

Babel

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Version 3.86.08068
2023/03/21

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	11
1.10	Shorthands	12
1.11	Package options	15
1.12	The base option	17
1.13	ini files	18
1.14	Selecting fonts	25
1.15	Modifying a language	27
1.16	Creating a language	28
1.17	Digits and counters	32
1.18	Dates	33
1.19	Accessing language info	34
1.20	Hyphenation and line breaking	35
1.21	Transforms	37
1.22	Selection based on BCP 47 tags	41
1.23	Selecting scripts	42
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	48
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	50
1.31	Current and future work	51
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	52
3	The interface between the core of babel and the language definition files	53
3.1	Guidelines for contributed languages	54
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	58
3.9	Executing code based on the selector	61
II	Source code	61
4	Identification and loading of required files	61
5	locale directory	62

6	Tools	62
6.1	Multiple languages	66
6.2	The Package File (\LaTeX , babel.sty)	67
6.3	base	68
6.4	key=value options and other general option	68
6.5	Conditional loading of shorthands	70
6.6	Interlude for Plain	71
7	Multiple languages	72
7.1	Selecting the language	74
7.2	Errors	82
7.3	Hooks	84
7.4	Setting up language files	86
7.5	Shorthands	88
7.6	Language attributes	97
7.7	Support for saving macro definitions	98
7.8	Short tags	100
7.9	Hyphens	100
7.10	Multiencoding strings	102
7.11	Macros common to a number of languages	108
7.12	Making glyphs available	108
	7.12.1 Quotation marks	108
	7.12.2 Letters	109
	7.12.3 Shorthands for quotation marks	110
	7.12.4 Umlauts and tremas	111
7.13	Layout	112
7.14	Load engine specific macros	113
7.15	Creating and modifying languages	113
8	Adjusting the Babel behavior	135
8.1	Cross referencing macros	137
8.2	Marks	140
8.3	Preventing clashes with other packages	141
	8.3.1 ifthen	141
	8.3.2 varioref	141
	8.3.3 hhline	142
8.4	Encoding and fonts	142
8.5	Basic bidi support	144
8.6	Local Language Configuration	147
8.7	Language options	147
9	The kernel of Babel (babel.def, common)	151
10	Loading hyphenation patterns	151
11	Font handling with fontspec	155
12	Hooks for XeTeX and LuaTeX	158
12.1	XeTeX	158
12.2	Layout	160
12.3	8-bit TeX	162
12.4	LuaTeX	163
12.5	Southeast Asian scripts	169
12.6	CJK line breaking	170
12.7	Arabic justification	172
12.8	Common stuff	176
12.9	Automatic fonts and ids switching	176
12.10	Bidi	182
12.11	Layout	184

12.12	Lua: transforms	191
12.13	Lua: Auto bidi with basic and basic-r	199
13	Data for CJK	210
14	The ‘nil’ language	210
15	Calendars	211
15.1	Islamic	211
16	Hebrew	213
17	Persian	217
18	Coptic and Ethiopic	217
19	Buddhist	218
20	Support for Plain T_EX (plain.def)	218
20.1	Not renaming hyphen.tex	218
20.2	Emulating some L ^A T _E X features	219
20.3	General tools	219
20.4	Encoding related macros	223
21	Acknowledgements	226

Troubleshootoing

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’	9
Argument of \language@active@arg” has an extra }	12
Package babel Info: The following fonts are not babel standard families	27

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, TeXLive, etc.) for further info about how to configure it.

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

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

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

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

NOTE Babel does not make any readjustments by default in font size, vertical positioning or line height by default. This is on purpose because the optimal solution depends on the document layout and the font, and very likely the most appropriate one is a combination of these settings.

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

You can also set the main language explicitly, but it is discouraged except if there 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 xet_{ex} and lua_{tex}, 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:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

New 3.84 With `pdfTeX`, when a language is loaded on the fly (actually, with `\babelprovide`) selectors now set the font encoding based on the list provided when loading `fontenc`. Not all scripts have an associated encoding, so this feature works only with Latin, Cyrillic, Greek, Arabic, Hebrew, Cherokee, Armenian, and Georgian, provided a suitable font is found.

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}.
```

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.

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

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

The most frequent reason is, by far, the latest (for example, you included spanish, but you realized this language is not used after all, and therefore you removed it from the option list). In most cases, the error vanishes when the document is typeset again, but in more severe ones you will need to remove the aux file.

1.6 Plain

In e-Plain and pdf-Plain, load languages styles with `\input` and then use `\begindocument` (the latter is defined by babel):

```
\input estonian.sty
\begindocument
```

WARNING Not all languages provide a sty file and some of them are not compatible with those formats. Please, refer to [Using babel with Plain](#) for further details.

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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.

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

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 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 `bidir` 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}`).

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with the option `fontenc`.⁴ A couple of examples:

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

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

1.10 Shorthands

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

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

only work on ‘known’ shorthand characters, and an error will be raised otherwise. You can check if a character is a shorthand with `\ifbabelshorthand` (see below).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).⁵ Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by `ngerman` with

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

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

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

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

`\babelshorthand` $\{\langle shorthand \rangle\}$

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

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

```
\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 " ^ =

⁵Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.

⁶Thanks to Enrico Gregorio

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 {<character>}{<true>}{<false>}

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

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

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

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

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

safe= none | ref | bib

Some L^AT_EX 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 ϵ T_EX 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 spoiled by an unexpected `.cfg` file. However, if the key **config** is set, this file is loaded.

showlanguages Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.

nocase **New 3.9l** Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.

silent **New 3.9l** No warnings and no *infos* are written to the log file.⁸

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;

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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

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 `{<option-name>}{<code>}`

This command is currently the only provided by `base`. Executes `<code>` 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 `<option-name>` 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 `TeX`, 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.

¹¹Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either `xetex` or `luatex` change this behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

1.13 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 T_EX 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 not 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{სამშარეული და სუფრის ტრადიციები}

ქართული ტრადიციული სამშარეული 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 lualatex also applies here. Some quick patterns can help, with something similar to:

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

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

Latin, Greek, Cyrillic Combining chars with the default luatex font renderer might be wrong; on the other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenation points are discarded (this bug 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}	ar-IQ	Arabic ^u
agq	Aghem	ar-JO	Arabic ^u
ak	Akan	ar-LB	Arabic ^u
am	Amharic ^{ul}	ar-MA	Arabic ^u
ar-DZ	Arabic ^u	ar-PS	Arabic ^u
ar-EG	Arabic ^u	ar-SA	Arabic ^u

ar-SY	Arabic ^u	en-NZ	English ^{ul}
ar-TN	Arabic ^u	en-US	American English ^{ul}
ar	Arabic ^u	en	English ^{ul}
as	Assamese ^u	eo	Esperanto ^{ul}
asa	Asu	es-MX	Mexican Spanish ^{ul}
ast	Asturian ^{ul}	es	Spanish ^{ul}
az-Cyrl	Azerbaijani	et	Estonian ^{ul}
az-Latn	Azerbaijani	eu	Basque ^{ul}
az	Azerbaijani ^{ul}	ewo	Ewondo
bas	Basaa	fa	Persian ^u
be	Belarusian ^{ul}	ff	Fulah
bem	Bemba	fi	Finnish ^{ul}
bez	Bena	fil	Filipino
bg	Bulgarian ^{ul}	fo	Faroese
bm	Bambara	fr-BE	French ^{ul}
bn	Bangla ^u	fr-CA	Canadian French ^{ul}
bo	Tibetan ^u	fr-CH	Swiss French ^{ul}
br	Breton ^{ul}	fr-LU	French ^{ul}
brx	Bodo	fr	French ^{ul}
bs-Cyrl	Bosnian	fur	Friulian ^{ul}
bs-Latn	Bosnian ^{ul}	fy	Western Frisian
bs	Bosnian ^{ul}	ga	Irish ^{ul}
ca	Catalan ^{ul}	gd	Scottish Gaelic ^{ul}
ce	Chechen	gl	Galician ^{ul}
cgg	Chiga	grc	Ancient Greek ^{ul}
chr	Cherokee	gsw	Swiss German
ckb-Arab	Central Kurdish ^u	gu	Gujarati
ckb-Latn	Central Kurdish ^u	guz	Gusii
ckb	Central Kurdish ^u	gv	Manx
cop	Coptic	ha-GH	Hausa
cs	Czech ^{ul}	ha-NE	Hausa
cu-Cyrs	Church Slavic ^u	ha	Hausa ^{ul}
cu-Glag	Church Slavic	haw	Hawaiian
cu	Church Slavic ^u	he	Hebrew ^{ul}
cy	Welsh ^{ul}	hi	Hindi ^u
da	Danish ^{ul}	hr	Croatian ^{ul}
dav	Taita	hsb	Upper Sorbian ^{ul}
de-1901	German ^{ul}	hu	Hungarian ^{ul}
de-1996	German ^{ul}	hy	Armenian ^{ul}
de-AT-1901	Austrian German ^{ul}	ia	Interlingua ^{ul}
de-AT-1996	Austrian German ^{ul}	id	Indonesian ^{ul}
de-AT	Austrian German ^{ul}	ig	Igbo
de-CH-1901	Swiss High German ^{ul}	ii	Sichuan Yi
de-CH-1996	Swiss High 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 ^u
dyo	Jola-Fonyi	kab	Kabyle
dz	Dzongkha	kam	Kamba
ebu	Embu	kde	Makonde
ee	Ewe	kea	Kabuverdianu
el-polyton	Polytonic Greek ^{ul}	kgp	Kaingang
el	Greek ^{ul}	khq	Koyra Chiini
en-AU	Australian English ^{ul}	ki	Kikuyu
en-CA	Canadian English ^{ul}	kk	Kazakh
en-GB	British English ^{ul}	kkj	Kako

kl	Kalaallisut	nus	Nuer
kln	Kalenjin	nyn	Nyankole
km	Khmer ^u	oc	Occitan ^{ul}
kmr-Arab	Northern Kurdish ^u	om	Oromo
kmr-Latn	Northern Kurdish ^{ul}	or	Odia
kmr	Northern Kurdish ^{ul}	os	Ossetic
kn	Kannada ^u	pa-Arab	Punjabi
ko-Hani	Korean ^u	pa-Guru	Punjabi ^u
ko	Korean ^u	pa	Punjabi ^u
kok	Konkani	pl	Polish ^{ul}
ks	Kashmiri	pms	Piedmontese ^{ul}
ksb	Shambala	ps	Pashto
ksf	Bafia	pt-BR	Brazilian Portuguese ^{ul}
ksh	Colognian	pt-PT	European Portuguese ^{ul}
kw	Cornish	pt	Portuguese ^{ul}
ky	Kyrgyz	qu	Quechua
la-x-classic	Classic Latin ^{ul}	rm	Romansh ^{ul}
la-x-ecclesia	Ecclesiastic Latin ^{ul}	rn	Rundi
la-x-medieval	Medieval Latin ^{ul}	ro-MD	Moldavian ^{ul}
la	Latin ^{ul}	ro	Romanian ^{ul}
lag	Langi	rof	Rombo
lb	Luxembourgish ^{ul}	ru	Russian ^{ul}
lg	Ganda	rw	Kinyarwanda
lkt	Lakota	rwk	Rwa
ln	Lingala	sa-Beng	Sanskrit
lo	Lao ^u	sa-Deva	Sanskrit
lrc	Northern Luri	sa-Gujr	Sanskrit
lt	Lithuanian ^{ulll}	sa-Knda	Sanskrit
lu	Luba-Katanga	sa-Mlym	Sanskrit
luo	Luo	sa-Telu	Sanskrit
luy	Luyia	sa	Sanskrit
lv	Latvian ^{ul}	sah	Sakha
mas	Masai	saq	Samburu
mer	Meru	sbp	Sangu
mfe	Morisyen	sc	Sardinian
mg	Malagasy	se	Northern Sami ^{ul}
mgh	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian ^{ul}	sg	Sango
ml	Malayalam ^u	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi ^u	shi	Tachelhit
ms-BN	Malay	si	Sinhala ^u
ms-SG	Malay	sk	Slovak ^{ul}
ms	Malay ^{ul}	sl	Slovenian ^{ul}
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian ^{ul}
naq	Nama	sr-Cyrl-BA	Serbian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Cyrl-ME	Serbian ^{ul}
nd	North Ndebele	sr-Cyrl-XK	Serbian ^{ul}
ne	Nepali	sr-Cyrl	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Latn-BA	Serbian ^{ul}
nmg	Kwasio	sr-Latn-ME	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sr-Latn-XK	Serbian ^{ul}
nnh	Ngiemboon	sr-Latn	Serbian ^{ul}
no	Norwegian ^{ul}	sr	Serbian ^{ul}

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

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

afrikaans	basaa
aghem	basque
akan	belarusian
albanian	bemba
american	bena
amharic	bangla
ancientgreek	bodo
arabic	bosnian-cyrillic
arabic-algeria	bosnian-cyrl
arabic-DZ	bosnian-latin
arabic-morocco	bosnian-latn
arabic-MA	bosnian
arabic-syria	brazilian
arabic-SY	breton
armenian	british
assamese	bulgarian
asturian	burmese
asu	canadian
australian	cantonese
austrian	catalan
azerbaijani-cyrillic	centralatlastamazight
azerbaijani-cyrl	centralkurdish
azerbaijani-latin	chechen
azerbaijani-latn	cherokee
azerbaijani	chiga
bafia	chinese-hans-hk
bambara	chinese-hans-mo

chinese-hans-sg	galician
chinese-hans	ganda
chinese-hant-hk	georgian
chinese-hant-mo	german-at
chinese-hant	german-austria
chinese-simplified-hongkongsarchina	german-ch
chinese-simplified-macausarchina	german-switzerland
chinese-simplified-singapore	german
chinese-simplified	greek
chinese-traditional-hongkongsarchina	gujarati
chinese-traditional-macausarchina	gusii
chinese-traditional	hausa-gh
chinese	hausa-ghana
churchslavic	hausa-ne
churchslavic-cyrs	hausa-niger
churchslavic-oldcyrillic ¹²	hausa
churchsslavic-glag	hawaiian
churchsslavic-glagolitic	hebrew
cognian	hindi
cornish	hungarian
croatian	icelandic
czech	igbo
danish	inarisami
duala	indonesian
dutch	interlingua
dzongkha	irish
embu	italian
english-au	japanese
english-australia	jolafonyi
english-ca	kabuverdianu
english-canada	kabyle
english-gb	kako
english-newzealand	kalaallisut
english-nz	kalenjin
english-unitedkingdom	kamba
english-unitedstates	kannada
english-us	kashmiri
english	kazakh
esperanto	khmer
estonian	kikuyu
ewe	kinyarwanda
ewondo	konkani
faroeese	korean
filipino	koyraborosenni
finnish	koyrachiini
french-be	kwasio
french-belgium	kyrgyz
french-ca	lakota
french-canada	langi
french-ch	lao
french-lu	latvian
french-luxembourg	lingala
french-switzerland	lithuanian
french	lowersorbian
friulian	lsorbian
fulah	lubakatanga

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

luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru

punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight

swahili	uyghur
swedish	uzbek-arab
swissgerman	uzbek-arabic
tachelhit-latin	uzbek-cyrillic
tachelhit-latn	uzbek-cyrl
tachelhit-tfng	uzbek-latin
tachelhit-tifinagh	uzbek-latn
tachelhit	uzbek
taita	vai-latin
tamil	vai-latn
tasawaq	vai-vai
telugu	vai-vaii
teso	vai
thai	vietnam
tibetan	vietnamese
tigrinya	vunjo
tongan	walser
turkish	welsh
turkmen	westernfrisian
ukenglish	yangben
ukrainian	yiddish
uppersorbian	yoruba
urdu	zarma
usenglish	zulu
usorbian	

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

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

many fonts as you want ‘just in case’, because if the language is never selected, the corresponding `\babelfont` declaration is just ignored. Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

