- ERROR
6680
6681     end -- ie replacement cases
6682
6683     -- Shared by disc, space and penalty.
6684     if sc == 1 then
6685         word_head = head
6686     end
6687     if crep.insert then
6688         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6689         table.insert(w_nodes, sc, new)
6690         last = last + 1
6691     else
6692         w_nodes[sc] = d
6693         node.remove(head, item)
6694         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6695     end
6696
6697     last_match = utf8.offset(w, sc+1+step)
6698
6699     ::next::
6700
6701     end -- for each replacement
6702
6703     if Babel.debug then
6704         print('.....', '/')
6705         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6706     end
6707
6708     end -- for match
6709
6710     end -- for patterns
6711
6712     ::next::
6713     word_head = nw
6714     end -- for substring
6715     return head
6716 end
6717
6718 -- This table stores capture maps, numbered consecutively
6719 Babel.capture_maps = {}

```

```

6720
6721 -- The following functions belong to the next macro
6722 function Babel.capture_func(key, cap)
6723   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6724   local cnt
6725   local u = unicode.utf8
6726   ret, cnt = ret:gsub('{{[0-9]}|^|+|(.-)}', Babel.capture_func_map)
6727   if cnt == 0 then
6728     ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6729                 function (n)
6730                   return u.char(tonumber(n, 16))
6731                 end)
6732   end
6733   ret = ret:gsub("%[%]%%.%", '')
6734   ret = ret:gsub("%.%[%]%%", '')
6735   return key .. "[[=function(m) return ]] .. ret .. [[ end]]
6736 end
6737
6738 function Babel.capt_map(from, mapno)
6739   return Babel.capture_maps[mapno][from] or from
6740 end
6741
6742 -- Handle the {n|abc|ABC} syntax in captures
6743 function Babel.capture_func_map(capno, from, to)
6744   local u = unicode.utf8
6745   from = u.gsub(from, '{{(%x%x%x%x+)}',
6746               function (n)
6747                 return u.char(tonumber(n, 16))
6748               end)
6749   to = u.gsub(to, '{{(%x%x%x%x+)}',
6750             function (n)
6751               return u.char(tonumber(n, 16))
6752             end)
6753   local froms = {}
6754   for s in string.utfcharacters(from) do
6755     table.insert(froms, s)
6756   end
6757   local cnt = 1
6758   table.insert(Babel.capture_maps, {})
6759   local mlen = table.getn(Babel.capture_maps)
6760   for s in string.utfcharacters(to) do
6761     Babel.capture_maps[mlen][froms[cnt]] = s
6762     cnt = cnt + 1
6763   end
6764   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6765         (mlen) .. ").." .. "[["
6766 end
6767
6768 -- Create/Extend reversed sorted list of kashida weights:
6769 function Babel.capture_kashida(key, wt)
6770   wt = tonumber(wt)
6771   if Babel.kashida_wts then
6772     for p, q in ipairs(Babel.kashida_wts) do
6773       if wt == q then
6774         break
6775       elseif wt > q then
6776         table.insert(Babel.kashida_wts, p, wt)
6777         break
6778       elseif table.getn(Babel.kashida_wts) == p then
6779         table.insert(Babel.kashida_wts, wt)
6780       end
6781     end
6782   else

```

```

6783     Babel.kashida_wts = { wt }
6784 end
6785 return 'kashida = ' .. wt
6786 end
6787 </transforms>

```

12.12 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):

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

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

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

```

6788 <*basic-r>
6789 Babel = Babel or {}
6790
6791 Babel.bidi_enabled = true
6792
6793 require('babel-data-bidi.lua')
6794
6795 local characters = Babel.characters
6796 local ranges = Babel.ranges
6797
6798 local DIR = node.id("dir")
6799
6800 local function dir_mark(head, from, to, outer)
6801   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6802   local d = node.new(DIR)
6803   d.dir = '+' .. dir
6804   node.insert_before(head, from, d)
6805   d = node.new(DIR)
6806   d.dir = '-' .. dir

```

```

6807 node.insert_after(head, to, d)
6808 end
6809
6810 function Babel.bidi(head, ispar)
6811   local first_n, last_n          -- first and last char with nums
6812   local last_es                  -- an auxiliary 'last' used with nums
6813   local first_d, last_d          -- first and last char in L/R block
6814   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):

```

6815   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6816   local strong_lr = (strong == 'l') and 'l' or 'r'
6817   local outer = strong
6818
6819   local new_dir = false
6820   local first_dir = false
6821   local inmath = false
6822
6823   local last_lr
6824
6825   local type_n = ''
6826
6827   for item in node.traverse(head) do
6828
6829     -- three cases: glyph, dir, otherwise
6830     if item.id == node.id'glyph'
6831       or (item.id == 7 and item.subtype == 2) then
6832
6833       local itemchar
6834       if item.id == 7 and item.subtype == 2 then
6835         itemchar = item.replace.char
6836       else
6837         itemchar = item.char
6838       end
6839       local chardata = characters[itemchar]
6840       dir = chardata and chardata.d or nil
6841       if not dir then
6842         for nn, et in ipairs(ranges) do
6843           if itemchar < et[1] then
6844             break
6845           elseif itemchar <= et[2] then
6846             dir = et[3]
6847             break
6848           end
6849         end
6850       end
6851       dir = dir or 'l'
6852       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).

```

6853   if new_dir then
6854     attr_dir = 0
6855     for at in node.traverse(item.attr) do
6856       if at.number == Babel.attr_dir then
6857         attr_dir = at.value % 3
6858       end
6859     end
6860     if attr_dir == 1 then

```

```

6861         strong = 'r'
6862     elseif attr_dir == 2 then
6863         strong = 'al'
6864     else
6865         strong = 'l'
6866     end
6867     strong_lr = (strong == 'l') and 'l' or 'r'
6868     outer = strong_lr
6869     new_dir = false
6870 end
6871
6872 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

6873     dir_real = dir          -- We need dir_real to set strong below
6874     if dir == 'al' then dir = 'r' end -- W3

```

By W2, there are no <en> <et> <es> if strong == <al>, only <an>. Therefore, there are not <et en> nor <en et>, W5 can be ignored, and W6 applied:

```

6875     if strong == 'al' then
6876         if dir == 'en' then dir = 'an' end          -- W2
6877         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6878         strong_lr = 'r'          -- W3
6879     end

```

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

```

6880     elseif item.id == node.id'dir' and not inmath then
6881         new_dir = true
6882         dir = nil
6883     elseif item.id == node.id'math' then
6884         inmath = (item.subtype == 0)
6885     else
6886         dir = nil          -- Not a char
6887     end

```

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

```

6888     if dir == 'en' or dir == 'an' or dir == 'et' then
6889         if dir ~= 'et' then
6890             type_n = dir
6891         end
6892         first_n = first_n or item
6893         last_n = last_es or item
6894         last_es = nil
6895     elseif dir == 'es' and last_n then -- W3+W6
6896         last_es = item
6897     elseif dir == 'cs' then          -- it's right - do nothing
6898     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6899         if strong_lr == 'r' and type_n ~= '' then
6900             dir_mark(head, first_n, last_n, 'r')
6901         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6902             dir_mark(head, first_n, last_n, 'r')
6903             dir_mark(head, first_d, last_d, outer)
6904             first_d, last_d = nil, nil
6905         elseif strong_lr == 'l' and type_n ~= '' then
6906             last_d = last_n
6907         end
6908         type_n = ''
6909         first_n, last_n = nil, nil
6910     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:

```

6911   if dir == 'l' or dir == 'r' then
6912       if dir ~= outer then
6913           first_d = first_d or item
6914           last_d = item
6915       elseif first_d and dir ~= strong_lr then
6916           dir_mark(head, first_d, last_d, outer)
6917           first_d, last_d = nil, nil
6918       end
6919   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.

```

6920   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6921       item.char = characters[item.char] and
6922           characters[item.char].m or item.char
6923   elseif (dir or new_dir) and last_lr ~= item then
6924       local mir = outer .. strong_lr .. (dir or outer)
6925       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6926           for ch in node.traverse(node.next(last_lr)) do
6927               if ch == item then break end
6928               if ch.id == node.id'glyph' and characters[ch.char] then
6929                   ch.char = characters[ch.char].m or ch.char
6930               end
6931           end
6932       end
6933   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).

```

6934   if dir == 'l' or dir == 'r' then
6935       last_lr = item
6936       strong = dir_real           -- Don't search back - best save now
6937       strong_lr = (strong == 'l') and 'l' or 'r'
6938   elseif new_dir then
6939       last_lr = nil
6940   end
6941 end

```

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

```

6942   if last_lr and outer == 'r' then
6943       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6944           if characters[ch.char] then
6945               ch.char = characters[ch.char].m or ch.char
6946           end
6947       end
6948   end
6949   if first_n then
6950       dir_mark(head, first_n, last_n, outer)
6951   end
6952   if first_d then
6953       dir_mark(head, first_d, last_d, outer)
6954   end

```

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

```

6955   return node.prev(head) or head

```

```

6956 end
6957 </basic-r>

And here the Lua code for bidi=basic:

6958 <*basic>
6959 Babel = Babel or {}
6960
6961 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6962
6963 Babel.fontmap = Babel.fontmap or {}
6964 Babel.fontmap[0] = {}      -- l
6965 Babel.fontmap[1] = {}      -- r
6966 Babel.fontmap[2] = {}      -- al/an
6967
6968 Babel.bidi_enabled = true
6969 Babel.mirroring_enabled = true
6970
6971 require('babel-data-bidi.lua')
6972
6973 local characters = Babel.characters
6974 local ranges = Babel.ranges
6975
6976 local DIR = node.id('dir')
6977 local GLYPH = node.id('glyph')
6978
6979 local function insert_implicit(head, state, outer)
6980   local new_state = state
6981   if state.sim and state.eim and state.sim ~= state.eim then
6982     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6983     local d = node.new(DIR)
6984     d.dir = '+' .. dir
6985     node.insert_before(head, state.sim, d)
6986     local d = node.new(DIR)
6987     d.dir = '-' .. dir
6988     node.insert_after(head, state.eim, d)
6989   end
6990   new_state.sim, new_state.eim = nil, nil
6991   return head, new_state
6992 end
6993
6994 local function insert_numeric(head, state)
6995   local new
6996   local new_state = state
6997   if state.san and state.ean and state.san ~= state.ean then
6998     local d = node.new(DIR)
6999     d.dir = '+TLT'
7000     _, new = node.insert_before(head, state.san, d)
7001     if state.san == state.sim then state.sim = new end
7002     local d = node.new(DIR)
7003     d.dir = '-TLT'
7004     _, new = node.insert_after(head, state.ean, d)
7005     if state.ean == state.eim then state.eim = new end
7006   end
7007   new_state.san, new_state.ean = nil, nil
7008   return head, new_state
7009 end
7010
7011 -- TODO - \hbox with an explicit dir can lead to wrong results
7012 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7013 -- was s made to improve the situation, but the problem is the 3-dir
7014 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7015 -- well.
7016

```

```

7017 function Babel.bidi(head, ispar, hdir)
7018   local d    -- d is used mainly for computations in a loop
7019   local prev_d = ''
7020   local new_d = false
7021
7022   local nodes = {}
7023   local outer_first = nil
7024   local inmath = false
7025
7026   local glue_d = nil
7027   local glue_i = nil
7028
7029   local has_en = false
7030   local first_et = nil
7031
7032   local ATDIR = Babel.attr_dir
7033
7034   local save_outer
7035   local temp = node.get_attribute(head, ATDIR)
7036   if temp then
7037     temp = temp % 3
7038     save_outer = (temp == 0 and 'l') or
7039                  (temp == 1 and 'r') or
7040                  (temp == 2 and 'al')
7041   elseif ispar then      -- Or error? Shouldn't happen
7042     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7043   else                  -- Or error? Shouldn't happen
7044     save_outer = ('TRT' == hdir) and 'r' or 'l'
7045   end
7046   -- when the callback is called, we are just _after_ the box,
7047   -- and the textdir is that of the surrounding text
7048   -- if not ispar and hdir ~= tex.textdir then
7049   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7050   -- end
7051   local outer = save_outer
7052   local last = outer
7053   -- 'al' is only taken into account in the first, current loop
7054   if save_outer == 'al' then save_outer = 'r' end
7055
7056   local fontmap = Babel.fontmap
7057
7058   for item in node.traverse(head) do
7059
7060     -- In what follows, #node is the last (previous) node, because the
7061     -- current one is not added until we start processing the neutrals.
7062
7063     -- three cases: glyph, dir, otherwise
7064     if item.id == GLYPH
7065       or (item.id == 7 and item.subtype == 2) then
7066
7067       local d_font = nil
7068       local item_r
7069       if item.id == 7 and item.subtype == 2 then
7070         item_r = item.replace    -- automatic discs have just 1 glyph
7071       else
7072         item_r = item
7073       end
7074       local chardata = characters[item_r.char]
7075       d = chardata and chardata.d or nil
7076       if not d or d == 'nsm' then
7077         for nn, et in ipairs(ranges) do
7078           if item_r.char < et[1] then
7079             break