NOTE You may load fontspec explicitly. For example:

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

NOTE There are a few alternative methods:

- With data 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, and that is often the recommended option. 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 is best written as:

```
\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= \langle language-list \rangle

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

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

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

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

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

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

New 3.58 Another special value is `unhyphenated`, which is an alternative to `justification=unhyphenated`.

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.polutonico]{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= \langle script-name \rangle

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

language= \langle language-name \rangle

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

alph= \langle counter-name \rangle

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

Alph= $\langle counter-name \rangle$

Same for \Alph.

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

onchar= ids | fonts | letters

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`). Characters can be added or modified with `\babelcharproperty`.

New 3.81 Option `letters` restricts the ‘actions’ to letters, in the T_EX sense (i. e., with `catcode 11`). Digits and punctuation are then considered part of current locale (as set by a selector). This option is useful when the main script is non-Latin and there is a secondary one whose script is Latin.

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.

NOTE There is no general rule to set the font for a punctuation mark, because it is a semantic decision and not a typographical one. Consider the following sentence: “یک, دو, and سه are Persian numbers”. In this case the punctuation font must be the English one, even if the commas are surrounded by non-Latin letters. Quotation marks, parenthesis, etc., are even more complex. Several criteria are possible, like the main language (the default in babel), the first letter in the paragraph, or the surrounding letters, among others, but even so manual switching can be still necessary.

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= unhyphenated | kashida | elongated | padding

New 3.59 There are currently 4 options. Note they are language dependent, so that they will not be applied to other languages. The first one (unhyphenated) activates a line breaking mode that allows spaces to be stretched to arbitrary amounts. Although for European standards the result may look odd, in some writing systems, like Malayalam and other Indic scripts, this has been the customary (although not always the desired) practice. Because of that, no locale sets currently this mode by default (Amharic is an exception). Unlike `\sloppy`, the `\hfuzz` and the `\vfuzz` are not changed, because this line breaking mode is not really ‘sloppy’ (in other words, overfull boxes are reported as usual).

The second and the third are 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](#).

New 3.81 The option padding has been devised primarily for Tibetan. It’s still somewhat experimental. Again, there is an explanation in the [babel site](#).

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

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 T_EX code). This means the local digits have the correct bidirectional behavior (unlike `Numbers=Arabic` in fontspec, which is not recommended).

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

`\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):

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

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

The styles are:

Ancient Greek `lower.ancient, upper.ancient`
Amharic `afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebona, 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, `\localdate` 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	ethiopic	islamic-civil	persian
coptic	hebrew	islamic-umalqura	

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 TeX sense, as a set of hyphenation patterns, and *not* as its babel name. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is true or false respectively.

`\localeinfo` *{*<field>*}

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

`name.english` as provided by the Unicode CLDR.
`tag.ini` is the tag of the ini file (the way this file is identified in its name).
`tag.bcp47` is the full BCP 47 tag (see the warning below). 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 \getlocaleproperty*, 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 \babel font 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 a 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 \TeX are entered as `-`, and (2) *optional* or *soft hyphens*, which are entered as `\-`. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in \TeX terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity. In \TeX , `-` and `\-` forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, `-` in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine `\-`, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic “hyphens” which can be used by themselves, to define a user shorthand, or even in language files.

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

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

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

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

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:

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.

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

```
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.
Greek	<code>sigma.final</code>	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms 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: !?;.
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zzs</i> as <i>cs-cs, 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, Odia, Tamil, Telugu.
Latin	<code>digraphs.ligatures</code>	Replaces the groups <i>ae, AE, oe, OE</i> with <i>æ, Æ, œ, Œ</i> .

Latin	letters.noj	Replaces <i>j, J</i> with <i>i, I</i> .
Latin	letters.uv	Replaces <i>v, U</i> with <i>u, V</i> .
Sanskrit	transliteration.iast	The IAST system to romanize Devanagari. ¹⁶
Serbian	transliteration.gajica	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	kashida.plain	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.

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

New 3.37-3.39 With *luatex* it is possible to define non-standard hyphenation rules, like $f-f \rightarrow ff-f$, repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. A few rules are currently provided (see above), but they can be defined as shown in the following example, where {1} is the first captured char (between () in the pattern):

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

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

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

New 3.85 Another option is `label`, which takes a value similar to those in `\babelprovide` key transforms (in fact, the latter just applies this option). This label can be used to turn on and off transforms with a higher level interface, by means of `\enablelocaletransform` and `\disablelocaletransform` (see below).

New 3.85 When used in conjunction with `label`, this key makes a transform font dependent. As an example, the rules for Arabic kashida can differ depending on the font design. The value consists in a list of space-separated font tags:

```
\babelprehyphenation[label=transform.name, fonts=rm sf]{...}{...}
```

Tags can adopt two forms: a family, such as `rm` or `tt`, or the set family/series/shape. If a font matches one of these conditions, the transform is enabled. The second tag in `rm rm/n/it` is redundant. There are no wildcards; so, for italics you may want to write something like `sf/m/it sf/b/it`.

Transforms set for specific fonts (at least once in any language) are always reset with a font selector.

In `\babelprovide`, transform labels can be tagged before its name, with a list separated with colons, like:

```
transforms = rm:sf:transform.name
```

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.

`\enablelocaletransform` {*label*}

`\disablelocaletransform` {*label*}

New 3.85 Enables and disables the transform with the given label in the current language.

1.22 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

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

`autoload.bcp47` with values on and off.

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

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

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

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

(You can deactivate it with `off`.) So, if `dutch` is one of the package (or class) options, you can write `\selectlanguage{nl}`. Note the language name does not change (in this

example is still dutch), but you can get it with `\localeinfo` or `\getlocaleproperty`. 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` $\langle\text{text}\rangle$

New 3.9i This macro makes sure $\langle\text{text}\rangle$ 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:

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.

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

```
\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the `bidi` package, which provides its own mechanism to control these elements). You may use several options with a space-separated list, like `layout=counters contents sectioning` (in **New 3.85** spaces are to be preferred over dots, which was the former syntax). 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.

New 3.84 Since `\thepage` is (indirectly) redefined, `makeindex` will reject many entries as invalid. With `counters*` `babel` attempts to remove the conflicting macros.

lists required in `xetex` and `pdftex`, but only in bidirectional (with both R and L paragraphs) documents in `luatex`.

WARNING As of April 2019 there is a bug with `\parshape` in `luatex` (a `TEX` primitive) which makes lists to be horizontally misplaced if they are inside a `\vbox` (like `minipage`) and the current direction is different from the main one. A workaround is to restore the main language before the box and then set the local one inside.

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

\localerestoredirs

New 3.86 *LuaTeX*. This command resets the internal text, paragraph and body directions to those of the current locale (if different). Sometimes changing directly these values can be useful for some hacks, and this command helps in restoring the directions to the correct ones. It can be used in `>` arguments of `array`, too.

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

Mainly for bidi text, but it can be useful in other cases. `\BabelPatchSection` and the corresponding option `layout=sectioning` takes a more logical approach (at least in many cases) because it applies the global language to the section format (including the `\chaptername` in `\chapter`), while the section text is still the current language. The latter is passed to `tocs` and `marks`, too, and with `sectioning` in `layout` they both reset the “global” language to the main one, while the text uses the “local” language.

With `layout=sectioning` all the standard sectioning commands are redefined (it also “isolates” the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

`\BabelFootnote` $\langle cmd \rangle \langle local-language \rangle \langle before \rangle \langle after \rangle$

New 3.17 Something like:

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

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

```
\footnote{(\foreignlanguage{\language}\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}{.}{.}
```

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

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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) ¹⁹
Romanian	romanian
Russian	russian
Scottish Gaelic	scottish
Spanish	spanish

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

Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

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

```

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

```

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

1.28 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).

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

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

```

\babelcharproperty{\_}{mirror}{\_}
\babelcharproperty{\_}{direction}{1} % or al, r, en, an, on, et, cs
\babelcharproperty{\_}{linebreak}{cl} % or id, op, cl, ns, ex, in, hy

```

Please, refer to the Unicode standard (Annex #9 and Annex #14) for the meaning of the available codes. For example, en is ‘European number’ and id is ‘ideographic’.

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

```

\babelcharproperty{\_,}{locale}{english}

```

1.29 Tweaking some features

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

New 3.36 Sometimes you might need to disable some babel features. Currently this macro understands the following keys [to be documented], with values on or off:

<code>bidi.mirroring</code>	<code>linebreak.cjk</code>	<code>autoload.bcp47</code>
<code>bidi.text</code>	<code>justify.arabic</code>	<code>bcp47.toname</code>
<code>bidi.math</code>	<code>layout.tabular</code>	
<code>linebreak.sea</code>	<code>layout.lists</code>	

Other keys [to be documented] are:

<code>autoload.options</code>	<code>autoload.bcp47.options</code>	<code>select.write</code>
<code>autoload.bcp47.prefix</code>	<code>prehyphenation.disable</code>	<code>select.encoding</code>

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\extrasrussian{\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.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesshorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is 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.)

²⁰This 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.

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

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

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

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

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

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

pdf_{La}TeX). babel provides a tool which has become standard in many distributions and based on a “configuration file” named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

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

2.1 Format

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

The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct 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.²⁵ For example:

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

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

A typical error when using babel is the following:

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

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

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

²³The loader for lua(e)tex is slightly different as it's not based on babel but on `etex.src`. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the babel way, i.e., with `language.dat`.

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\langle lang \rangle hyphenmins`, `\captions\langle lang \rangle`, `\date\langle lang \rangle`, `\extras\langle lang \rangle` and `\noextras\langle lang \rangle` (the last two may be left empty); where `\langle lang \rangle` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date\langle lang \rangle` but not `\captions\langle lang \rangle` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@\langle lang \rangle` to be a dialect of `\language0` when `\l@\langle lang \rangle` 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, `\babel@save` and `\babel@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁶

²⁶But not removed, for backward compatibility.

- Please, for “private” internal macros do not use the `\bb1@` 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.

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

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

3.3 Skeleton

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

```

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

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

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

```



```

\bb1@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.
- `\bb1@activate` The command `\bb1@activate` is used to change the way an active character expands.
- `\bb1@deactivate` `\bb1@activate` ‘switches on’ the active behavior of the character. `\bb1@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`. \LaTeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special⟨char⟩` and `\bbl@remove@special⟨char⟩` add and remove the character `⟨char⟩` to these two sets.

`\@safe@activetrue` Enables and disables the “safe” mode. It is a tool for package and class authors. See the

`\@safe@activetrue` description below.

3.5 Support for saving macro definitions

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

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

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

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

3.6 Support for extending macros

`\addto` The macro `\addto{⟨control sequence⟩}{⟨ \TeX code⟩}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`. Be careful when using this macro, because depending on the case the assignment 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

`\bbl@nonfrenchspacing` properly switch French spacing on and off.

²⁷This mechanism was introduced by Bernd Raichle.

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

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

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

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

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

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

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

The $\langle\textit{category}\rangle$ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.²⁸ It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

²⁸In future releases further categories may be added.

```
\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}
\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.²⁹

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands `{\langle code \rangle}`

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

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

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would typically be things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A *<map-list>* is a series of macros using the internal format of `\@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=`İ\relax
   \uccode`ı=`I\relax}
  {\lccode`İ=`i\relax
   \lccode`I=`ı\relax}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

- `\BabelLower{<uccode>}{<lccode>}` is similar to `\lccode` but it's ignored if the char has been set and saves the original `\lccode` to restore it when switching the language (except with `hyphenmap=first`).

- `\BabelLowerMM{⟨uccode-from⟩}{⟨uccode-to⟩}{⟨step⟩}{⟨lccode-from⟩}` loops through the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{⟨uccode-from⟩}{⟨uccode-to⟩}{⟨step⟩}{⟨lccode⟩}` loops through the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for *many-to-one*).

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

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

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

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 sets options and loads language styles.

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

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

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

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

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter.

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

6 Tools

```
1 <<version=3.86.08068>>
2 <<date=2023/03/21>>
```

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@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
16 \def\bbl@cs#1{\csname bbl@#1\endcsname}
17 \def\bbl@c1#1{\csname bbl@#1\language\endcsname}
18 \def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
19 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
20 \def\bbl@@loop#1#2#3,%
21   \ifx\@nnil#3\relax\else
22     \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
23   \fi
```

```

23 \fi}
24 \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.

25 \def\bbl@add@list#1#2{%
26   \edef#1{%
27     \bbl@ifunset{\bbl@stripslash#1}%
28     }%
29     {\ifx#1\@empty\else#1,\fi}%
30     #2}}

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

31 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
32 \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
one-level expansion (where ... is the macro name without the backslash). The result may be
followed by extra arguments, if necessary.

33 \def\bbl@exp#1{%
34   \begingroup
35   \let\>\noexpand
36   \let\<\bbl@exp@en
37   \let\[\bbl@exp@ue
38   \edef\bbl@exp@aux{\endgroup#1}%
39   \bbl@exp@aux}
40 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
41 \def\bbl@exp@ue#1][{%
42   \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.

43 \def\bbl@tempa#1{%
44   \long\def\bbl@trim##1##2{%
45     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46   \def\bbl@trim@c{%
47     \ifx\bbl@trim@a\@sptoken
48       \expandafter\bbl@trim@b
49     \else
50       \expandafter\bbl@trim@b\expandafter#1%
51     \fi}%
52   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
53 \bbl@tempa{ }
54 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \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  $\epsilon$ -tex engine, it is based on \ifcsname, which is more efficient, and does not waste
memory. Defined inside a group, to avoid \ifcsname being implicitly set to \relax by the \csname
test.

56 \begingroup
57   \gdef\bbl@ifunset#1{%
58     \expandafter\ifx\csname#1\endcsname\relax
59     \expandafter\@firstoftwo

```

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


```

60 \else
61 \expandafter\@secondoftwo
62 \fi}
63 \bbl@ifunset{ifcsname}%
64 {}%
65 {\gdef\bbl@ifunset#1{%
66 \ifcsname#1\endcsname
67 \expandafter\ifx\csname#1\endcsname\relax
68 \bbl@afterelse\expandafter\@firstoftwo
69 \else
70 \bbl@afterfi\expandafter\@secondoftwo
71 \fi
72 \else
73 \expandafter\@firstoftwo
74 \fi}}
75 \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,

```

76 \def\bbl@ifblank#1{%
77 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
79 \def\bbl@ifset#1#2#3{%
80 \bbl@ifunset{#1}{#3}{\bbl@exp{\@nameuse{#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).

```

81 \def\bbl@forkv#1#2{%
82 \def\bbl@kvcmd##1##2##3{#2}%
83 \bbl@kvnext#1,\@nil,}
84 \def\bbl@kvnext#1,{%
85 \ifx\@nil#1\relax\else
86 \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
87 \expandafter\bbl@kvnext
88 \fi}
89 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
90 \bbl@trim\def\bbl@forkv@a{#1}%
91 \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).

```

92 \def\bbl@vforeach#1#2{%
93 \def\bbl@forcmd##1{#2}%
94 \bbl@fornext#1,\@nil,}
95 \def\bbl@fornext#1,{%
96 \ifx\@nil#1\relax\else
97 \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
98 \expandafter\bbl@fornext
99 \fi}
100 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

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

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
102 \toks@{}}%
103 \def\bbl@replace@aux##1#2##2#2{%
104 \ifx\bbl@nil##2%
105 \toks@\expandafter{\the\toks@##1}%
106 \else
107 \toks@\expandafter{\the\toks@##1#3}%
108 \bbl@afterfi
109 \bbl@replace@aux##2#2%
110 \fi}%

```

```

111 \expandafter\bb1@replace@aux#1#2\bb1@nil#2%
112 \edef#1{\the\toks@}

```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in \bb1@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bb1@replace; I'm not sure ckecking the replacement is really necessary or just paranoia).