```



```

7080         elseif item_r.char <= et[2] then
7081             if not d then d = et[3]
7082             elseif d == 'nsm' then d_font = et[3]
7083             end
7084             break
7085         end
7086     end
7087 end
7088 d = d or 'l'
7089
7090 -- A short 'pause' in bidi for mapfont
7091 d_font = d_font or d
7092 d_font = (d_font == 'l' and 0) or
7093           (d_font == 'nsm' and 0) or
7094           (d_font == 'r' and 1) or
7095           (d_font == 'al' and 2) or
7096           (d_font == 'an' and 2) or nil
7097 if d_font and fontmap and fontmap[d_font][item_r.font] then
7098     item_r.font = fontmap[d_font][item_r.font]
7099 end
7100
7101 if new_d then
7102     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7103     if inmath then
7104         attr_d = 0
7105     else
7106         attr_d = node.get_attribute(item, ATDIR)
7107         attr_d = attr_d % 3
7108     end
7109     if attr_d == 1 then
7110         outer_first = 'r'
7111         last = 'r'
7112     elseif attr_d == 2 then
7113         outer_first = 'r'
7114         last = 'al'
7115     else
7116         outer_first = 'l'
7117         last = 'l'
7118     end
7119     outer = last
7120     has_en = false
7121     first_et = nil
7122     new_d = false
7123 end
7124
7125 if glue_d then
7126     if (d == 'l' and 'l' or 'r') ~= glue_d then
7127         table.insert(nodes, {glue_i, 'on', nil})
7128     end
7129     glue_d = nil
7130     glue_i = nil
7131 end
7132
7133 elseif item.id == DIR then
7134     d = nil
7135     if head ~= item then new_d = true end
7136
7137 elseif item.id == node.id'glue' and item.subtype == 13 then
7138     glue_d = d
7139     glue_i = item
7140     d = nil
7141
7142 elseif item.id == node.id'math' then

```

```

7143     inmath = (item.subtype == 0)
7144
7145 else
7146     d = nil
7147 end
7148
7149 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7150 if last == 'al' and d == 'en' then
7151     d = 'an'           -- W3
7152 elseif last == 'al' and (d == 'et' or d == 'es') then
7153     d = 'on'           -- W6
7154 end
7155
7156 -- EN + CS/ES + EN      -- W4
7157 if d == 'en' and #nodes >= 2 then
7158     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7159         and nodes[#nodes-1][2] == 'en' then
7160         nodes[#nodes][2] = 'en'
7161     end
7162 end
7163
7164 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7165 if d == 'an' and #nodes >= 2 then
7166     if (nodes[#nodes][2] == 'cs')
7167         and nodes[#nodes-1][2] == 'an' then
7168         nodes[#nodes][2] = 'an'
7169     end
7170 end
7171
7172 -- ET/EN                  -- W5 + W7->l / W6->on
7173 if d == 'et' then
7174     first_et = first_et or (#nodes + 1)
7175 elseif d == 'en' then
7176     has_en = true
7177     first_et = first_et or (#nodes + 1)
7178 elseif first_et then      -- d may be nil here !
7179     if has_en then
7180         if last == 'l' then
7181             temp = 'l'    -- W7
7182         else
7183             temp = 'en'   -- W5
7184         end
7185     else
7186         temp = 'on'      -- W6
7187     end
7188     for e = first_et, #nodes do
7189         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7190     end
7191     first_et = nil
7192     has_en = false
7193 end
7194
7195 -- Force mathdir in math if ON (currently works as expected only
7196 -- with 'l')
7197 if inmath and d == 'on' then
7198     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7199 end
7200
7201 if d then
7202     if d == 'al' then
7203         d = 'r'
7204         last = 'al'
7205     elseif d == 'l' or d == 'r' then

```

```

7206         last = d
7207     end
7208     prev_d = d
7209     table.insert(nodes, {item, d, outer_first})
7210 end
7211
7212     outer_first = nil
7213
7214 end
7215
7216 -- TODO -- repeated here in case EN/ET is the last node. Find a
7217 -- better way of doing things:
7218 if first_et then      -- dir may be nil here !
7219     if has_en then
7220         if last == 'l' then
7221             temp = 'l'    -- W7
7222         else
7223             temp = 'en'    -- W5
7224         end
7225     else
7226         temp = 'on'        -- W6
7227     end
7228     for e = first_et, #nodes do
7229         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7230     end
7231 end
7232
7233 -- dummy node, to close things
7234 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7235
7236 ----- NEUTRAL -----
7237
7238 outer = save_outer
7239 last = outer
7240
7241 local first_on = nil
7242
7243 for q = 1, #nodes do
7244     local item
7245
7246     local outer_first = nodes[q][3]
7247     outer = outer_first or outer
7248     last = outer_first or last
7249
7250     local d = nodes[q][2]
7251     if d == 'an' or d == 'en' then d = 'r' end
7252     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7253
7254     if d == 'on' then
7255         first_on = first_on or q
7256     elseif first_on then
7257         if last == d then
7258             temp = d
7259         else
7260             temp = outer
7261         end
7262         for r = first_on, q - 1 do
7263             nodes[r][2] = temp
7264             item = nodes[r][1]    -- MIRRORING
7265             if Babel.mirroring_enabled and item.id == GLYPH
7266                 and temp == 'r' and characters[item.char] then
7267                 local font_mode = ''
7268                 if font.fonts[item.font].properties then

```

```

7269         font_mode = font.fonts[item.font].properties.mode
7270     end
7271     if font_mode ~= 'harf' and font_mode ~= 'plug' then
7272         item.char = characters[item.char].m or item.char
7273     end
7274     end
7275 end
7276 first_on = nil
7277 end
7278
7279 if d == 'r' or d == 'l' then last = d end
7280 end
7281
7282 ----- IMPLICIT, REORDER -----
7283
7284 outer = save_outer
7285 last = outer
7286
7287 local state = {}
7288 state.has_r = false
7289
7290 for q = 1, #nodes do
7291     local item = nodes[q][1]
7292
7293     outer = nodes[q][3] or outer
7294
7295     local d = nodes[q][2]
7296
7297     if d == 'nsm' then d = last end          -- W1
7298     if d == 'en' then d = 'an' end
7299     local isdir = (d == 'r' or d == 'l')
7300
7301     if outer == 'l' and d == 'an' then
7302         state.san = state.san or item
7303         state.ean = item
7304     elseif state.san then
7305         head, state = insert_numeric(head, state)
7306     end
7307
7308     if outer == 'l' then
7309         if d == 'an' or d == 'r' then      -- im -> implicit
7310             if d == 'r' then state.has_r = true end
7311             state.sim = state.sim or item
7312             state.eim = item
7313         elseif d == 'l' and state.sim and state.has_r then
7314             head, state = insert_implicit(head, state, outer)
7315         elseif d == 'l' then
7316             state.sim, state.eim, state.has_r = nil, nil, false
7317         end
7318     else
7319         if d == 'an' or d == 'l' then
7320             if nodes[q][3] then -- nil except after an explicit dir
7321                 state.sim = item -- so we move sim 'inside' the group
7322             else
7323                 state.sim = state.sim or item
7324             end
7325             state.eim = item
7326         elseif d == 'r' and state.sim then
7327             head, state = insert_implicit(head, state, outer)
7328         elseif d == 'r' then
7329             state.sim, state.eim = nil, nil
7330         end
7331     end

```

```

7332     end
7333
7334     if isdir then
7335         last = d          -- Don't search back - best save now
7336     elseif d == 'on' and state.san then
7337         state.san = state.san or item
7338         state.ean = item
7339     end
7340
7341 end
7342
7343 return node.prev(head) or head
7344 end
7345 </basic>

```

13 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.

14 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.

```

7346 <*nil>
7347 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7348 \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.

```

7349 \ifx\l@nil\@undefined
7350   \newlanguage\l@nil
7351   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
7352   \let\bbl@elt\relax
7353   \edef\bbl@languages{% Add it to the list of languages
7354     \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
7355 \fi

```

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

```

7356 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
7357 \let\captionnil\@empty
7358 \let\datenil\@empty

```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```

7359 \def\bbl@inidata@nil{%
7360   \bbl@elt{identification}{tag.ini}{und}%

```

```

7361 \bbl@elt{identification}{load.level}{0}%
7362 \bbl@elt{identification}{charset}{utf8}%
7363 \bbl@elt{identification}{version}{1.0}%
7364 \bbl@elt{identification}{date}{2022-05-16}%
7365 \bbl@elt{identification}{name.local}{nil}%
7366 \bbl@elt{identification}{name.english}{nil}%
7367 \bbl@elt{identification}{name.babel}{nil}%
7368 \bbl@elt{identification}{tag.bcp47}{und}%
7369 \bbl@elt{identification}{language.tag.bcp47}{und}%
7370 \bbl@elt{identification}{tag.opentype}{dflt}%
7371 \bbl@elt{identification}{script.name}{Latin}%
7372 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7373 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7374 \bbl@elt{identification}{level}{1}%
7375 \bbl@elt{identification}{encodings}{}%
7376 \bbl@elt{identification}{derivate}{no}}
7377 \@namedef{bbl@tbc@nil}{und}
7378 \@namedef{bbl@lbc@nil}{und}
7379 \@namedef{bbl@lotf@nil}{dflt}
7380 \@namedef{bbl@elname@nil}{nil}
7381 \@namedef{bbl@lname@nil}{nil}
7382 \@namedef{bbl@esname@nil}{Latin}
7383 \@namedef{bbl@sname@nil}{Latin}
7384 \@namedef{bbl@sbc@nil}{Latn}
7385 \@namedef{bbl@sotf@nil}{Latn}

```

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.

```

7386 \ldf@finish{nil}
7387 \</nil>

```

15 Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the identification section with `require.calendars`.

Start with function to compute the Julian day. It's based on the little library `calendar.js`, by John Walker, in the public domain.

```

7388 <(*Compute Julian day)> ≡
7389 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7390 \def\bbl@cs@gregleap#1{%
7391   (\bbl@fpmo{#1}{4} == 0) &&
7392   (!(\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0))}
7393 \def\bbl@cs@jd#1#2#3{% year, month, day
7394   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7395     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7396     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7397     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }
7398 </Compute Julian day>

```

15.1 Islamic

The code for the Civil calendar is based on it, too.

```

7399 <*ca-islamic>
7400 \ExplSyntaxOn
7401 <(*Compute Julian day)>
7402 % == islamic (default)
7403 % Not yet implemented
7404 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar:

```

7405 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7406   ((#3 + ceil(29.5 * (#2 - 1)) +

```

```

7407  (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7408  1948439.5) - 1) }
7409 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7410 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7411 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7412 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7413 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7414 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7415   \edef\bbl@tempa{%
7416     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7417   \edef#5{%
7418     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7419   \edef#6{\fp_eval:n{
7420     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7421   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}

```

The Umm al-Qura calendar, used mainly in Saudi Arabia, is based on moment-hijri, by Abdullah Alsigar (license MIT).

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ~1435/~1460 (Gregorian ~2014/~2038).

```

7422 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7423  56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7424  57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7425  57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7426  57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7427  58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7428  58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7429  58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7430  58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7431  59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7432  59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7433  59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7434  60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7435  60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7436  60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7437  60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7438  61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7439  61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7440  61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7441  62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7442  62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7443  62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7444  63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7445  63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7446  63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7447  63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7448  64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7449  64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7450  64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7451  65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7452  65401,65431,65460,65490,65520}
7453 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7454 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7455 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7456 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7457   \ifnum#2>2014 \ifnum#2<2038
7458     \bbl@afterfi\expandafter\@gobble
7459   \fi\fi
7460   {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7461   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7462     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7463   \count@\@ne
7464   \bbl@foreach\bbl@cs@umalqura@data{%

```

```

7465 \advance\count@\@ne
7466 \ifnum##1>\bbl@tempd\else
7467 \edef\bbl@tempe{\the\count@}%
7468 \edef\bbl@tempb{##1}%
7469 \fi}%
7470 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7471 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7472 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7473 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7474 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7475 \ExplSyntaxOff
7476 \bbl@add\bbl@precalendar{%
7477 \bbl@replace\bbl@ld@calendar{-civil}{}}%
7478 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
7479 \bbl@replace\bbl@ld@calendar{+}{}}%
7480 \bbl@replace\bbl@ld@calendar{-}{}}%
7481 \</ca-islamic>

```

16 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in `hebcald.sty`

```

7482 \<ca-hebrew>
7483 \newcount\bbl@cntcommon
7484 \def\bbl@remainder#1#2#3{%
7485 #3=#1\relax
7486 \divide #3 by #2\relax
7487 \multiply #3 by -#2\relax
7488 \advance #3 by #1\relax}%
7489 \newif\ifbbl@divisible
7490 \def\bbl@checkifdivisible#1#2{%
7491 {\countdef\tmp=0
7492 \bbl@remainder{#1}{#2}{\tmp}%
7493 \ifnum \tmp=0
7494 \global\bbl@divisibletrue
7495 \else
7496 \global\bbl@divisiblefalse
7497 \fi}}
7498 \newif\ifbbl@gregleap
7499 \def\bbl@ifgregleap#1{%
7500 \bbl@checkifdivisible{#1}{4}%
7501 \ifbbl@divisible
7502 \bbl@checkifdivisible{#1}{100}%
7503 \ifbbl@divisible
7504 \bbl@checkifdivisible{#1}{400}%
7505 \ifbbl@divisible
7506 \bbl@gregleaptrue
7507 \else
7508 \bbl@gregleapfalse
7509 \fi
7510 \else
7511 \bbl@gregleaptrue
7512 \fi
7513 \else
7514 \bbl@gregleapfalse
7515 \fi
7516 \ifbbl@gregleap}
7517 \def\bbl@gregdayspriormonths#1#2#3{%
7518 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7519 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7520 \bbl@ifgregleap{#2}%

```