```

113 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
114 \bb1@exp{\def\\bb1@parsedef##1\detokenize{macro:}}#2->#3\relax{%
115   \def\bb1@tempa{#1}%
116   \def\bb1@tempb{#2}%
117   \def\bb1@tempe{#3}}
118 \def\bb1@sreplace#1#2#3{%
119   \begingroup
120     \expandafter\bb1@parsedef\meaning#1\relax
121     \def\bb1@tempc{#2}%
122     \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
123     \def\bb1@tempd{#3}%
124     \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
125     \bb1@xin@\bb1@tempc\bb1@tempe}% If not in macro, do nothing
126     \ifin@
127       \bb1@exp{\\bb1@replace\\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
128       \def\bb1@tempc{% Expanded an executed below as 'uplevel'
129         \\makeatletter % "internal" macros with @ are assumed
130         \\scantokens{%
131           \bb1@tempa\\@namedef{\bb1@stripslash#1}\bb1@tempb{\bb1@tempe}}%
132         \catcode64=\the\catcode64\relax}% Restore @
133     \else
134       \let\bb1@tempc\empty % Not \relax
135     \fi
136     \bb1@exp{% For the 'uplevel' assignments
137   \endgroup
138   \bb1@tempc}} % empty or expand to set #1 with changes
139 \fi

```

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

```

140 \def\bb1@ifsamestring#1#2{%
141   \begingroup
142     \protected@edef\bb1@tempb{#1}%
143     \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
144     \protected@edef\bb1@tempc{#2}%
145     \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
146     \ifx\bb1@tempb\bb1@tempc
147       \aftergroup\@firstoftwo
148     \else
149       \aftergroup\@secondoftwo
150     \fi
151   \endgroup}
152 \chardef\bb1@engine=%
153 \ifx\directlua\undefined
154   \ifx\XeTeXinputencoding\undefined
155     \z@
156   \else
157     \tw@
158   \fi
159 \else
160   \@ne
161 \fi

```

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

```

162 \def\bbl@bsphack{%
163   \ifhmode
164     \hskip\z@skip
165     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166   \else
167     \let\bbl@esphack\@empty
168   \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`.

```

169 \def\bbl@cased{%
170   \ifx\oe\OE
171     \expandafter\in@\expandafter
172       {\expandafter\OE\expandafter}\expandafter{\oe}%
173     \ifin@
174       \bbl@afterelse\expandafter\MakeUppercase
175     \else
176       \bbl@afterfi\expandafter\MakeLowercase
177     \fi
178   \else
179     \expandafter\@firstofone
180   \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`).

```

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

```

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

```

199 \<{*Define core switching macros}> \equiv
200 \ifx\language\@undefined
201   \csname newcount\endcsname\language
202 \fi
203 \</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 $\mathrm{T}_{\mathrm{E}}\mathrm{X} < 2$. Preserved for compatibility.

```
204 <(*Define core switching macros)> ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 <(/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 ($\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$, `babel.sty`)

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

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

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

```
245 <(Basic macros)>
```

```

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

```

254 \ifx\bbl@languages\undefined\else
255   \begingroup
256     \catcode\^^I=12
257     \@ifpackagewith{babel}{showlanguages}{%
258       \begingroup
259         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
260         \wlog{<*languages>}%
261         \bbl@languages
262         \wlog{</languages>}%
263       \endgroup}{%
264     \endgroup
265     \def\bbl@elt#1#2#3#4{%
266       \ifnum#2=\z@
267         \gdef\bbl@nulllanguage{#1}%
268         \def\bbl@elt##1##2##3##4{%
269           \fi}%
270     \bbl@languages
271 \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`.

```

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

```

292 \bbl@trace{key=value and another general options}
293 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
294 \def\bbl@tempb#1.#2{% Remove trailing dot
295   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
296 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
297   \ifx\@empty#2%
298     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
299   \else
300     \in@{,provide=}{, #1}%
301     \ifin@
302       \edef\bbl@tempc{%
303         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
304     \else
305       \in@{=}{#1}%
306       \ifin@
307         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
308       \else
309         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
310         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
311       \fi
312     \fi
313   \fi}
314 \let\bbl@tempc\@empty
315 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
316 \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.

```

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

```

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

```

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

```

343 \def\bbl@tempa#1=#2\bbl@tempa{%
344   \bbl@csarg\ifx{opt@#1}\@nnil

```

```

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

```

354 \let\bbl@language@opts\@empty
355 \DeclareOption*{%
356 \bbl@xin@{\string=}{\CurrentOption}%
357 \ifin@
358 \expandafter\bbl@tempa\CurrentOption\bbl@tempa
359 \else
360 \bbl@add@list\bbl@language@opts{\CurrentOption}%
361 \fi}

```

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

```

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

```

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=...

```

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

```

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

```

389 \def\bbl@ifshorthand#1{%
390 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
391 \ifin@
392 \expandafter\@firstoftwo

```

```

393 \else
394 \expandafter\@secondoftwo
395 \fi}

```

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

```

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

```

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

```

403 \ifx\bbl@opt@headfoot\@nnil\else
404 \g@addto@macro\resetactivechars{%
405 \set@typeset@protect
406 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
407 \let\protect\noexpand}
408 \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.

```

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

```

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

```

413 \bbl@trace{Defining IfBabelLayout}
414 \ifx\bbl@opt@layout\@nnil
415 \newcommand\IfBabelLayout[3]{#3}%
416 \else
417 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
418 \in@{,layout,},{, #1,}%
419 \ifin@
420 \def\bbl@opt@layout{#2}%
421 \bbl@replace\bbl@opt@layout{ }{.}%
422 \fi}
423 \newcommand\IfBabelLayout[1]{%
424 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
425 \ifin@
426 \expandafter\@firstoftwo
427 \else
428 \expandafter\@secondoftwo
429 \fi}
430 \fi
431 </package>
432 <*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*.

```

433 \ifx\ldf@quit\@undefined\else
434 \endinput\fi % Same line!

```



```

435 <<Make sure ProvidesFile is defined>>
436 \ProvidesFile{babel.def}[\<date>] <<version>> Babel common definitions]
437 \ifx\AtBeginDocument\@undefined % TODO. change test.
438 <<Emulate LaTeX>>
439 \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.

```

440 </core>
441 <*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.

```

442 \def\bbl@version{\<version>}
443 \def\bbl@date{\<date>}
444 <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.

```

445 \def\adddialect#1#2{%
446   \global\chardef#1#2\relax
447   \bbl@usehooks{adddialect}{\#1}{\#2}}%
448   \begingroup
449     \count@#1\relax
450     \def\bbl@elt##1##2##3##4{%
451       \ifnum\count@=##2\relax
452         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
453         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
454           set to \expandafter\string\csname l@##1\endcsname\%
455           (\string\language\the\count@). Reported}%
456         \def\bbl@elt####1####2####3####4{%
457           \fi}%
458         \bbl@cs{languages}%
459         \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.

```

460 \def\bbl@fixname#1{%
461   \begingroup
462     \def\bbl@tempe{l@}%
463     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
464     \bbl@tempd
465     {\lowercase\expandafter{\bbl@tempd}%
466     {\uppercase\expandafter{\bbl@tempd}%
467     \@empty
468     {\edef\bbl@tempd{\def\noexpand#1{\#1}}%
469     {\uppercase\expandafter{\bbl@tempd}}}%
470     {\edef\bbl@tempd{\def\noexpand#1{\#1}}%
471     {\lowercase\expandafter{\bbl@tempd}}}%
472     \@empty
473     \edef\bbl@tempd{\endgroup\def\noexpand#1{\#1}}%
474     \bbl@tempd
475     \bbl@exp{\bbl@usehooks{language}\#1}}%
476 \def\bbl@iflanguage#1{%
477   \@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`.

```

478 \def\bbl@bcpcase#1#2#3#4\@#5{%
479   \ifx\@empty#3%
480     \uppercase{\def#5{#1#2}}%
481   \else
482     \uppercase{\def#5{#1}}%
483     \lowercase{\edef#5{#5#2#3#4}}%
484   \fi}
485 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
486   \let\bbl@bcp\relax
487   \lowercase{\def\bbl@tempa{#1}}%
488   \ifx\@empty#2%
489     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
490   \else\ifx\@empty#3%
491     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
492     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
493     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
494     }%
495     \ifx\bbl@bcp\relax
496       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
497     \fi
498   \else
499     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
500     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
501     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
502     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
503     }%
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-\bbl@tempc.ini}%
511       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
512       }%
513     \fi
514     \ifx\bbl@bcp\relax
515       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
516     \fi
517   \fi\fi}
518 \let\bbl@initoload\relax
519 \def\bbl@provide@locale{%
520   \ifx\babelprovide\@undefined
521     \bbl@error{For a language to be defined on the fly 'base'\\%
522               is not enough, and the whole package must be\\%
523               loaded. Either delete the 'base' option or\\%
524               request the languages explicitly}%
525     {See the manual for further details.}%
526   \fi
527   \let\bbl@auxname\language\name % Still necessary. TODO
528   \bbl@ifunset{bbl@bcp@map@\language\name}{}% Move uplevel??
529   {\edef\language\name{\@nameuse{bbl@bcp@map@\language\name}}}%
530   \ifbbl@bcppallowed
531     \expandafter\ifx\csname date\language\name\endcsname\relax
532     \expandafter
533     \bbl@bcpllookup\language\name-\@empty-\@empty-\@empty\@
534     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcpllookup
535       \edef\language\name{\bbl@bcp@prefix\bbl@bcp}%

```

```

536 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
537 \expandafter\ifx\csname date\language\endcsname\relax
538 \let\bbl@initoload\bbl@bcp
539 \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
540 \let\bbl@initoload\relax
541 \fi
542 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
543 \fi
544 \fi
545 \fi
546 \expandafter\ifx\csname date\language\endcsname\relax
547 \IfFileExists{babel-\language.tex}%
548 {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
549 {}%
550 \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.

```

551 \def\iflanguage#1{%
552 \bbl@iflanguage{#1}%
553 \ifnum\csname l@#1\endcsname=\language
554 \expandafter\@firstoftwo
555 \else
556 \expandafter\@secondoftwo
557 \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.

```

558 \let\bbl@select@type\z@
559 \edef\selectlanguage{%
560 \noexpand\protect
561 \expandafter\noexpand\csname selectlanguage \endcsname}

```

Because the command `\selectlanguage` could be used in a moving argument it expands to `\protect\selectlanguageL`. Therefore, we have to make sure that a macro `\protect` exists. If it doesn't it is `\let` to `\relax`.

```
562 \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.

```
563 \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.

```
564 \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`

```

565 \def\bbl@push@language{%
566   \ifx\language\undefined\else
567     \ifx\currentgrouplevel\undefined
568       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
569     \else
570       \ifnum\currentgrouplevel=\z@
571         \xdef\bbl@language@stack{\language+}%
572       \else
573         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
574       \fi
575     \fi
576   \fi}

```

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

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

```

577 \def\bbl@pop@lang#1+#2\@@{%
578   \edef\language{#1}%
579   \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).

```

580 \let\bbl@ifrestoring\@secondoftwo
581 \def\bbl@pop@language{%
582   \expandafter\bbl@pop@lang\bbl@language@stack\@@
583   \let\bbl@ifrestoring\@firstoftwo
584   \expandafter\bbl@set@language\expandafter{\language}%
585   \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).

```

586 \chardef\localeid\z@
587 \def\bbl@id@last{0} % No real need for a new counter
588 \def\bbl@id@assign{%
589   \bbl@ifunset{\bbl@id@\language}%
590   {\count@\bbl@id@last\relax
591    \advance\count@\@ne
592    \bbl@csarg\chardef{id@\language}\count@
593    \edef\bbl@id@last{\the\count@}%
594    \ifcase\bbl@engine\or
595      \directlua{
596        Babel = Babel or {}
597        Babel.locale_props = Babel.locale_props or {}
598        Babel.locale_props[\bbl@id@last] = {}
599        Babel.locale_props[\bbl@id@last].name = '\language'
600      }%
601    \fi}%
602  {}%
603  \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

604 \expandafter\def\csname selectlanguage \endcsname#1{%

```

```

605 \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\tw@\fi
606 \bbl@push@language
607 \aftergroup\bbl@pop@language
608 \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 `laTeX`, is to avoid the `\write` altogether when not needed).

```

609 \def\BabelContentsFiles{toc,lof,lot}
610 \def\bbl@set@language#1{% from selectlanguage, pop@
611 % The old buggy way. Preserved for compatibility.
612 \edef\language{%
613 \ifnum\escapechar=\expandafter`\string#1\@empty
614 \else\string#1\@empty\fi}%
615 \ifcat\relax\noexpand#1%
616 \expandafter\ifx\csname date\language\endcsname\relax
617 \edef\language{#1}%
618 \let\localname\language
619 \else
620 \bbl@info{Using '\string\language' instead of 'language' is\\%
621 deprecated. If what you want is to use a\\%
622 macro containing the actual locale, make\\%
623 sure it does not not match any language.\\%
624 Reported}%
625 \ifx\scantokens\@undefined
626 \def\localname{??}%
627 \else
628 \scantokens\expandafter{\expandafter
629 \def\expandafter\localname\expandafter{\language}}%
630 \fi
631 \fi
632 \else
633 \def\localname{#1}% This one has the correct catcodes
634 \fi
635 \select@language{\language}%
636 % write to auxs
637 \expandafter\ifx\csname date\language\endcsname\relax\else
638 \if@filesw
639 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
640 \bbl@savelastskip
641 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
642 \bbl@restorelastskip
643 \fi
644 \bbl@usehooks{write}{}}%
645 \fi
646 \fi}
647 %
648 \let\bbl@restorelastskip\relax
649 \let\bbl@savelastskip\relax
650 %
651 \newif\ifbbl@bcpallowed
652 \bbl@bcpallowedfalse
653 \def\select@language#1{% from set@, babel@aux
654 \ifx\bbl@selectorname\@empty

```

```

655 \def\bbl@selectorname{select}%
656 % set hmap
657 \fi
658 \ifnum\bbl@hmapsel=\@cclv\chardef\bbl@hmapsel4\relax\fi
659 % set name
660 \edef\language{#1}%
661 \bbl@fixname\language
662 % TODO. name@map must be here?
663 \bbl@provide@locale
664 \bbl@iflanguage\language{%
665 \let\bbl@select@type\z@
666 \expandafter\bbl@switch\expandafter{\language}}
667 \def\babel@aux#1#2{%
668 \select@language{#1}%
669 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
670 \writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
671 \def\babel@toc#1#2{%
672 \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.