```

7521         \ifnum #1 > 2
7522             \advance #3 by 1
7523         \fi
7524     \fi
7525     \global\bb1@cntcommon=#3}%
7526     #3=\bb1@cntcommon}
7527 \def\bb1@gregdaysprioryears#1#2{%
7528     {\countdef\tmpc=4
7529     \countdef\tmpb=2
7530     \tmpb=#1\relax
7531     \advance \tmpb by -1
7532     \tmpc=\tmpb
7533     \multiply \tmpc by 365
7534     #2=\tmpc
7535     \tmpc=\tmpb
7536     \divide \tmpc by 4
7537     \advance #2 by \tmpc
7538     \tmpc=\tmpb
7539     \divide \tmpc by 100
7540     \advance #2 by -\tmpc
7541     \tmpc=\tmpb
7542     \divide \tmpc by 400
7543     \advance #2 by \tmpc
7544     \global\bb1@cntcommon=#2\relax}%
7545     #2=\bb1@cntcommon}
7546 \def\bb1@absfromgreg#1#2#3#4{%
7547     {\countdef\tmpd=0
7548     #4=#1\relax
7549     \bb1@gregdayspriormonths{#2}{#3}{\tmpd}%
7550     \advance #4 by \tmpd
7551     \bb1@gregdaysprioryears{#3}{\tmpd}%
7552     \advance #4 by \tmpd
7553     \global\bb1@cntcommon=#4\relax}%
7554     #4=\bb1@cntcommon}
7555 \newif\ifbb1@hebrleap
7556 \def\bb1@checkleaphebryear#1{%
7557     {\countdef\tmpa=0
7558     \countdef\tmpb=1
7559     \tmpa=#1\relax
7560     \multiply \tmpa by 7
7561     \advance \tmpa by 1
7562     \bb1@remainder{\tmpa}{19}{\tmpb}%
7563     \ifnum \tmpb < 7
7564         \global\bb1@hebrleaptrue
7565     \else
7566         \global\bb1@hebrleapfalse
7567     \fi}}
7568 \def\bb1@hebrlapsedmonths#1#2{%
7569     {\countdef\tmpa=0
7570     \countdef\tmpb=1
7571     \countdef\tmpc=2
7572     \tmpa=#1\relax
7573     \advance \tmpa by -1
7574     #2=\tmpa
7575     \divide #2 by 19
7576     \multiply #2 by 235
7577     \bb1@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7578     \tmpc=\tmpb
7579     \multiply \tmpb by 12
7580     \advance #2 by \tmpb
7581     \multiply \tmpc by 7
7582     \advance \tmpc by 1
7583     \divide \tmpc by 19

```

```

7584 \advance #2 by \tmpc
7585 \global\bb1@cntcommon=#2}%
7586 #2=\bb1@cntcommon}
7587 \def\bb1@hebreleapseddays#1#2{%
7588 {\countdef\tmpa=0
7589 \countdef\tmpb=1
7590 \countdef\tmpc=2
7591 \bb1@hebreleapsedmonths{#1}{#2}%
7592 \tmpa=#2\relax
7593 \multiply \tmpa by 13753
7594 \advance \tmpa by 5604
7595 \bb1@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7596 \divide \tmpa by 25920
7597 \multiply #2 by 29
7598 \advance #2 by 1
7599 \advance #2 by \tmpa
7600 \bb1@remainder{#2}{7}{\tmpa}%
7601 \ifnum \tmpc < 19440
7602 \ifnum \tmpc < 9924
7603 \else
7604 \ifnum \tmpa=2
7605 \bb1@checkleaphebrewyear{#1}% of a common year
7606 \ifbb1@hebrleap
7607 \else
7608 \advance #2 by 1
7609 \fi
7610 \fi
7611 \fi
7612 \ifnum \tmpc < 16789
7613 \else
7614 \ifnum \tmpa=1
7615 \advance #1 by -1
7616 \bb1@checkleaphebrewyear{#1}% at the end of leap year
7617 \ifbb1@hebrleap
7618 \advance #2 by 1
7619 \fi
7620 \fi
7621 \fi
7622 \else
7623 \advance #2 by 1
7624 \fi
7625 \bb1@remainder{#2}{7}{\tmpa}%
7626 \ifnum \tmpa=0
7627 \advance #2 by 1
7628 \else
7629 \ifnum \tmpa=3
7630 \advance #2 by 1
7631 \else
7632 \ifnum \tmpa=5
7633 \advance #2 by 1
7634 \fi
7635 \fi
7636 \fi
7637 \global\bb1@cntcommon=#2\relax}%
7638 #2=\bb1@cntcommon}
7639 \def\bb1@daysinhebrewyear#1#2{%
7640 {\countdef\tmpe=12
7641 \bb1@hebreleapseddays{#1}{\tmpe}%
7642 \advance #1 by 1
7643 \bb1@hebreleapseddays{#1}{#2}%
7644 \advance #2 by -\tmpe
7645 \global\bb1@cntcommon=#2}%
7646 #2=\bb1@cntcommon}

```