```

673 \newif\ifbbl@usedategroup
674 \let\bbl@savextras\@empty
675 \def\bbl@switch#1{% from select@, foreign@
676 % make sure there is info for the language if so requested
677 \bbl@ensureinfo{#1}%
678 % restore
679 \originalTeX
680 \expandafter\def\expandafter\originalTeX\expandafter{%
681 \csname noextras#1\endcsname
682 \let\originalTeX\@empty
683 \babel@beginsave}%
684 \bbl@usehooks{afterreset}}}%
685 \languageshorthands{none}%
686 % set the locale id
687 \bbl@id@assign
688 % switch captions, date
689 % No text is supposed to be added here, so we remove any
690 % spurious spaces.
691 \bbl@bsphack
692 \ifcase\bbl@select@type
693 \csname captions#1\endcsname\relax
694 \csname date#1\endcsname\relax
695 \else
696 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
697 \ifin@
698 \csname captions#1\endcsname\relax
699 \fi
700 \bbl@xin@{,date,}{,\bbl@select@opts,}%
701 \ifin@ % if \foreign... within \<lang>date
702 \csname date#1\endcsname\relax
703 \fi
704 \fi

```

```

705 \bbl@esphack
706 % switch extras
707 \csname bbl@preextras@#1\endcsname
708 \bbl@usehooks{beforeextras}{}%
709 \csname extras#1\endcsname\relax
710 \bbl@usehooks{afterextras}{}%
711 % > babel-ensure
712 % > babel-sh-<short>
713 % > babel-bidi
714 % > babel-fontspec
715 \let\bbl@savextras\@empty
716 % hyphenation - case mapping
717 \ifcase\bbl@opt@hyphenmap\or
718   \def\BabelLower##1##2{\lccode##1=##2\relax}%
719   \ifnum\bbl@hymapsel>4\else
720     \csname\language\ @bbl@hyphenmap\endcsname
721     \fi
722   \chardef\bbl@opt@hyphenmap\z@
723 \else
724   \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
725     \csname\language\ @bbl@hyphenmap\endcsname
726     \fi
727 \fi
728 \let\bbl@hymapsel\@cclv
729 % hyphenation - select rules
730 \ifnum\csname l@\language\endcsname=\l@unhyphenated
731   \edef\bbl@tempa{u}%
732 \else
733   \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
734 \fi
735 % linebreaking - handle u, e, k (v in the future)
736 \bbl@xin@{/u}{/\bbl@tempa}%
737 \ifin@ \else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
738 \ifin@ \else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
739 \ifin@ \else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
740 \ifin@ \else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
741 \ifin@
742   % unhyphenated/kashida/elongated/padding = allow stretching
743   \language\l@unhyphenated
744   \babel@savevariable\emergencystretch
745   \emergencystretch\maxdimen
746   \babel@savevariable\hbadness
747   \hbadness\@M
748 \else
749   % other = select patterns
750   \bbl@patterns{#1}%
751 \fi
752 % hyphenation - mins
753 \babel@savevariable\lefthyphenmin
754 \babel@savevariable\righthyphenmin
755 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
756   \set@hyphenmins\tw@\thr@@\relax
757 \else
758   \expandafter\expandafter\expandafter\set@hyphenmins
759     \csname #1hyphenmins\endcsname\relax
760 \fi
761 \let\bbl@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.

```
762 \long\def\otherlanguage#1{%
763   \def\bbl@selectorname{other}%
764   \ifnum\bbl@hymapset=\@cclv\let\bbl@hymapset\thr@@\fi
765   \csname selectlanguage \endcsname{#1}%
766   \ignorespaces}
```

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

```
767 \long\def\endotherlanguage{%
768   \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`.

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

```
776 \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`.

```
777 \providecommand\bbl@beforeforeign{}
778 \edef\foreignlanguage{%
779   \noexpand\protect
780   \expandafter\noexpand\csname foreignlanguage \endcsname}
781 \expandafter\def\csname foreignlanguage \endcsname{%
782   \@ifstar\bbl@foreign@s\bbl@foreign@x}
783 \providecommand\bbl@foreign@x[3][{}]{%
784   \beginngroup
785     \def\bbl@selectorname{foreign}%
786     \def\bbl@select@opts{#1}%
787     \let\BabelText\@firstofone
788     \bbl@beforeforeign
789     \foreign@language{#2}%
790     \bbl@usehooks{foreign}{}%
791     \BabelText{#3}% Now in horizontal mode!
```



```

792 \endgroup}
793 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
794 \begingroup
795 {\par}%
796 \def\bbl@selectorname{foreign*}%
797 \let\bbl@select@opts\empty
798 \let\BabelText\@firstofone
799 \foreign@language{#1}%
800 \bbl@usehooks{foreign*}{}%
801 \bbl@dirparastext
802 \BabelText{#2}% Still in vertical mode!
803 {\par}%
804 \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`.

```

805 \def\foreign@language#1{%
806 % set name
807 \edef\language#1}%
808 \ifbbl@usedategroup
809 \bbl@add\bbl@select@opts{,date,}%
810 \bbl@usedategroupfalse
811 \fi
812 \bbl@fixname\language#1
813 % TODO. name@map here?
814 \bbl@provide@locale
815 \bbl@iflanguage\language#1{%
816 \let\bbl@select@type\ne
817 \expandafter\bbl@switch\expandafter{\language#1}}

```

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

```

818 \def\IfBabelSelectorTF#1{%
819 \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \empty,}%
820 \ifin@
821 \expandafter\@firstoftwo
822 \else
823 \expandafter\@secondoftwo
824 \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's` has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\babelhyphenation`, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that `:ENC` is taken into account) has been set, then use `\hyphenation` with both global and language exceptions and empty the latter to mark they must not be set again.

```

825 \let\bbl@hyphlist\empty
826 \let\bbl@hyphenation@relax
827 \let\bbl@pttnlist\empty
828 \let\bbl@patterns@relax
829 \let\bbl@hymapsel=\ccclv
830 \def\bbl@patterns#1{%
831 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
832 \csname l@#1\endcsname
833 \edef\bbl@tempa{#1}%
834 \else
835 \csname l@#1:\f@encoding\endcsname
836 \edef\bbl@tempa{#1:\f@encoding}%
837 \fi
838 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
839 % > luatex

```

```

840 \@ifundefined{bbl@hyphenation@}{\relax!
841 \beginngroup
842 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
843 \ifin@ \else
844 \@expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
845 \hyphenation{%
846 \bbl@hyphenation@
847 \@ifundefined{bbl@hyphenation@#1}%
848 \@empty
849 {\space\csname bbl@hyphenation@#1\endcsname}}%
850 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
851 \fi
852 \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*`.

```

853 \def\hyphenrules#1{%
854 \edef\bbl@tempf{\#1}%
855 \bbl@fixname\bbl@tempf
856 \bbl@iflanguage\bbl@tempf{%
857 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
858 \ifx\languageshorthands\undefined\else
859 \languageshorthands{none}%
860 \fi
861 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
862 \set@hyphenmins\tw@\thr@@\relax
863 \else
864 \expandafter\expandafter\expandafter\set@hyphenmins
865 \csname\bbl@tempf hyphenmins\endcsname\relax
866 \fi}}
867 \let\endhyphenrules\@empty

```

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

```

868 \def\providehyphenmins#1#2{%
869 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
870 \@namedef{\#1hyphenmins}{\#2}%
871 \fi}

```

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

```

872 \def\set@hyphenmins#1#2{%
873 \lefthyphenmin#1\relax
874 \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.

```

875 \ifx\ProvidesFile\undefined
876 \def\ProvidesLanguage#1[#2 #3 #4]{%
877 \wlog{Language: #1 #4 #3 <#2>}%
878 }
879 \else
880 \def\ProvidesLanguage#1{%
881 \beginngroup
882 \catcode`\ 10 %
883 \@makeother\/%
884 \@ifnextchar[%]

```

```

885      {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
886 \def\@provideslanguage#1[#2]{%
887   \wlog{Language: #1 #2}%
888   \expandafter\xdef\csname ver@#1.1df\endcsname{#2}%
889   \endgroup}
890 \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`.

```
891 \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.

```
892 \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’:

```

893 \providecommand\setlocale{%
894   \bbl@error
895   {Not yet available}%
896   {Find an armchair, sit down and wait}}
897 \let\uselocale\setlocale
898 \let\locale\setlocale
899 \let\selectlocale\setlocale
900 \let\textlocale\setlocale
901 \let\textlanguage\setlocale
902 \let\languagetext\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.

```

903 \edef\bbl@nulllanguage{\string\language=0}
904 \def\bbl@nocaption{\protect\bbl@nocaption@i}
905 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
906   \global\@namedef{#2}{\textbf{?#1?}}}%
907   \@nameuse{#2}%
908   \edef\bbl@tempa{#1}%
909   \bbl@sreplace\bbl@tempa{name}{}}%
910   \bbl@warning{%
911     \@backslashchar#1 not set for '\language'. Please,\\%
912     define it after the language has been loaded\\%
913     (typically in the preamble) with:\\%
914     \string\setlocalecaption{\language}{\bbl@tempa}{.}\\%
915     Feel free to contribute on github.com/latex3/babel.\\%
916     Reported}}
917 \def\bbl@tentative{\protect\bbl@tentative@i}
918 \def\bbl@tentative@i#1{%
919   \bbl@warning{%
920     Some functions for '#1' are tentative.\\%
921     They might not work as expected and their behavior\\%
922     could change in the future.\\%
923     Reported}}
924 \def\@nolanerr#1{%
925   \bbl@error

```

```

926 {You haven't defined the language '#1' yet.\\%
927 Perhaps you misspelled it or your installation\\%
928 is not complete}%
929 {Your command will be ignored, type <return> to proceed}}
930 \def\@nopatterns#1{%
931 \bbl@warning
932 {No hyphenation patterns were preloaded for\\%
933 the language '#1' into the format.\\%
934 Please, configure your TeX system to add them and\\%
935 rebuild the format. Now I will use the patterns\\%
936 preloaded for \bbl@nulllanguage\space instead}}
937 \let\bbl@usehooks\@gobbletwo
938 \ifx\bbl@onlyswitch\@empty\endinput\fi
939 % Here ended switch.def

```

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

```

940 \ifx\directlua\@undefined\else
941 \ifx\bbl@luapatterns\@undefined
942 \input luababel.def
943 \fi
944 \fi
945 <Basic macros>
946 \bbl@trace{Compatibility with language.def}
947 \ifx\bbl@languages\@undefined
948 \ifx\directlua\@undefined
949 \openin1 = language.def % TODO. Remove hardcoded number
950 \ifeof1
951 \closein1
952 \message{I couldn't find the file language.def}
953 \else
954 \closein1
955 \begingroup
956 \def\addlanguage#1#2#3#4#5{%
957 \expandafter\ifx\csname lang@#1\endcsname\relax\else
958 \global\expandafter\let\csname l@#1\expandafter\endcsname
959 \csname lang@#1\endcsname
960 \fi}%
961 \def\uselanguage#1{%
962 \input language.def
963 \endgroup
964 \fi
965 \closein1
966 \fi
967 \chardef\l@english\z@
968 \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.

```

969 \def\addto#1#2{%
970 \ifx#1\@undefined
971 \def#1{#2}%
972 \else
973 \ifx#1\relax
974 \def#1{#2}%
975 \else
976 {\toks@\expandafter{#1#2}%
977 \xdef#1{\the\toks@}}%
978 \fi
979 \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.

```
980 \def\bbl@withactive#1#2{%
981   \begingroup
982     \lccode`~=`#2\relax
983     \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`.

```
984 \def\bbl@redefine#1{%
985   \edef\bbl@tempa{\bbl@stripslash#1}%
986   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
987   \expandafter\def\csname\bbl@tempa\endcsname{
988     \@onlypreamble\bbl@redefine
```

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

```
989 \def\bbl@redefine@long#1{%
990   \edef\bbl@tempa{\bbl@stripslash#1}%
991   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
992   \long\expandafter\def\csname\bbl@tempa\endcsname{
993     \@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`.

```
994 \def\bbl@redefineroobust#1{%
995   \edef\bbl@tempa{\bbl@stripslash#1}%
996   \bbl@ifunset{\bbl@tempa\space}%
997     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
998       \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
999     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1000   \@namedef{\bbl@tempa\space}%
1001   \@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.

```
1002 \bbl@trace{Hooks}
1003 \newcommand\AddBabelHook[3][]{%
1004   \bbl@ifunset{\bbl@hk##2}{\EnableBabelHook{#2}}}%
1005   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1006   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1007   \bbl@ifunset{\bbl@ev##2@##3@#1}%
1008     {\bbl@csarg\bbl@add{ev##3@#1}{\bbl@elth{#2}}}%
1009     {\bbl@csarg\let{ev##2@##3@#1}\relax}%
1010   \bbl@csarg\newcommand{ev##2@##3@#1}{\bbl@tempb}}
1011 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk##1}\@firstofone}
1012 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk##1}\@gobble}
1013 \def\bbl@usehooks#1#2{%
1014   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1015   \def\bbl@elth##1{%
1016     \bbl@cs{hk##1}{\bbl@cs{ev##1@#1@#2}}%
1017     \bbl@cs{ev##1@#1}%
1018     \ifx\language\@undefined\else % Test required for Plain (?)
1019       \ifx\UseHook\@undefined\else\UseHook{babel/\language/#1}\fi
1020       \def\bbl@elth##1{%
1021         \bbl@cs{hk##1}{\bbl@c1{ev##1@#1@#2}}%
```

```

1022 \bbl@c1{ev@#1}%
1023 \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).

```

1024 \def\bbl@evargs{,% <- don't delete this comma
1025 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1026 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1027 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1028 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1029 beforestart=0,languagename=2}
1030 \ifx\NewHook\undefined\else
1031 \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1032 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1033 \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.

```

1034 \bbl@trace{Defining babelensure}
1035 \newcommand\babelensure[2][{}]{%
1036 \AddBabelHook{babel-ensure}{afterextras}{%
1037 \ifcase\bbl@select@type
1038 \bbl@c1{e}%
1039 \fi}%
1040 \begingroup
1041 \let\bbl@ens@include\@empty
1042 \let\bbl@ens@exclude\@empty
1043 \def\bbl@ens@fontenc{\relax}%
1044 \def\bbl@tempb##1{%
1045 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1046 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1047 \def\bbl@tempb##1=#2\@@{\@namedef{\bbl@ens@##1}{##2}}%
1048 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1049 \def\bbl@tempc{\bbl@ensure}%
1050 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1051 \expandafter{\bbl@ens@include}}%
1052 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1053 \expandafter{\bbl@ens@exclude}}%
1054 \toks@\expandafter{\bbl@tempc}%
1055 \bbl@exp{%
1056 \endgroup
1057 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1058 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1059 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist
1060 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1061 \edef##1{\noexpand\bbl@nocaption
1062 {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1063 \fi
1064 \ifx##1\@empty\else
1065 \in@{##1}{#2}%
1066 \ifin@\else
1067 \bbl@ifunset{\bbl@ensure@\languagename}%
1068 {\bbl@exp{%
1069 \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1070 \\\foreignlanguage{\languagename}%

```

```

1071         {\ifx\relax#3\else
1072         \\fontencoding{#3}\\selectfont
1073         \fi
1074         #####1}}}%
1075     }%
1076     \toks@{\expandafter{##1}%
1077     \edef##1{%
1078         \bbl@csarg\noexpand{ensure@\language}\expandafter{##1}%
1079         {\the\toks@}}%
1080     \fi
1081     \expandafter\bbl@tempb
1082     \fi}%
1083 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1084 \def\bbl@tempa##1{% elt for include list
1085     \ifx##1\@empty\else
1086         \bbl@csarg\in@{ensure@\language}\expandafter}\expandafter{##1}%
1087         \ifin@\else
1088             \bbl@tempb##1\@empty
1089             \fi
1090         \expandafter\bbl@tempa
1091         \fi}%
1092     \bbl@tempa#1\@empty}
1093 \def\bbl@captionslist{%
1094     \prefacename\refname\abstractname\bibname\chaptername\appendixname
1095     \contentsname\listfigurename\listtablename\indexname\figurename
1096     \tablename\partname\enclname\ccname\headtoname\pagename\seename
1097     \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`.

```

1098 \bbl@trace{Macros for setting language files up}
1099 \def\bbl@ldfinit{%
1100     \let\bbl@screset\@empty
1101     \let\BabelStrings\bbl@opt@string
1102     \let\BabelOptions\@empty
1103     \let\BabelLanguages\relax
1104     \ifx\originalTeX\@undefined
1105         \let\originalTeX\@empty
1106     \else
1107         \originalTeX
1108     \fi}
1109 \def\LdfInit#1#2{%
1110     \chardef\atcatcode=\catcode`\@
1111     \catcode`\@=11\relax
1112     \chardef\eqcatcode=\catcode`\=

```

```

1113 \catcode`\==12\relax
1114 \expandafter\if\expandafter\@backslashchar
1115         \expandafter\@car\string#2\@nil
1116     \ifx#2\@undefined\else
1117         \ldf@quit{#1}%
1118     \fi
1119 \else
1120     \expandafter\ifx\csname#2\endcsname\relax\else
1121         \ldf@quit{#1}%
1122     \fi
1123 \fi
1124 \bbl@ldfinit}

```

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

```

1125 \def\ldf@quit#1{%
1126     \expandafter\main@language\expandafter{#1}%
1127     \catcode`\@=\atcatcode \let\atcatcode\relax
1128     \catcode`\==\eqcatcode \let\eqcatcode\relax
1129     \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.