```

7647 \def\bbl@hebrdayspriormonths#1#2#3{%
7648   {\countdef\tmpf= 14
7649    #3=\ifcase #1\relax
7650      0 \or
7651      0 \or
7652      30 \or
7653      59 \or
7654      89 \or
7655      118 \or
7656      148 \or
7657      148 \or
7658      177 \or
7659      207 \or
7660      236 \or
7661      266 \or
7662      295 \or
7663      325 \or
7664      400
7665   \fi
7666   \bbl@checkleaphebryear{#2}%
7667   \ifbbl@hebrleap
7668     \ifnum #1 > 6
7669       \advance #3 by 30
7670     \fi
7671   \fi
7672   \bbl@daysinhebryear{#2}{\tmpf}%
7673   \ifnum #1 > 3
7674     \ifnum \tmpf=353
7675       \advance #3 by -1
7676     \fi
7677     \ifnum \tmpf=383
7678       \advance #3 by -1
7679     \fi
7680   \fi
7681   \ifnum #1 > 2
7682     \ifnum \tmpf=355
7683       \advance #3 by 1
7684     \fi
7685     \ifnum \tmpf=385
7686       \advance #3 by 1
7687     \fi
7688   \fi
7689   \global\bbl@cntcommon=#3\relax}%
7690   #3=\bbl@cntcommon}
7691 \def\bbl@absfromhebr#1#2#3#4{%
7692   {#4=#1\relax
7693   \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7694   \advance #4 by #1\relax
7695   \bbl@hebreleapseddays{#3}{#1}%
7696   \advance #4 by #1\relax
7697   \advance #4 by -1373429
7698   \global\bbl@cntcommon=#4\relax}%
7699   #4=\bbl@cntcommon}
7700 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7701   {\countdef\tmpx= 17
7702    \countdef\tmpy= 18
7703    \countdef\tmpz= 19
7704    #6=#3\relax
7705    \global\advance #6 by 3761
7706    \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7707    \tmpz=1 \tmpy=1
7708    \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7709    \ifnum \tmpx > #4\relax

```

```

7710      \global\advance #6 by -1
7711      \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7712  \fi
7713  \advance #4 by -\tmpx
7714  \advance #4 by 1
7715  #5=#4\relax
7716  \divide #5 by 30
7717  \loop
7718      \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7719      \ifnum \tmpx < #4\relax
7720          \advance #5 by 1
7721          \tmpy=\tmpx
7722  \repeat
7723  \global\advance #5 by -1
7724  \global\advance #4 by -\tmpy}}
7725 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
7726 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7727 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7728  \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7729  \bbl@hebrfromgreg
7730      {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7731      {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
7732  \edef#4{\the\bbl@hebyear}%
7733  \edef#5{\the\bbl@hebrmonth}%
7734  \edef#6{\the\bbl@hebrday}}
7735 </ca-hebrew>

```

17 Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use with LPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```

7736 <*ca-persian>
7737 \ExplSyntaxOn
7738 <(<Compute Julian day>)
7739 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7740 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7741 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
7742  \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
7743  \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7744      \bbl@afterfi\expandafter\@gobble
7745  \fi\fi
7746      {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}}%
7747  \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7748  \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7749  \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7750  \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7751  \ifnum\bbl@tempc<\bbl@tempb
7752      \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7753      \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7754      \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7755      \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}}%
7756  \fi
7757  \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7758  \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7759  \edef#5{\fp_eval:n{% set Jalali month
7760      (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7761  \edef#6{\fp_eval:n{% set Jalali day
7762      (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}%
7763 \ExplSyntaxOff

```

7764 \langle /ca-persian \rangle

18 Coptic and Ethiopic

Adapted from `jquery.calendars.package-1.1.4`, written by Keith Wood, 2010. Dual license: GPL and MIT. The only difference is the epoch.

```
7765  $\langle$ *ca-coptic $\rangle$ 
7766 \ExplSyntaxOn
7767  $\langle$ Compute Julian day $\rangle$ 
7768 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
7769   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7770   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
7771   \edef#4{\fp_eval:n{%
7772     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7773   \edef\bbl@tempc{\fp_eval:n{%
7774     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7775   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7776   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7777 \ExplSyntaxOff
7778  $\langle$ /ca-coptic $\rangle$ 
7779  $\langle$ *ca-ethiopic $\rangle$ 
7780 \ExplSyntaxOn
7781  $\langle$ Compute Julian day $\rangle$ 
7782 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
7783   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7784   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
7785   \edef#4{\fp_eval:n{%
7786     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7787   \edef\bbl@tempc{\fp_eval:n{%
7788     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
7789   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7790   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7791 \ExplSyntaxOff
7792  $\langle$ /ca-ethiopic $\rangle$ 
```

19 Buddhist

That's very simple.

```
7793  $\langle$ *ca-buddhist $\rangle$ 
7794 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
7795   \edef#4{\number\numexpr#1+543\relax}%
7796   \edef#5{#2}%
7797   \edef#6{#3}%
7798  $\langle$ /ca-buddhist $\rangle$ 
```

20 Support for Plain T_EX (plain.def)

20.1 Not renaming hyphen.tex

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

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

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

The files `bplain.tex` and `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

```

7799 <*bplain | blplain>
7800 \catcode`\{=1 % left brace is begin-group character
7801 \catcode`\}=2 % right brace is end-group character
7802 \catcode`\#=6 % hash mark is macro parameter character

```

```
7803 \openin 0 hyphen.cfg
7804 \ifeof0
7805 \else
7806   \let\input
```

```

7807 \def\input #1 {%
7808     \let\input\@
7809     \a hyphen.cfg
7810     \let\@undefined
7811 }
7812 \fi
7813 </bplain | bplain>

```

```
7814 \bplain\la plain.tex
7815 \blplain\la lplain.tex
```

```

7816 <bplain>\def\fmtname{babel-plain}
7817 <blplain>\def\fmtname{babel-lplain}

```

20.2 Emulating some L^AT_EX features

```

7818 <<{*Emulate LaTeX}> ≡
7819 \def\@empty{}
7820 \def\loadlocalcfg#1{%
7821   \openin0#1.cfg
7822   \ifeof0
7823     \closein0
7824   \else
7825     \closein0
7826     {\immediate\write16{*****}%
7827       \immediate\write16{* Local config file #1.cfg used}%
7828       \immediate\write16{*}%
7829     }
7830     \input #1.cfg\relax
7831   \fi
7832   \@endoflpdf}

```

20.3 General tools

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

```
7833 \long\def\@firstofone#1{#1}
7834 \long\def\@firstoftwo#1#2{#1}
7835 \long\def\@secondoftwo#1#2{#2}
7836 \def\@nnil{\nil}
7837 \def\@gobbletwo#1#2{}
7838 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7839 \def\@star@or@long#1{%
7840   \@ifstar
7841   {\let\l@ngrel@x\relax#1}%
7842   {\let\l@ngrel@x\long#1}}
7843 \let\l@ngrel@x\relax
7844 \def\@car#1#2\@nil{#1}
7845 \def\@cdr#1#2\@nil{#2}
7846 \let\@typeset@protect\relax
7847 \let\protected@edef\edef
7848 \long\def\@gobble#1{}
7849 \edef\@backslashchar{\expandafter\@gobble\string\}
7850 \def\strip@prefix#1>{}
7851 \def\g@addto@macro#1#2{%
7852   \toks@\expandafter{#1#2}%
7853   \xdef#1{\the\toks@}}
7854 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7855 \def\@nameuse#1{\csname #1\endcsname}
7856 \def\@ifundefined#1{%
7857   \expandafter\ifx\csname#1\endcsname\relax
7858     \expandafter\@firstoftwo
7859     \else
7860       \expandafter\@secondoftwo
7861     \fi}
7862 \def\@expandtwoargs#1#2#3{%
7863   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7864 \def\zap@space#1 #2{%
7865   #1%
7866   \ifx#2\@empty\else\expandafter\zap@space\fi
7867   #2}
7868 \let\bbl@trace\@gobble
7869 \def\bbl@error#1#2{%
7870   \begingroup
7871     \newlinechar=`^^J
7872     \def\{^^J(babel) }%
7873     \errhelp{#2}\errmessage{\#1}%
7874   \endgroup}
7875 \def\bbl@warning#1{%
7876   \begingroup
7877     \newlinechar=`^^J
7878     \def\{^^J(babel) }%
7879     \message{\#1}%
7880   \endgroup}
7881 \let\bbl@infowarn\bbl@warning
7882 \def\bbl@info#1{%
7883   \begingroup
7884     \newlinechar=`^^J
7885     \def\{^^J}%
7886     \wlog{#1}%
7887   \endgroup}
```

$\text{\LaTeX} 2_{\epsilon}$ has the command `\onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
7888 \ifx\@preamblecmds\undefined
7889   \def\@preamblecmds{}
```

```

7890 \fi
7891 \def\@onlypreamble#1{%
7892   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7893     \@preamblecmds\do#1}}
7894 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

7895 \def\begindocument{%
7896   \@begindocumenthook
7897   \global\let\@begindocumenthook\@undefined
7898   \def\do##1{\global\let##1\@undefined}%
7899   \@preamblecmds
7900   \global\let\do\noexpand}

7901 \ifx\@begindocumenthook\@undefined
7902   \def\@begindocumenthook{}
7903 \fi
7904 \@onlypreamble\@begindocumenthook
7905 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

7906 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7907 \@onlypreamble\AtEndOfPackage
7908 \def\@endofldf{}
7909 \@onlypreamble\@endofldf
7910 \let\bb1@afterlang\@empty
7911 \chardef\bb1@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.

```

7912 \catcode`\&=\z@
7913 \ifx&\if@files\@undefined
7914   \expandafter\let\csname if@files\expandafter\endcsname
7915     \csname iffalse\endcsname
7916 \fi
7917 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7918 \def\newcommand{\@star@or@long\new@command}
7919 \def\new@command#1{%
7920   \@testopt{\@newcommand#1}0}
7921 \def\@newcommand#1[#2]{%
7922   \@ifnextchar [{\@xargdef#1[#2]}%
7923     {\@argdef#1[#2]}}
7924 \long\def\@argdef#1[#2]#3{%
7925   \@yargdef#1\@ne{#2}{#3}}
7926 \long\def\@xargdef#1[#2][#3]#4{%
7927   \expandafter\def\expandafter#1\expandafter{%
7928     \expandafter\@protected@testopt\expandafter #1%
7929     \csname\string#1\expandafter\endcsname{#3}}}%
7930 \expandafter\@yargdef \csname\string#1\endcsname
7931   \tw@{#2}{#4}}
7932 \long\def\@yargdef#1#2#3{%
7933   \@tempcnta#3\relax
7934   \advance \@tempcnta \@ne
7935   \let\@hash@\relax
7936   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7937   \@tempcntb #2%
7938   \@whilenum\@tempcntb <\@tempcnta
7939   \do{%
7940     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7941     \advance\@tempcntb \@ne}%

```



```

7942 \let\@hash@##%
7943 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7944 \def\providecommand{\@star@or@long\provide@command}
7945 \def\provide@command#1{%
7946 \begingroup
7947 \escapechar\m@ne\def\@gtempa{\string#1}}%
7948 \endgroup
7949 \expandafter\ifundefined\@gtempa
7950 {\def\reserved@a{\new@command#1}}%
7951 {\let\reserved@a\relax
7952 \def\reserved@a{\new@command\reserved@a}}%
7953 \reserved@a}%

7954 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7955 \def\declare@robustcommand#1{%
7956 \edef\reserved@a{\string#1}%
7957 \def\reserved@b{#1}%
7958 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7959 \edef#1{%
7960 \ifx\reserved@a\reserved@b
7961 \noexpand\x@protect
7962 \noexpand#1%
7963 \fi
7964 \noexpand\protect
7965 \expandafter\noexpand\csname
7966 \expandafter\@gobble\string#1 \endcsname
7967 }%
7968 \expandafter\new@command\csname
7969 \expandafter\@gobble\string#1 \endcsname
7970 }
7971 \def\x@protect#1{%
7972 \ifx\protect\@typeset@protect\else
7973 \@x@protect#1%
7974 \fi
7975 }
7976 \catcode\&=\z@ % Trick to hide conditionals
7977 \def\@x@protect#1&#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`.

```

7978 \def\bbl@tempa{\csname newif\endcsname&fin@}
7979 \catcode\&=4
7980 \ifx\in@\@undefined
7981 \def\in@#1#2{%
7982 \def\in@@##1#1##2##3\in@{%
7983 \ifx\in@@##2\in@false\else\in@true\fi}%
7984 \in@@##2#1\in@\in@@}
7985 \else
7986 \let\bbl@tempa\@empty
7987 \fi
7988 \bbl@tempa

```

\LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain \TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

7989 \def\@ifpackagewith#1#2#3#4{#3}

```

The \LaTeX macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```

7990 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\text{\LaTeX} 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```
7991 \ifx\@tempcnta\@undefined
7992   \csname newcount\endcsname\@tempcnta\relax
7993 \fi
7994 \ifx\@tempcntb\@undefined
7995   \csname newcount\endcsname\@tempcntb\relax
7996 \fi
```

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

```
7997 \ifx\bye\@undefined
7998   \advance\count10 by -2\relax
7999 \fi
8000 \ifx\@ifnextchar\@undefined
8001   \def\@ifnextchar#1#2#3{%
8002     \let\reserved@d=#1%
8003     \def\reserved@a{#2}\def\reserved@b{#3}%
8004     \futurelet\@let@token\@ifnch}
8005   \def\@ifnch{%
8006     \ifx\@let@token\@sptoken
8007       \let\reserved@c\@xifnch
8008     \else
8009       \ifx\@let@token\reserved@d
8010         \let\reserved@c\reserved@a
8011       \else
8012         \let\reserved@c\reserved@b
8013     \fi
8014   \fi
8015   \reserved@c}
8016   \def\:\let\@sptoken= } \: % this makes \sptoken a space token
8017   \def\:\@xifnch} \expandafter\def\:\futurelet\@let@token\@ifnch}
8018 \fi
8019 \def\@testopt#1#2{%
8020   \@ifnextchar[#{#1}{#1[#2]}}
8021 \def\@protected@testopt#1{%
8022   \ifx\protect\@typeset@protect
8023     \expandafter\@testopt
8024   \else
8025     \@x@protect#1%
8026   \fi}
8027 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8028   #2\relax}\fi}
8029 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8030   \else\expandafter\@gobble\fi{#1}}
```

20.4 Encoding related macros

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