```

1130 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1131     \bbl@afterlang
1132     \let\bbl@afterlang\relax
1133     \let\BabelModifiers\relax
1134     \let\bbl@screset\relax}%
1135 \def\ldf@finish#1{%
1136     \loadlocalcfg{#1}%
1137     \bbl@afterldf{#1}%
1138     \expandafter\main@language\expandafter{#1}%
1139     \catcode`\@=\atcatcode \let\atcatcode\relax
1140     \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`.

```

1141 \@onlypreamble\LdfInit
1142 \@onlypreamble\ldf@quit
1143 \@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.

```

1144 \def\main@language#1{%
1145     \def\bbl@main@language{#1}%
1146     \let\language\bbl@main@language % TODO. Set locale name
1147     \bbl@id@assign
1148     \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`.

```

1149 \def\bbl@beforestart{%
1150     \def\@nolanerr##1{%
1151         \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1152     \bbl@usehooks{beforestart}{}%
1153     \global\let\bbl@beforestart\relax}
1154 \AtBeginDocument{%
1155     {\@nameuse{bbl@beforestart}}% Group!
1156     \if@filesw

```



```

1157 \providecommand\babel@aux[2]{}%
1158 \immediate\write\@mainaux{%
1159   \string\providecommand\string\babel@aux[2]{}%
1160   \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1161 \fi
1162 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1163 \ifbbl@single % must go after the line above.
1164   \renewcommand\selectlanguage[1]{}%
1165   \renewcommand\foreignlanguage[2]{#2}%
1166   \global\let\babel@aux\@gobbletwo % Also as flag
1167 \fi
1168 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1169 \def\select@language@x#1{%
1170   \ifcase\bbl@select@type
1171     \bbl@ifsamestring\language#1\{\}\select@language{#1}%
1172   \else
1173     \select@language{#1}%
1174   \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.

```

1175 \bbl@trace{Shorhands}
1176 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1177   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1178   \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1179   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1180     \begingroup
1181       \catcode`#1\active
1182       \nfss@catcodes
1183       \ifnum\catcode`#1=\active
1184         \endgroup
1185         \bbl@add\nfss@catcodes{\@makeother#1}%
1186       \else
1187         \endgroup
1188       \fi
1189   \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.

```

1190 \def\bbl@remove@special#1{%
1191   \begingroup
1192     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1193       \else\noexpand##1\noexpand##2\fi}%
1194     \def\do{\x\do}%
1195     \def\@makeother{\x\@makeother}%
1196   \edef\x{\endgroup
1197     \def\noexpand\dospecials{\dospecials}%
1198     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1199       \def\noexpand\@sanitize{\@sanitize}%
1200     \fi}%
1201   \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).

```
1202 \def\bbl@active@def#1#2#3#4{%
1203   \@namedef{#3#1}{%
1204     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1205       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1206     \else
1207       \bbl@afterfi\csname#2@sh@#1\endcsname
1208     \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.

```
1209   \long\@namedef{#3@arg#1}##1{%
1210     \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1211       \bbl@afterelse\csname#4#1\endcsname##1%
1212     \else
1213       \bbl@afterfi\csname#2@sh@#1@string##1\endcsname
1214     \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.

```
1215 \def\initiate@active@char#1{%
1216   \bbl@ifunset{active@char\string#1}%
1217   {\bbl@withactive
1218     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1219   {}}

```

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).

```
1220 \def\@initiate@active@char#1#2#3{%
1221   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1222   \ifx#1\undefined
1223     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1224   \else
1225     \bbl@csarg\let{oridef@#2}#1%
1226     \bbl@csarg\edef{oridef@#2}{%
1227       \let\noexpand#1%
1228       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1229   \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*).

```
1230   \ifx#1#3\relax
1231     \expandafter\let\csname normal@char#2\endcsname#3%

```

```

1232 \else
1233 \bbl@info{Making #2 an active character}%
1234 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1235 \namedef{normal@char#2}{%
1236 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1237 \else
1238 \namedef{normal@char#2}{#3}%
1239 \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).

```

1240 \bbl@restoreactive{#2}%
1241 \AtBeginDocument{%
1242 \catcode`#2\active
1243 \if@files
1244 \immediate\write\@mainaux{\catcode`\string#2\active}%
1245 \fi}%
1246 \expandafter\bbl@add@special\csname#2\endcsname
1247 \catcode`#2\active
1248 \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}).

```

1249 \let\bbl@tempa\@firstoftwo
1250 \if\string^#2%
1251 \def\bbl@tempa{\noexpand\textormath}%
1252 \else
1253 \ifx\bbl@mathnormal\@undefined\else
1254 \let\bbl@tempa\bbl@mathnormal
1255 \fi
1256 \fi
1257 \expandafter\edef\csname active@char#2\endcsname{%
1258 \bbl@tempa
1259 {\noexpand\if@safe@actives
1260 \noexpand\expandafter
1261 \expandafter\noexpand\csname normal@char#2\endcsname
1262 \noexpand\else
1263 \noexpand\expandafter
1264 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1265 \noexpand\fi}%
1266 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1267 \bbl@csarg\edef{doactive#2}{%
1268 \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!).

```

1269 \bbl@csarg\edef{active@#2}{%
1270 \noexpand\active@prefix\noexpand#1%
1271 \expandafter\noexpand\csname active@char#2\endcsname}%
1272 \bbl@csarg\edef{normal@#2}{%
1273 \noexpand\active@prefix\noexpand#1%
1274 \expandafter\noexpand\csname normal@char#2\endcsname}%
1275 \bbl@ncarg\let#1{\bbl@normal@#2}%

```

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.

```
1276 \bbl@active@def#2\user@group{user@active}{language@active}%
1277 \bbl@active@def#2\language@group{language@active}{system@active}%
1278 \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.

```
1279 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1280 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1281 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1282 {\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.

```
1283 \if\string'#2%
1284 \let\prim@s\bbl@prim@s
1285 \let\active@math@prime#1%
1286 \fi
1287 \bbl@usehooks{initiateactive}{#{1}{#2}{#3}}}
```

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

```
1288 <<{*More package options}>> ≡
1289 \DeclareOption{math=active}{%
1290 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1291 <</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.

```
1292 \ifpackagewith{babel}{KeepShorthandsActive}%
1293 {\let\bbl@restoreactive\@gobble}%
1294 {\def\bbl@restoreactive#1{%
1295 \bbl@exp{%
1296 \\\AfterBabelLanguage\\CurrentOption
1297 {\catcode`#1=\the\catcode`#1\relax}%
1298 \\\AtEndOfPackage
1299 {\catcode`#1=\the\catcode`#1\relax}}}%
1300 \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.

```
1301 \def\bbl@sh@select#1#2{%
1302 \expandafter\ifx\csname#1sh#2@sel\endcsname\relax
1303 \bbl@afterelse\bbl@scndcs
1304 \else
1305 \bbl@afterfi\csname#1sh#2@sel\endcsname
1306 \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.

```

1307 \begingroup
1308 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1309 {\gdef\active@prefix#1{%
1310   \ifx\protect\@typeset@protect
1311   \else
1312     \ifx\protect\@unexpandable@protect
1313       \noexpand#1%
1314     \else
1315       \protect#1%
1316     \fi
1317     \expandafter\@gobble
1318   \fi}}
1319 {\gdef\active@prefix#1{%
1320   \ifincsname
1321     \string#1%
1322     \expandafter\@gobble
1323   \else
1324     \ifx\protect\@typeset@protect
1325     \else
1326       \ifx\protect\@unexpandable@protect
1327         \noexpand#1%
1328       \else
1329         \protect#1%
1330       \fi
1331       \expandafter\expandafter\expandafter\@gobble
1332     \fi
1333   \fi}}
1334 \endgroup

```

`\if@safe@actives` In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ‘other’) 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⟩`. When this expansion mode is active (with `\@safe@activetrue`), something like `"13"13` becomes `"12"12` in an `\edef` (in other words, shorthands are `\string`’ed). This contrasts with `\protected@edef`, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with `\@safe@activefalse`).

```

1335 \newif\if@safe@actives
1336 \@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.

```

1337 \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`.

```

1338 \chardef\bbl@activated\z@
1339 \def\bbl@activate#1{%
1340   \chardef\bbl@activated\@ne
1341   \bbl@withactive{\expandafter\let\expandafter}#1%
1342   \csname bbl@active@string#1\endcsname}
1343 \def\bbl@deactivate#1{%
1344   \chardef\bbl@activated\tw@
1345   \bbl@withactive{\expandafter\let\expandafter}#1%
1346   \csname bbl@normal@string#1\endcsname}

```

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

```

\bbl@scndcs
1347 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1348 \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.

```

1349 \def\babel@texpdf#1#2#3#4{%
1350   \ifx\texorpdfstring\undefined
1351     \textormath{#1}{#3}%
1352   \else
1353     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1354     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1355   \fi}
1356 %
1357 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1358 \def\@decl@short#1#2#3\@nil#4{%
1359   \def\bbl@tempa{#3}%
1360   \ifx\bbl@tempa\empty
1361     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1362     \bbl@ifunset{#1@sh@\string#2@}{}%
1363     {\def\bbl@tempa{#4}%
1364      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1365      \else
1366        \bbl@info
1367          {Redefining #1 shorthand \string#2\\%
1368           in language \CurrentOption}%
1369      \fi}%
1370     \@namedef{#1@sh@\string#2@}{#4}%
1371   \else
1372     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1373     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1374     {\def\bbl@tempa{#4}%
1375      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1376      \else
1377        \bbl@info
1378          {Redefining #1 shorthand \string#2\string#3\\%
1379           in language \CurrentOption}%
1380      \fi}%
1381     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1382   \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.

```

1383 \def\textormath{%
1384   \ifmmode
1385     \expandafter\@secondoftwo
1386   \else
1387     \expandafter\@firstoftwo
1388   \fi}

```

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

```

1389 \def\user@group{user}
1390 \def\language@group{english} % TODO. I don't like defaults
1391 \def\system@group{system}

```

`\useshorthands` This is the user level macro. It initializes and activates the character for use as a shorthand character (ie, it's active in the preamble). Languages can deactivate shorthands, so a starred version is also provided which activates them always after the language has been switched.

```

1392 \def\useshorthands{%
1393   \@ifstar\bb1@usesesh@s{\bb1@usesesh@x{}}%
1394 \def\bb1@usesesh@s#1{%
1395   \bb1@usesesh@
1396   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
1397   {#1}}
1398 \def\bb1@usesesh@x#1#2{%
1399   \bb1@ifshorthand{#2}%
1400   {\def\user@group{user}%
1401    \initiate@active@char{#2}%
1402    #1%
1403    \bb1@activate{#2}}%
1404   {\bb1@error
1405    {I can't declare a shorthand turned off (\string#2)}
1406    {Sorry, but you can't use shorthands which have been\\
1407     turned off in the package options}}}

```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user@<lang>` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bb1@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1408 \def\user@language@group{user@\language@group}
1409 \def\bb1@set@user@generic#1#2{%
1410   \bb1@ifunset{user@generic@active#1}%
1411   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
1412    \bb1@active@def#1\user@group{user@generic@active}{language@active}%
1413    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1414      \expandafter\noexpand\csname normal@char#1\endcsname}%
1415    \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1416      \expandafter\noexpand\csname user@active#1\endcsname}}%
1417   \@empty}
1418 \newcommand\defineshorthand[3][user]{%
1419   \edef\bb1@tempa{\zap@space#1 \@empty}%
1420   \bb1@for\bb1@tempb\bb1@tempa{%
1421     \if*\expandafter\@car\bb1@tempb\@nil
1422       \edef\bb1@tempb{user\expandafter\@gobble\bb1@tempb}%
1423       \@expandtwoargs
1424       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
1425     \fi
1426     \declare@shorthand{\bb1@tempb}{#2}{#3}}

```

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

```

1427 \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`.

```

1428 \def\aliasshorthand#1#2{%
1429   \bb1@ifshorthand{#2}%
1430   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1431     \ifx\document\@notprerr
1432       \@notshorthand{#2}%
1433     \else
1434       \initiate@active@char{#2}%
1435       \bb1@ccarg\let{active@char\string#2}{active@char\string#1}%
1436       \bb1@ccarg\let{normal@char\string#2}{normal@char\string#1}%

```

```

1437     \bbl@activate{#2}%
1438     \fi
1439     \fi}%
1440     {\bbl@error
1441      {Cannot declare a shorthand turned off (\string#2)}
1442      {Sorry, but you cannot use shorthands which have been\\%
1443       turned off in the package options}}}

```

\@notshorthand

```

1444 \def\@notshorthand#1{%
1445   \bbl@error{%
1446     The character '\string #1' should be made a shorthand character;\\%
1447     add the command \string\usesshorthands\string{#1\string} to
1448     the preamble.\\%
1449     I will ignore your instruction}%
1450   {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.

```

1451 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1452 \DeclareRobustCommand*\shorthandoff{%
1453   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1454 \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.

```

1455 \def\bbl@switch@sh#1#2{%
1456   \ifx#2\@nnil\else
1457     \bbl@ifunset{bbl@active@\string#2}%
1458     {\bbl@error
1459      {I can't switch '\string#2' on or off--not a shorthand}%
1460      {This character is not a shorthand. Maybe you made\\%
1461       a typing mistake? I will ignore your instruction.}}}%
1462     {\ifcase#1%   off, on, off*
1463      \catcode`#2\relax
1464      \or
1465      \catcode`#2\active
1466      \bbl@ifunset{bbl@shdef@\string#2}%
1467      {}%
1468      {\bbl@withactive{\expandafter\let\expandafter}#2%
1469       \csname bbl@shdef@\string#2\endcsname
1470       \bbl@csarg\let{shdef@\string#2}\relax}%
1471      \ifcase\bbl@activated\or
1472       \bbl@activate{#2}%
1473      \else
1474       \bbl@deactivate{#2}%
1475      \fi
1476      \or
1477      \bbl@ifunset{bbl@shdef@\string#2}%
1478      {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1479      {}%
1480      \csname bbl@oricat@\string#2\endcsname
1481      \csname bbl@oridef@\string#2\endcsname
1482      \fi}%
1483     \bbl@afterfi\bbl@switch@sh#1%
1484     \fi}

```

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


```

1485 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1486 \def\bbl@putsh#1{%
1487   \bbl@ifunset{\bbl@active@\string#1}%
1488   {\bbl@putsh@i#1\@empty\@nnil}%
1489   {\csname bbl@active@\string#1\endcsname}}
1490 \def\bbl@putsh@i#1#2\@nnil{%
1491   \csname\language@group @sh@\string#1@%
1492   \ifx\@empty#2\else\string#2\fi\endcsname}
1493 %
1494 \ifx\bbl@opt@shorthands\@nnil\else
1495   \let\bbl@s@initiate@active@char\initiate@active@char
1496   \def\initiate@active@char#1{%
1497     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1498   \let\bbl@s@switch@sh\bbl@switch@sh
1499   \def\bbl@switch@sh#1#2{%
1500     \ifx#2\@nnil\else
1501       \bbl@afterfi
1502       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1503       \fi}
1504   \let\bbl@s@activate\bbl@activate
1505   \def\bbl@activate#1{%
1506     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1507   \let\bbl@s@deactivate\bbl@deactivate
1508   \def\bbl@deactivate#1{%
1509     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1510 \fi

```

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

```

1511 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

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

```

1512 \def\bbl@prim@s{%
1513   \prime\futurelet\@let@token\bbl@pr@m@s}
1514 \def\bbl@if@primes#1#2{%
1515   \ifx#1\@let@token
1516     \expandafter\@firstoftwo
1517   \else\ifx#2\@let@token
1518     \bbl@afterelse\expandafter\@firstoftwo
1519   \else
1520     \bbl@afterfi\expandafter\@secondoftwo
1521   \fi\fi}
1522 \begingroup
1523   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^^
1524   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\^^
1525   \lowercase{%
1526     \gdef\bbl@pr@m@s{%
1527       \bbl@if@primes"%
1528       \pr@@@s
1529       {\bbl@if@primes*\^pr@@@t\egroup}}}
1530 \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).

```

1531 \initiate@active@char{~}
1532 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1533 \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.

```
1534 \expandafter\def\csname OT1dqpos\endcsname{127}
1535 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1536 \ifx\f@encoding\undefined
1537   \def\f@encoding{OT1}
1538 \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.

```
1539 \bbl@trace{Language attributes}
1540 \newcommand\languageattribute[2]{%
1541   \def\bbl@tempc{#1}%
1542   \bbl@fixname\bbl@tempc
1543   \bbl@iflanguage\bbl@tempc{%
1544     \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.

```
1545     \ifx\bbl@known@attrs\undefined
1546       \in@false
1547     \else
1548       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1549     \fi
1550     \ifin@
1551       \bbl@warning{%
1552         You have more than once selected the attribute '##1'\%
1553         for language #1. Reported}%
1554     \else
```

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

```
1555       \bbl@exp{%
1556         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1557       \edef\bbl@tempa{\bbl@tempc-##1}%
1558       \expandafter\bbl@ifknown@ttrrib\expandafter{\bbl@tempa}\bbl@attributes%
1559       {\csname\bbl@tempc @attr@##1\endcsname}%
1560       {\@attrerr{\bbl@tempc}{##1}}%
1561     \fi}}
1562 \@onlypreamble\languageattribute
```

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

```
1563 \newcommand*{\@attrerr}[2]{%
1564   \bbl@error
1565   {The attribute #2 is unknown for language #1.}%
1566   {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}`.

```
1567 \def\bbl@declare@ttribute#1#2#3{%
1568   \bbl@xin@{,#2,}{,\BabelModifiers,}%
```

```

1569 \ifin@
1570 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1571 \fi
1572 \bbl@add@list\bbl@attributes{#1-#2}%
1573 \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.

```

1574 \def\bbl@ifattributeset#1#2#3#4{%
1575 \ifx\bbl@known@attribs\undefined
1576 \in@false
1577 \else
1578 \bbl@xin@{, #1-#2, }{, \bbl@known@attribs,}%
1579 \fi
1580 \ifin@
1581 \bbl@afterelse#3%
1582 \else
1583 \bbl@afterfi#4%
1584 \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.

```

1585 \def\bbl@ifknown@ttrib#1#2{%
1586 \let\bbl@tempa\@secondoftwo
1587 \bbl@loopx\bbl@tempb{#2}{%
1588 \expandafter\in@\expandafter{\expandafter, \bbl@tempb, }{, #1,}%
1589 \ifin@
1590 \let\bbl@tempa\@firstoftwo
1591 \else
1592 \fi}%
1593 \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).

```

1594 \def\bbl@clear@ttribs{%
1595 \ifx\bbl@attributes\undefined\else
1596 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1597 \expandafter\bbl@clear@ttrib\bbl@tempa.
1598 }%
1599 \let\bbl@attributes\undefined
1600 \fi}
1601 \def\bbl@clear@ttrib#1-#2.{%
1602 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1603 \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`

```

1604 \bbl@trace{Macros for saving definitions}
1605 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1606 \newcount\babel@savecnt
1607 \babel@beginsave

```

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX`³¹. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive. To avoid messing saved definitions up, they are saved only the very first time.

```

1608 \def\babel@save#1{%
1609   \def\bbl@tempa{, #1,}% Clumsy, for Plain
1610   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1611     \expandafter{\expandafter,\bbl@savedextras,}%
1612     \expandafter\in\bbl@tempa
1613     \ifin@%else
1614       \bbl@add\bbl@savedextras{, #1,}%
1615       \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1616       \toks@ \expandafter{\originalTeX\let#1=}
1617       \bbl@exp{%
1618         \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}
1619       \advance\babel@savecnt@ne
1620     \fi}
1621 \def\babel@savevariable#1{%
1622   \toks@ \expandafter{\originalTeX #1=}
1623   \bbl@exp{\def\originalTeX{\the\toks@ \the#1\relax}}

```

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

```

1624 \def\bbl@frenchspacing{%
1625   \ifnum\the\sfcode`\.=\@m
1626     \let\bbl@nonfrenchspacing\relax
1627   \else
1628     \frenchspacing
1629     \let\bbl@nonfrenchspacing\nonfrenchspacing
1630   \fi}
1631 \let\bbl@nonfrenchspacing\nonfrenchspacing
1632 \let\bbl@elt\relax
1633 \edef\bbl@fs@chars{%
1634   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1635   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1636   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1637 \def\bbl@pre@fs{%
1638   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1639   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1640 \def\bbl@post@fs{%
1641   \bbl@save@sfcodes
1642   \edef\bbl@tempa{\bbl@cl{frspc}}%
1643   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1644   \if u\bbl@tempa      % do nothing
1645   \else\if n\bbl@tempa % non french
1646     \def\bbl@elt##1##2##3{%
1647       \ifnum\sfcode`##1=##2\relax
1648       \babel@savevariable{\sfcode`##1}%
1649       \sfcode`##1=##3\relax

```

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

```

1650     \fi}%
1651     \bbl@fs@chars
1652 \elseif y\bbl@tempa % french
1653 \def\bbl@elt##1##2##3{%
1654     \ifnum\sfcode`##1=##3\relax
1655         \babel@savevariable{\sfcode`##1}%
1656         \sfcode`##1=##2\relax
1657     \fi}%
1658     \bbl@fs@chars
1659 \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 `\cname` but the actual macro.

```

1660 \bbl@trace{Short tags}
1661 \def\babeltags#1{%
1662     \edef\bbl@tempa{\zap@space#1 \@empty}%
1663     \def\bbl@tempb##1=##2\@{%
1664         \edef\bbl@tempc{%
1665             \noexpand\newcommand
1666             \expandafter\noexpand\cname ##1\endcsname{%
1667                 \noexpand\protect
1668                 \expandafter\noexpand\cname otherlanguage*\endcsname{##2}}
1669             \noexpand\newcommand
1670             \expandafter\noexpand\cname text##1\endcsname{%
1671                 \noexpand\foreignlanguage{##2}}
1672         \bbl@tempc}%
1673     \bbl@for\bbl@tempa\bbl@tempa{%
1674         \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.

```

1675 \bbl@trace{Hyphens}
1676 \@onlypreamble\babelhyphenation
1677 \AtEndOfPackage{%
1678     \newcommand\babelhyphenation[2][\@empty]{%
1679         \ifx\bbl@hyphenation@ \relax
1680             \let\bbl@hyphenation@ \@empty
1681         \fi
1682         \ifx\bbl@hyphlist \@empty \else
1683             \bbl@warning{%
1684                 You must not intermingle \string\selectlanguage\space and\%
1685                 \string\babelhyphenation\space or some exceptions will not\%
1686                 be taken into account. Reported}%
1687             \fi
1688         \ifx\@empty#1%
1689             \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1690         \else
1691             \bbl@vforeach{#1}{%
1692                 \def\bbl@tempa{##1}%
1693                 \bbl@fixname\bbl@tempa
1694                 \bbl@iflanguage\bbl@tempa{%
1695                     \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1696                         \bbl@ifunset{\bbl@hyphenation@\bbl@tempa}%
1697                         {}%
1698                         {\cname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1699                         #2}}}%

```

1700 \fi}}

\bbl@allowhyphens This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip Opt plus Opt³².

```
1701 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1702 \def\bbl@t@one{T1}
1703 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
```

\babelhyphen Macros to insert common hyphens. Note the space before @ in \babelhyphen. Instead of protecting it with \DeclareRobustCommand, which could insert a \relax, we use the same procedure as shorthands, with \active@prefix.

```
1704 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1705 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1706 \def\bbl@hyphen{%
1707   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1708 \def\bbl@hyphen@i#1#2{%
1709   \bbl@ifunset{\bbl@hy#1#2\@empty}%
1710   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1711   {\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.

```
1712 \def\bbl@usehyphen#1{%
1713   \leavevmode
1714   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1715   \nobreak\hskip\z@skip}
1716 \def\bbl@usehyphen#1{%
1717   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1718 \def\bbl@hyphenchar{%
1719   \ifnum\hyphenchar\font=\m@ne
1720     \babellnullhyphen
1721   \else
1722     \char\hyphenchar\font
1723   \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.

```
1724 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1725 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1726 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1727 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1728 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1729 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1730 \def\bbl@hy@repeat{%
1731   \bbl@usehyphen{%
1732     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1733 \def\bbl@hy@@repeat{%
1734   \bbl@usehyphen{%
1735     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1736 \def\bbl@hy@empty{\hskip\z@skip}
1737 \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.

```
1738 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}
```

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

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 tool. It makes global a local variable. This is not the best solution, but it works.

```
1739 \bbl@trace{Multiencoding strings}
1740 \def\bbl@tglobal#1{\global\let#1#1}
```

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).

```
1741 \@ifpackagewith{babel}{nocase}%
1742   {\let\bbl@patchuclc\relax}%
1743   {\def\bbl@patchuclc{%
1744     \global\let\bbl@patchuclc\relax
1745     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1746     \gdef\bbl@uclc##1{%
1747       \let\bbl@encoded\bbl@encoded@uclc
1748       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1749       {##1}%
1750       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1751         \csname\language @bbl@uclc\endcsname}%
1752       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1753   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1754   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1755 <<(*More package options)>> ≡
1756 \DeclareOption{nocase}{}
1757 <</More package options>>
```

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

```
1758 <<(*More package options)>> ≡
1759 \let\bbl@opt@strings\@nnil % accept strings=value
1760 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1761 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1762 \def\BabelStringsDefault{generic}
1763 <</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.

```
1764 \@onlypreamble\StartBabelCommands
1765 \def\StartBabelCommands{%
1766   \begingroup
1767   \@tempcnta="7F
1768   \def\bbl@tempa{%
1769     \ifnum\@tempcnta>"FF\else
1770       \catcode\@tempcnta=11
1771       \advance\@tempcnta\@ne
1772       \expandafter\bbl@tempa
1773     \fi}%
1774   \bbl@tempa
1775   <<Macros local to BabelCommands>>
```

```

1776 \def\bbl@provstring##1##2{%
1777 \providecommand##1{##2}%
1778 \bbl@tglobal##1}%
1779 \global\let\bbl@scafter\@empty
1780 \let\StartBabelCommands\bbl@startcmds
1781 \ifx\BabelLanguages\relax
1782 \let\BabelLanguages\CurrentOption
1783 \fi
1784 \begingroup
1785 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1786 \StartBabelCommands}
1787 \def\bbl@startcmds{%
1788 \ifx\bbl@screset\@nnil\else
1789 \bbl@usehooks{stopcommands}{}%
1790 \fi
1791 \endgroup
1792 \begingroup
1793 \@ifstar
1794 {\ifx\bbl@opt@strings\@nnil
1795 \let\bbl@opt@strings\BabelStringsDefault
1796 \fi
1797 \bbl@startcmds@i}%
1798 \bbl@startcmds@i}
1799 \def\bbl@startcmds@i#1#2{%
1800 \edef\bbl@L{\zap@space#1 \@empty}%
1801 \edef\bbl@G{\zap@space#2 \@empty}%
1802 \bbl@startcmds@ii}
1803 \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.

```

1804 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1805 \let\SetString\@gobbletwo
1806 \let\bbl@stringdef\@gobbletwo
1807 \let\AfterBabelCommands\@gobble
1808 \ifx\@empty#1%
1809 \def\bbl@sc@label{generic}%
1810 \def\bbl@encstring##1##2{%
1811 \ProvideTextCommandDefault##1{##2}%
1812 \bbl@tglobal##1%
1813 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1814 \let\bbl@sctest\in@true
1815 \else
1816 \let\bbl@sc@charset\space % <- zapped below
1817 \let\bbl@sc@fontenc\space % <- " "
1818 \def\bbl@tempa##1=##2\@nil{%
1819 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1820 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1821 \def\bbl@tempa##1 ##2{% space -> comma
1822 ##1%
1823 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1824 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1825 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1826 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1827 \def\bbl@encstring##1##2{%
1828 \bbl@foreach\bbl@sc@fontenc{%

```



```

1829     \bbl@ifunset{T@####1}%
1830     }%
1831     {\ProvideTextCommand##1{####1}{##2}%
1832     \bbl@tglobal##1%
1833     \expandafter
1834     \bbl@tglobal\csname####1\string##1\endcsname}}}%
1835     \def\bbl@sctest{%
1836     \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1837     \fi
1838     \ifx\bbl@opt@strings\@nnil      % ie, no strings key -> defaults
1839     \else\ifx\bbl@opt@strings\relax  % ie, strings=encoded
1840     \let\AfterBabelCommands\bbl@aftercmds
1841     \let\SetString\bbl@setstring
1842     \let\bbl@stringdef\bbl@encstring
1843     \else      % ie, strings=value
1844     \bbl@sctest
1845     \ifin@
1846     \let\AfterBabelCommands\bbl@aftercmds
1847     \let\SetString\bbl@setstring
1848     \let\bbl@stringdef\bbl@provstring
1849     \fi\fi\fi
1850     \bbl@scswitch
1851     \ifx\bbl@G\@empty
1852     \def\SetString##1##2{%
1853     \bbl@error{Missing group for string \string##1}%
1854     {You must assign strings to some category, typically\\%
1855     captions or extras, but you set none}}%
1856     \fi
1857     \ifx\@empty#1%
1858     \bbl@usehooks{defaultcommands}{}%
1859     \else
1860     \@expandtwoargs
1861     \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1862     \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing. The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has been loaded).

```

1863 \def\bbl@forlang#1#2{%
1864   \bbl@for#1\bbl@L{%
1865     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1866     \ifin#2\relax\fi}}
1867 \def\bbl@scswitch{%
1868   \bbl@forlang\bbl@tempa{%
1869     \ifx\bbl@G\@empty\else
1870       \ifx\SetString\@gobbletwo\else
1871       \edef\bbl@GL{\bbl@G\bbl@tempa}%
1872       \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1873       \ifin\@empty\else
1874       \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1875       \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1876       \fi
1877     \fi
1878   \fi}}
1879 \AtEndOfPackage{%
1880   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}}{##2}}}%
1881   \let\bbl@scswitch\relax}
1882 \onlypreamble\EndBabelCommands

```

```

1883 \def\EndBabelCommands{%
1884   \bbl@usehooks{stopcommands}{}%
1885   \endgroup
1886   \endgroup
1887   \bbl@scafter}
1888 \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.

```

1889 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1890   \bbl@forlang\bbl@tempa{%
1891     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1892     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1893     {\bbl@exp{%
1894       \global\bbl@add\<\bbl@G\bbl@tempa>{\bbl@scset\#1\<\bbl@LC>}}}%
1895     }%
1896   \def\BabelString{#2}%
1897   \bbl@usehooks{stringprocess}{}%
1898   \expandafter\bbl@stringdef
1899     \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.

```

1900 \ifx\bbl@opt@strings\relax
1901   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1902   \bbl@patchuclc
1903   \let\bbl@encoded\relax
1904   \def\bbl@encoded@uclc#1{%
1905     \@inmathwarn#1%
1906     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1907       \expandafter\ifx\csname ?\string#1\endcsname\relax
1908         \TextSymbolUnavailable#1%
1909       \else
1910         \csname ?\string#1\endcsname
1911       \fi
1912     \else
1913       \csname\cf@encoding\string#1\endcsname
1914     \fi}
1915 \else
1916   \def\bbl@scset#1#2{\def#1{#2}}
1917 \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.