```
8031 \def\DeclareTextCommand{%
8032   \@dec@text@cmd\providecommand
8033 }
8034 \def\ProvideTextCommand{%
8035   \@dec@text@cmd\providecommand
8036 }
8037 \def\DeclareTextSymbol#1#2#3{%
8038   \@dec@text@cmd\chardef#1{#2}#3\relax
8039 }
8040 \def\@dec@text@cmd#1#2#3{%
8041   \expandafter\def\expandafter#2%
8042     \expandafter{%
```

```

8043         \csname#3-cmd\expandafter\endcsname
8044         \expandafter#2%
8045         \csname#3\string#2\endcsname
8046     }%
8047 % \let\@ifdefinable\@rc@ifdefinable
8048     \expandafter#1\csname#3\string#2\endcsname
8049 }
8050 \def\@current@cmd#1{%
8051     \ifx\protect\@typeset@protect\else
8052         \noexpand#1\expandafter\@gobble
8053     \fi
8054 }
8055 \def\@changed@cmd#1#2{%
8056     \ifx\protect\@typeset@protect
8057         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8058             \expandafter\ifx\csname ?\string#1\endcsname\relax
8059                 \expandafter\def\csname ?\string#1\endcsname{%
8060                     \@changed@x@err{#1}%
8061                 }%
8062             \fi
8063             \global\expandafter\let
8064                 \csname\cf@encoding \string#1\expandafter\endcsname
8065                 \csname ?\string#1\endcsname
8066             \fi
8067             \csname\cf@encoding\string#1%
8068             \expandafter\endcsname
8069         \else
8070             \noexpand#1%
8071         \fi
8072 }
8073 \def\@changed@x@err#1{%
8074     \errhelp{Your command will be ignored, type <return> to proceed}%
8075     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8076 \def\DeclareTextCommandDefault#1{%
8077     \DeclareTextCommand#1?%
8078 }
8079 \def\ProvideTextCommandDefault#1{%
8080     \ProvideTextCommand#1?%
8081 }
8082 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8083 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8084 \def\DeclareTextAccent#1#2#3{%
8085     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8086 }
8087 \def\DeclareTextCompositeCommand#1#2#3#4{%
8088     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8089     \edef\reserved@b{\string##1}%
8090     \edef\reserved@c{%
8091         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8092     \ifx\reserved@b\reserved@c
8093         \expandafter\expandafter\expandafter\ifx
8094             \expandafter\@car\reserved@a\relax\relax\@nil
8095             \@text@composite
8096         \else
8097             \edef\reserved@b##1{%
8098                 \def\expandafter\noexpand
8099                     \csname#2\string#1\endcsname####1{%
8100                         \noexpand\@text@composite
8101                         \expandafter\noexpand\csname#2\string#1\endcsname
8102                         #####1\noexpand\@empty\noexpand\@text@composite
8103                         {##1}%
8104                     }%
8105                 }%

```

```

8106     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8107     \fi
8108     \expandafter\def\csname\expandafter\string\csname
8109       #2\endcsname\string#1-\string#3\endcsname{#4}
8110   \else
8111     \errhelp{Your command will be ignored, type <return> to proceed}%
8112     \errmessage{\string\DeclareTextCompositeCommand\space used on
8113       inappropriate command \protect#1}
8114   \fi
8115 }
8116 \def\@text@composite#1#2#3\@text@composite{%
8117   \expandafter\@text@composite@x
8118     \csname\string#1-\string#2\endcsname
8119 }
8120 \def\@text@composite@x#1#2{%
8121   \ifx#1\relax
8122     #2%
8123   \else
8124     #1%
8125   \fi
8126 }
8127 %
8128 \def\@strip@args#1:#2-#3\@strip@args{#2}
8129 \def\DeclareTextComposite#1#2#3#4{%
8130   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8131   \bgroup
8132     \lccode`\@=#4%
8133     \lowercase{%
8134       \egroup
8135       \reserved@a @%
8136     }%
8137 }
8138 %
8139 \def\UseTextSymbol#1#2{#2}
8140 \def\UseTextAccent#1#2#3{}
8141 \def\@use@text@encoding#1{}
8142 \def\DeclareTextSymbolDefault#1#2{%
8143   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8144 }
8145 \def\DeclareTextAccentDefault#1#2{%
8146   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8147 }
8148 \def\cf@encoding{OT1}

```

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

```

8149 \DeclareTextAccent{"}{OT1}{127}
8150 \DeclareTextAccent{'}{OT1}{19}
8151 \DeclareTextAccent{^}{OT1}{94}
8152 \DeclareTextAccent{\`}{OT1}{18}
8153 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8154 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8155 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8156 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8157 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8158 \DeclareTextSymbol{\i}{OT1}{16}
8159 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```

8160 \ifx\scriptsize\undefined
8161   \let\scriptsize\sevenrm

```

```

8162 \fi
And a few more “dummy” definitions.
8163 \def\language{english}%
8164 \let\bbl@opt@shorthands@nnil
8165 \def\bbl@ifshorthand#1#2#3{#2}%
8166 \let\bbl@language@opts@empty
8167 \ifx\babeloptionstrings\undefined
8168   \let\bbl@opt@strings@nnil
8169 \else
8170   \let\bbl@opt@strings\babeloptionstrings
8171 \fi
8172 \def\BabelStringsDefault{generic}
8173 \def\bbl@tempa{normal}
8174 \ifx\babeloptionmath\bbl@tempa
8175   \def\bbl@mathnormal{\noexpand\textormath}
8176 \fi
8177 \def\AfterBabelLanguage#1#2{}
8178 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
8179 \let\bbl@afterlang\relax
8180 \def\bbl@opt@safe{BR}
8181 \ifx\@uclclist\undefined\let\@uclclist\empty\fi
8182 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
8183 \expandafter\newif\csname ifbbl@single\endcsname
8184 \chardef\bbl@bidimode\z@
8185 <</Emulate LaTeX>>
A proxy file:
8186 <*plain>
8187 \input babel.def
8188 </plain>

```

21 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 \TeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O’Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O’Reilly, 2006.
- [6] Leslie Lamport, *\TeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O’Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International \TeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij (s-Gravenhage, 1988).