```

1918 << *Macros local to BabelCommands >> ≡
1919 \def\SetStringLoop###1##2{%
1920   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1921   \count@\z@
1922   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1923     \advance\count@\@ne
1924     \toks@\expandafter{\bbl@tempa}%
1925     \bbl@exp{%
1926       \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1927       \count@=\the\count@\relax}}}%
1928 <</Macros local to BabelCommands >>

```

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

```
1929 \def\bbl@aftercmds#1{%
1930   \toks@\expandafter{\bbl@scafter#1}%
1931   \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.

```
1932 <<(*Macros local to BabelCommands)>> ≡
1933   \newcommand\SetCase[3][{}{%
1934     \bbl@patchuclc
1935     \bbl@forlang\bbl@tempa{%
1936       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1937       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1938       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1939 <</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.

```
1940 <<(*Macros local to BabelCommands)>> ≡
1941   \newcommand\SetHyphenMap[1]{%
1942     \bbl@forlang\bbl@tempa{%
1943       \expandafter\bbl@stringdef
1944       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1945 <</Macros local to BabelCommands>>
```

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

```
1946 \newcommand\BabelLower[2]{% one to one.
1947   \ifnum\lccode#1=#2\else
1948     \babel@savevariable{\lccode#1}%
1949     \lccode#1=#2\relax
1950   \fi}
1951 \newcommand\BabelLowerMM[4]{% many-to-many
1952   \@tempcnta=#1\relax
1953   \@tempcntb=#4\relax
1954   \def\bbl@tempa{%
1955     \ifnum\@tempcnta>#2\else
1956       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1957       \advance\@tempcnta#3\relax
1958       \advance\@tempcntb#3\relax
1959       \expandafter\bbl@tempa
1960     \fi}%
1961   \bbl@tempa}
1962 \newcommand\BabelLowerMO[4]{% many-to-one
1963   \@tempcnta=#1\relax
1964   \def\bbl@tempa{%
1965     \ifnum\@tempcnta>#2\else
1966       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1967       \advance\@tempcnta#3
1968       \expandafter\bbl@tempa
1969     \fi}%
1970   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1971 <<(*More package options)>> ≡
1972 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1973 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1974 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1975 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@}
1976 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1977 <</More package options>>
```

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

```

1978 \AtEndOfPackage{%
1979   \ifx\bbbl@opt@hyphenmap\undefined
1980     \bbbl@xin@{,}{\bbbl@language@opts}%
1981     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1982   \fi}

```

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

```

1983 \newcommand\setlocalecaption{% TODO. Catch typos.
1984   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1985 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1986   \bbbl@trim@def\bbbl@tempa{#2}%
1987   \bbbl@xin@{.template}{\bbbl@tempa}%
1988   \ifin@
1989     \bbbl@ini@captions@template{#3}{#1}%
1990   \else
1991     \edef\bbbl@tempd{%
1992       \expandafter\expandafter\expandafter
1993       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1994     \bbbl@xin@
1995       {\expandafter\string\csname #2name\endcsname}%
1996       {\bbbl@tempd}%
1997     \ifin@ % Renew caption
1998       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
1999     \ifin@
2000       \bbbl@exp{%
2001         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2002         {\\bbbl@scset\<#2name>\<#1#2name>}%
2003         {}}%
2004       \else % Old way converts to new way
2005         \bbbl@ifunset{#1#2name}%
2006         {\bbbl@exp{%
2007           \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2008           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2009           {\def\<#2name>{\<#1#2name>}}%
2010           {}}}%
2011         {}}%
2012     \fi
2013   \else
2014     \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2015     \ifin@ % New way
2016       \bbbl@exp{%
2017         \\bbbl@add\<captions#1>{\\bbbl@scset\<#2name>\<#1#2name>}%
2018         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2019         {\\bbbl@scset\<#2name>\<#1#2name>}%
2020         {}}%
2021       \else % Old way, but defined in the new way
2022         \bbbl@exp{%
2023           \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2024           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2025           {\def\<#2name>{\<#1#2name>}}%
2026           {}}%
2027         \fi%
2028     \fi
2029   \@namedef{#1#2name}{#3}%
2030   \toks@\expandafter{\bbbl@captionslist}%
2031   \bbbl@exp{\\in@{\<#2name>}{\the\toks@}}%
2032   \ifin@ \else
2033     \bbbl@exp{\\bbbl@add\\bbbl@captionslist{\<#2name>}}%
2034     \bbbl@tglobal\bbbl@captionslist
2035   \fi

```

```

2036 \fi}
2037 % \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.

```

2038 \bbl@trace{Macros related to glyphs}
2039 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2040 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2041 \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.

```

2042 \def\save@sf@q#1{\leavevmode
2043 \begingroup
2044 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2045 \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.

```

2046 \ProvideTextCommand{\quotedblbase}{OT1}{%
2047 \save@sf@q{\set@low@box{\textquotedblright\}}%
2048 \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.

```

2049 \ProvideTextCommandDefault{\quotedblbase}{%
2050 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2051 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2052 \save@sf@q{\set@low@box{\textquoteright\}}%
2053 \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.

```

2054 \ProvideTextCommandDefault{\quotesinglbase}{%
2055 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2056 \ProvideTextCommand{\guillemetleft}{OT1}{%
2057 \ifmmode
2058 \ll
2059 \else
2060 \save@sf@q{\nobreak
2061 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2062 \fi}
2063 \ProvideTextCommand{\guillemetright}{OT1}{%
2064 \ifmmode
2065 \gg
2066 \else
2067 \save@sf@q{\nobreak
2068 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2069 \fi}
2070 \ProvideTextCommand{\guillemotleft}{OT1}{%

```

```

2071 \ifmmode
2072   \ll
2073 \else
2074   \save@sf@q{\nobreak
2075     \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2076   \fi}
2077 \ProvideTextCommand{\guillemotright}{OT1}{%
2078   \ifmmode
2079     \gg
2080   \else
2081     \save@sf@q{\nobreak
2082       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2083     \fi}

```

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

```

2084 \ProvideTextCommandDefault{\guillemetleft}{%
2085   \UseTextSymbol{OT1}{\guillemetleft}}
2086 \ProvideTextCommandDefault{\guillemetright}{%
2087   \UseTextSymbol{OT1}{\guillemetright}}
2088 \ProvideTextCommandDefault{\guillemotleft}{%
2089   \UseTextSymbol{OT1}{\guillemotleft}}
2090 \ProvideTextCommandDefault{\guillemotright}{%
2091   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2092 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2093   \ifmmode
2094     <%
2095   \else
2096     \save@sf@q{\nobreak
2097       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2098     \fi}
2099 \ProvideTextCommand{\guilsinglright}{OT1}{%
2100   \ifmmode
2101     >%
2102   \else
2103     \save@sf@q{\nobreak
2104       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2105     \fi}

```

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

```

2106 \ProvideTextCommandDefault{\guilsinglleft}{%
2107   \UseTextSymbol{OT1}{\guilsinglleft}}
2108 \ProvideTextCommandDefault{\guilsinglright}{%
2109   \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.

```

2110 \DeclareTextCommand{\ij}{OT1}{%
2111   i\kern-0.02em\bbl@allowhyphens j}
2112 \DeclareTextCommand{\IJ}{OT1}{%
2113   I\kern-0.02em\bbl@allowhyphens J}
2114 \DeclareTextCommand{\ij}{T1}{\char188}
2115 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2116 \ProvideTextCommandDefault{\ij}{%
2117   \UseTextSymbol{OT1}{\ij}}
2118 \ProvideTextCommandDefault{\IJ}{%
2119   \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).

```
2120 \def\crrtic@{\hrule height0.1ex width0.3em}
2121 \def\crttic@{\hrule height0.1ex width0.33em}
2122 \def\ddj@{%
2123   \setbox0\hbox{d}\dimen@=\ht0
2124   \advance\dimen@1ex
2125   \dimen@.45\dimen@
2126   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2127   \advance\dimen@ii.5ex
2128   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2129 \def\DDJ@{%
2130   \setbox0\hbox{D}\dimen@=.55\ht0
2131   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2132   \advance\dimen@ii.15ex % correction for the dash position
2133   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2134   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2135   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2136 %
2137 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2138 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2139 \ProvideTextCommandDefault{\dj}{%
2140   \UseTextSymbol{OT1}{\dj}}
2141 \ProvideTextCommandDefault{\DJ}{%
2142   \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.

```
2143 \DeclareTextCommand{\SS}{OT1}{SS}
2144 \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 2145 \ProvideTextCommandDefault{\glq}{%
2146   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
The definition of \grq depends on the fontencoding. With T1 encoding no extra kerning is needed.
2147 \ProvideTextCommand{\grq}{T1}{%
2148   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2149 \ProvideTextCommand{\grq}{TU}{%
2150   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2151 \ProvideTextCommand{\grq}{OT1}{%
2152   \save@sf@q{\kern-.0125em
2153     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2154     \kern.07em\relax}}
2155 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2156 \ProvideTextCommandDefault{\glqq}{%
2157   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
The definition of \grqq depends on the fontencoding. With T1 encoding no extra kerning is needed.
2158 \ProvideTextCommand{\grqq}{T1}{%
2159   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
```

```

2160 \ProvideTextCommand{\grqq}{TU}{%
2161   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2162 \ProvideTextCommand{\grqq}{OT1}{%
2163   \save@sf@q{\kern-.07em
2164     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2165     \kern.07em\relax}}
2166 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

\flq The ‘french’ single guillemets.

```

\frq 2167 \ProvideTextCommandDefault{\flq}{%
2168   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2169 \ProvideTextCommandDefault{\frq}{%
2170   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq The ‘french’ double guillemets.

```

\frqq 2171 \ProvideTextCommandDefault{\flqq}{%
2172   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2173 \ProvideTextCommandDefault{\frqq}{%
2174   \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).

```

2175 \def\umlauthigh{%
2176   \def\bbl@umlauta##1{\leavevmode\bgroup%
2177     \accent\csname\fontencoding dpos\endcsname
2178     ##1\bbl@allowhyphens\egroup}%
2179   \let\bbl@umlaute\bbl@umlauta}
2180 \def\umlautlow{%
2181   \def\bbl@umlauta{\protect\lower@umlaut}}
2182 \def\umlautelow{%
2183   \def\bbl@umlaute{\protect\lower@umlaut}}
2184 \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.

```

2185 \expandafter\ifx\csname U@D\endcsname\relax
2186   \csname newdimen\endcsname\U@D
2187 \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.

```

2188 \def\lower@umlaut#1{%
2189   \leavevmode\bgroup
2190     \U@D 1ex%
2191     {\setbox\z@\hbox{%
2192       \char\csname\fontencoding dpos\endcsname}%
2193       \dimen@ -.45ex\advance\dimen@\ht\z@
2194       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2195     \accent\csname\fontencoding dpos\endcsname
2196     \fontdimen5\font\U@D #1%
2197   \egroup}

```


For all vowels we declare `\` to be a composite command which uses `\bbl@umlauta` or `\bbl@umlaute` to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package `fontenc` with option `OT1` is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but `babel` sets them for *all* languages – you may want to redefine `\bbl@umlauta` and/or `\bbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2198 \AtBeginDocument{%
2199   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2200   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2201   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2202   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{\i}}%
2203   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2204   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2205   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2206   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2207   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2208   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2209   \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}%

```

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

```

2210 \ifx\l@english\@undefined
2211   \chardef\l@english\z@
2212 \fi
2213 % The following is used to cancel rules in ini files (see Amharic).
2214 \ifx\l@unhyphenated\@undefined
2215   \newlanguage\l@unhyphenated
2216 \fi

```

7.13 Layout

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

```

2217 \bbl@trace{Bidi layout}
2218 \providecommand\IfBabelLayout[3]{#3}%
2219 \newcommand\BabelPatchSection[1]{%
2220   \@ifundefined{#1}{}{%
2221     \bbl@exp{\let<bbl@ss@#1>\<#1>}%
2222     \@namedef{#1}{%
2223       \ifstar{\bbl@presec@#1}%
2224       {\@dblarg{\bbl@presec@x{#1}}}}%
2225 \def\bbl@presec@x#1[#2]#3{%
2226   \bbl@exp{%
2227     \\\select@language@x{\bbl@main@language}%
2228     \\\bbl@cs{sspre@#1}%
2229     \\\bbl@cs{ss@#1}%
2230     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2231     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2232     \\\select@language@x{\language}}%
2233 \def\bbl@presec@s#1#2{%
2234   \bbl@exp{%
2235     \\\select@language@x{\bbl@main@language}%
2236     \\\bbl@cs{sspre@#1}%
2237     \\\bbl@cs{ss@#1}*%
2238     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2239     \\\select@language@x{\language}}%
2240 \IfBabelLayout{sectioning}%
2241   {\BabelPatchSection{part}%
2242    \BabelPatchSection{chapter}%
2243    \BabelPatchSection{section}%
2244    \BabelPatchSection{subsection}%
2245    \BabelPatchSection{subsubsection}%
2246    \BabelPatchSection{paragraph}%

```

```

2247 \BabelPatchSection{subparagraph}%
2248 \def\babel@toc#1{%
2249 \select@language{x{\bbl@main@language}}{}
2250 \IfBabelLayout{captions}%
2251 {\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.

```

2252 \bbl@trace{Input engine specific macros}
2253 \ifcase\bbl@engine
2254 \input txtbabel.def
2255 \or
2256 \input luababel.def
2257 \or
2258 \input xebabel.def
2259 \fi
2260 \providecommand\babelfont{%
2261 \bbl@error
2262 {This macro is available only in LuaLaTeX and XeLaTeX.}%
2263 {Consider switching to these engines.}}
2264 \providecommand\babelprehyphenation{%
2265 \bbl@error
2266 {This macro is available only in LuaLaTeX.}%
2267 {Consider switching to that engine.}}
2268 \ifx\babelposthyphenation\@undefined
2269 \let\babelposthyphenation\babelprehyphenation
2270 \let\babelpatterns\babelprehyphenation
2271 \let\babelcharproperty\babelprehyphenation
2272 \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.

```

2273 \bbl@trace{Creating languages and reading ini files}
2274 \let\bbl@extend@ini\@gobble
2275 \newcommand\babelprovide[2][]{%
2276 \let\bbl@savelangname\language
2277 \edef\bbl@savlocaleid{\the\localeid}%
2278 % Set name and locale id
2279 \edef\language{#2}%
2280 \bbl@id@assign
2281 % Initialize keys
2282 \bbl@foreach{captions,date,import,main,script,language,%
2283 hyphenrules,linebreaking,justification,mapfont,maparabic,%
2284 mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2285 Alph,labels,labels*,calendar,date}%
2286 {\bbl@csarg\let{KVP@##1}\@nnil}%
2287 \global\let\bbl@release@transforms\@empty
2288 \let\bbl@calendars\@empty
2289 \global\let\bbl@inidata\@empty
2290 \global\let\bbl@extend@ini\@gobble
2291 \gdef\bbl@key@list{;}%
2292 \bbl@forkv{#1}{%
2293 \in@{/}{##1}%
2294 \ifin@
2295 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2296 \bbl@renewinikey##1\@{##2}%
2297 \else

```

```

2298     \bbl@csarg\ifx{KVP@##1}\@nnil\else
2299     \bbl@error
2300     {Unknown key '##1' in \string\babelprovide}%
2301     {See the manual for valid keys}%
2302     \fi
2303     \bbl@csarg\def{KVP@##1}{##2}%
2304     \fi}%
2305 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2306 \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2307 % == init ==
2308 \ifx\bbl@screset\@undefined
2309     \bbl@ldfinit
2310 \fi
2311 % == date (as option) ==
2312 % \ifx\bbl@KVP@date\@nnil\else
2313 % \fi
2314 % ==
2315 \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2316 \ifcase\bbl@howloaded
2317     \let\bbl@lbkflag\@empty % new
2318 \else
2319     \ifx\bbl@KVP@hyphenrules\@nnil\else
2320         \let\bbl@lbkflag\@empty
2321     \fi
2322     \ifx\bbl@KVP@import\@nnil\else
2323         \let\bbl@lbkflag\@empty
2324     \fi
2325 \fi
2326 % == import, captions ==
2327 \ifx\bbl@KVP@import\@nnil\else
2328     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2329     {\ifx\bbl@initoload\relax
2330         \begingroup
2331         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2332         \bbl@input@texini{##2}%
2333         \endgroup
2334     \else
2335         \xdef\bbl@KVP@import{\bbl@initoload}%
2336     \fi}%
2337     {}%
2338     \let\bbl@KVP@date\@empty
2339 \fi
2340 \let\bbl@KVP@captions@\bbl@KVP@captions % TODO. A dirty hack
2341 \ifx\bbl@KVP@captions\@nnil
2342     \let\bbl@KVP@captions\bbl@KVP@import
2343 \fi
2344 % ==
2345 \ifx\bbl@KVP@transforms\@nnil\else
2346     \bbl@replace\bbl@KVP@transforms{ }{,}%
2347 \fi
2348 % == Load ini ==
2349 \ifcase\bbl@howloaded
2350     \bbl@provide@new{##2}%
2351 \else
2352     \bbl@ifblank{##1}%
2353     {}% With \bbl@load@basic below
2354     {\bbl@provide@renew{##2}}%
2355 \fi
2356 % Post tasks
2357 % -----
2358 % == subsequent calls after the first provide for a locale ==
2359 \ifx\bbl@inidata\@empty\else
2360     \bbl@extend@ini{##2}%

```

```

2361 \fi
2362 % == ensure captions ==
2363 \ifx\bbl@KVP@captions\@nnil\else
2364   \bbl@ifunset{bbl@extracaps@#2}%
2365     {\bbl@exp{\bbl@babelensure[exclude=\today]{#2}}}%
2366     {\bbl@exp{\bbl@babelensure[exclude=\today,
2367       include=\[bbl@extracaps@#2]]{#2}}}%
2368   \bbl@ifunset{bbl@ensure@language}%
2369     {\bbl@exp{%
2370       \\\DeclareRobustCommand\<bbl@ensure@language>[1]{%
2371         \\\foreignlanguage{language}%
2372         {###1}}}%
2373     }%
2374   \bbl@exp{%
2375     \\\bbl@tglobal\<bbl@ensure@language>%
2376     \\\bbl@tglobal\<bbl@ensure@language\space>%
2377 \fi
2378 % ==
2379 % At this point all parameters are defined if 'import'. Now we
2380 % execute some code depending on them. But what about if nothing was
2381 % imported? We just set the basic parameters, but still loading the
2382 % whole ini file.
2383 \bbl@load@basic{#2}%
2384 % == script, language ==
2385 % Override the values from ini or defines them
2386 \ifx\bbl@KVP@script\@nnil\else
2387   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2388 \fi
2389 \ifx\bbl@KVP@language\@nnil\else
2390   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2391 \fi
2392 \ifcase\bbl@engine\or
2393   \bbl@ifunset{bbl@chrng@language}{}%
2394   {\directlua{
2395     Babel.set_chranges_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2396 \fi
2397 % == onchar ==
2398 \ifx\bbl@KVP@onchar\@nnil\else
2399   \bbl@luahyphenate
2400   \bbl@exp{%
2401     \\\AddToHook{env/document/before}{\select@language{#2}}}%
2402   \directlua{
2403     if Babel.locale_mapped == nil then
2404       Babel.locale_mapped = true
2405       Babel.linebreaking.add_before(Babel.locale_map, 1)
2406       Babel.loc_to_scr = {}
2407       Babel.chr_to_loc = Babel.chr_to_loc or {}
2408     end
2409     Babel.locale_props[\the\localeid].letters = false
2410   }%
2411   \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2412   \ifin@
2413     \directlua{
2414       Babel.locale_props[\the\localeid].letters = true
2415     }%
2416   \fi
2417   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2418   \ifin@
2419     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2420       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2421     \fi
2422     \bbl@exp{\bbl@add\bbl@starthyphens
2423       {\bbl@patterns@lua{language}}}%

```

```

2424 % TODO - error/warning if no script
2425 \directlua{
2426   if Babel.script_blocks['\bbl@cl{sbc}'] then
2427     Babel.loc_to_scr[\the\localeid] =
2428       Babel.script_blocks['\bbl@cl{sbc}']
2429     Babel.locale_props[\the\localeid].lc = \the\localeid\space
2430     Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\languagename}\space
2431   end
2432 }%
2433 \fi
2434 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2435 \ifin@
2436   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys@\languagename}}{}%
2437   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs@\languagename}}{}%
2438   \directlua{
2439     if Babel.script_blocks['\bbl@cl{sbc}'] then
2440       Babel.loc_to_scr[\the\localeid] =
2441         Babel.script_blocks['\bbl@cl{sbc}']
2442     end}%
2443   \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2444     \AtBeginDocument{%
2445       \bbl@patchfont{\bbl@mapselect}}%
2446       {\selectfont}}%
2447     \def\bbl@mapselect{%
2448       \let\bbl@mapselect\relax
2449       \edef\bbl@prefontid{\fontid\font}}%
2450     \def\bbl@mapdir##1{%
2451       {\def\languagename{##1}%
2452        \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2453        \bbl@switchfont
2454        \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2455          \directlua{
2456            Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2457              [\bbl@prefontid] = \fontid\font\space}%
2458          \fi}}%
2459     \fi
2460     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
2461   \fi
2462   % TODO - catch non-valid values
2463 \fi
2464 % == mapfont ==
2465 % For bidi texts, to switch the font based on direction
2466 \ifx\bbl@KVP@mapfont\@nnil\else
2467   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2468   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
2469     mapfont. Use 'direction'.%
2470     {See the manual for details.}}}%
2471   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys@\languagename}}{}%
2472   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs@\languagename}}{}%
2473   \ifx\bbl@mapselect\undefined % TODO. See onchar.
2474     \AtBeginDocument{%
2475       \bbl@patchfont{\bbl@mapselect}}%
2476       {\selectfont}}%
2477     \def\bbl@mapselect{%
2478       \let\bbl@mapselect\relax
2479       \edef\bbl@prefontid{\fontid\font}}%
2480     \def\bbl@mapdir##1{%
2481       {\def\languagename{##1}%
2482        \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2483        \bbl@switchfont
2484        \directlua{Babel.fontmap
2485          [\the\csname bbl@wdir@##1\endcsname]%
2486          [\bbl@prefontid]=\fontid\font}}}%

```

```

2487 \fi
2488 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
2489 \fi
2490 % == Line breaking: intraspace, intrapenalty ==
2491 % For CJK, East Asian, Southeast Asian, if interspace in ini
2492 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2493 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2494 \fi
2495 \bbl@provide@intraspace
2496 % == Line breaking: CJK quotes == TODO -> @extras
2497 \ifcase\bbl@engine\or
2498 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2499 \ifin@
2500 \bbl@ifunset{\bbl@quote@\languagename}{}%
2501 {directlua{
2502 Babel.locale_props[\the\localeid].cjk_quotes = {}
2503 local cs = 'op'
2504 for c in string.utfvalues(
2505 [[\csname bbl@quote@\languagename\endcsname]]) do
2506 if Babel.cjk_characters[c].c == 'qu' then
2507 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2508 end
2509 cs = ( cs == 'op') and 'cl' or 'op'
2510 end
2511 }}%
2512 \fi
2513 \fi
2514 % == Line breaking: justification ==
2515 \ifx\bbl@KVP@justification\@nnil\else
2516 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2517 \fi
2518 \ifx\bbl@KVP@linebreaking\@nnil\else
2519 \bbl@xin@{,\bbl@KVP@linebreaking,}%
2520 {,elongated,kashida,cjk,padding,unhyphenated,}%
2521 \ifin@
2522 \bbl@csarg\xdef
2523 {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2524 \fi
2525 \fi
2526 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2527 \ifin@ \else \bbl@xin@{/k}{/\bbl@cl{lnbrk}} \fi
2528 \ifin@ \bbl@arabicjust \fi
2529 \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2530 \ifin@ \AtBeginDocument{\@nameuse{\bbl@tibetanjust}} \fi
2531 % == Line breaking: hyphenate.other.(locale|script) ==
2532 \ifx\bbl@lbkflag\@empty
2533 \bbl@ifunset{\bbl@hyotl@\languagename}{}%
2534 {\bbl@csarg\bbl@replace{\hyotl@\languagename}{ }{ },}%
2535 \bbl@startcommands*\languagename}{}%
2536 \bbl@csarg\bbl@foreach{\hyotl@\languagename}{%
2537 \ifcase\bbl@engine
2538 \ifnum##1<257
2539 \SetHyphenMap{\BabelLower{##1}{##1}}%
2540 \fi
2541 \else
2542 \SetHyphenMap{\BabelLower{##1}{##1}}%
2543 \fi}%
2544 \bbl@endcommands}%
2545 \bbl@ifunset{\bbl@hyots@\languagename}{}%
2546 {\bbl@csarg\bbl@replace{\hyots@\languagename}{ }{ },}%
2547 \bbl@csarg\bbl@foreach{\hyots@\languagename}{%
2548 \ifcase\bbl@engine
2549 \ifnum##1<257

```

```

2550         \global\lccode##1=##1\relax
2551     \fi
2552 \else
2553     \global\lccode##1=##1\relax
2554 \fi}}%
2555 \fi
2556 % == Counters: maparabic ==
2557 % Native digits, if provided in ini (TeX level, xe and lua)
2558 \ifcase\bbbl@engine\else
2559     \bbbl@ifunset{\bbbl@dgnat@\language\name}{}%
2560     {\expandafter\ifx\csname \bbbl@dgnat@\language\name\endcsname\@empty\else
2561         \expandafter\expandafter\expandafter
2562         \bbbl@setdigits\csname \bbbl@dgnat@\language\name\endcsname
2563         \ifx\bbbl@KVP@maparabic\@nnil\else
2564             \ifx\bbbl@latinarabic\undefined
2565                 \expandafter\let\expandafter\@arabic
2566                 \csname \bbbl@counter@\language\name\endcsname
2567             \else % ie, if layout=counters, which redefines \@arabic
2568                 \expandafter\let\expandafter\bbbl@latinarabic
2569                 \csname \bbbl@counter@\language\name\endcsname
2570             \fi
2571         \fi
2572     \fi}%
2573 \fi
2574 % == Counters: mapdigits ==
2575 % > luababel.def
2576 % == Counters: alph, Alph ==
2577 \ifx\bbbl@KVP@alph\@nnil\else
2578     \bbbl@exp{%
2579         \\\bbbl@add\<\bbbl@preextras@\language\name>{%
2580             \\\babel@save\\\@alph
2581             \let\\\@alph\<\bbbl@cntr@\bbbl@KVP@alph @\language\name>}}%
2582 \fi
2583 \ifx\bbbl@KVP@Alph\@nnil\else
2584     \bbbl@exp{%
2585         \\\bbbl@add\<\bbbl@preextras@\language\name>{%
2586             \\\babel@save\\\@Alph
2587             \let\\\@Alph\<\bbbl@cntr@\bbbl@KVP@Alph @\language\name>}}%
2588 \fi
2589 % == Calendars ==
2590 \ifx\bbbl@KVP@calendar\@nnil
2591     \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}%
2592 \fi
2593 \def\bbbl@tempe##1 ##2\@{% % Get first calendar
2594     \def\bbbl@tempa{##1}}%
2595     \bbbl@exp{\bbbl@tempe\bbbl@KVP@calendar\space\\\@}%
2596 \def\bbbl@tempe##1.##2.##3\@{%
2597     \def\bbbl@tempc{##1}%
2598     \def\bbbl@tempb{##2}}%
2599 \expandafter\bbbl@tempe\bbbl@tempa..\@
2600 \bbbl@csarg\edef{calpr@\language\name}{%
2601     \ifx\bbbl@tempc\@empty\else
2602         calendar=\bbbl@tempc
2603     \fi
2604     \ifx\bbbl@tempb\@empty\else
2605         ,variant=\bbbl@tempb
2606     \fi}%
2607 % == engine specific extensions ==
2608 % Defined in XXXbabel.def
2609 \bbbl@provide@extra{#2}%
2610 % == require.babel in ini ==
2611 % To load or reload the babel-*.tex, if require.babel in ini
2612 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body

```

```

2613 \bbl@ifunset{\bbl@rqtex@\languagename}{}%
2614 {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2615 \let\BabelBeforeIni\@gobbletwo
2616 \chardef\atcatcode=\catcode`\@
2617 \catcode`\@=11\relax
2618 \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2619 \catcode`\@=\atcatcode
2620 \let\atcatcode\relax
2621 \global\bbl@csarg\let{rqtex@\languagename}\relax
2622 \fi}%
2623 \bbl@foreach\bbl@calendars{%
2624 \bbl@ifunset{\bbl@ca##1}{%
2625 \chardef\atcatcode=\catcode`\@
2626 \catcode`\@=11\relax
2627 \InputIfFileExists{babel-ca-##1.tex}{}}%
2628 \catcode`\@=\atcatcode
2629 \let\atcatcode\relax}%
2630 }%
2631 \fi
2632 % == frenchspacing ==
2633 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2634 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2635 \ifin@
2636 \bbl@extras@wrap{\bbl@pre@fs}%
2637 {\bbl@pre@fs}%
2638 {\bbl@post@fs}%
2639 \fi
2640 % == transforms ==
2641 % > luababel.def
2642 % == main ==
2643 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2644 \let\languagename\bbl@savelangname
2645 \chardef\localeid\bbl@savelocaleid\relax
2646 \fi
2647 % == hyphenrules (apply if current) ==
2648 \ifx\bbl@KVP@hyphenrules\@nnil\else
2649 \ifnum\bbl@savelocaleid=\localeid
2650 \language\@nameuse{l@\languagename}%
2651 \fi
2652 \fi}

```

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

```

2653 \def\bbl@provide@new#1{%
2654 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2655 \@namedef{extras#1}{}%
2656 \@namedef{noextras#1}{}%
2657 \bbl@startcommands*{#1}{captions}%
2658 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2659 \def\bbl@tempb##1{% elt for \bbl@captionslist
2660 \ifx##1\@empty\else
2661 \bbl@exp{%
2662 \\\SetString\\##1{%
2663 \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2664 \expandafter\bbl@tempb
2665 \fi}%
2666 \expandafter\bbl@tempb\bbl@captionslist\@empty
2667 \else
2668 \ifx\bbl@initload\relax
2669 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2670 \else
2671 \bbl@read@ini{\bbl@initload}2% % Same
2672 \fi

```



```

2673 \fi
2674 \StartBabelCommands*{#1}{date}%
2675 \ifx\bbbl@KVP@date\@nnil
2676 \bbbl@exp{%
2677 \SetString\bbbl@nocaption{today}{#1today}}}%
2678 \else
2679 \bbbl@savetoday
2680 \bbbl@savestate
2681 \fi
2682 \bbbl@endcommands
2683 \bbbl@load@basic{#1}%
2684 % == hyphenmins == (only if new)
2685 \bbbl@exp{%
2686 \gdef\<#1hyphenmins>{%
2687 {\bbbl@ifunset{bbbl@lftm#1}{2}{\bbbl@cs{lftm#1}}}%
2688 {\bbbl@ifunset{bbbl@rgtm#1}{3}{\bbbl@cs{rgtm#1}}}%}%
2689 % == hyphenrules (also in renew) ==
2690 \bbbl@provide@hyphens{#1}%
2691 \ifx\bbbl@KVP@main\@nnil\else
2692 \expandafter\main@language\expandafter{#1}%
2693 \fi}
2694 %
2695 \def\bbbl@provide@renew#1{%
2696 \ifx\bbbl@KVP@captions\@nnil\else
2697 \StartBabelCommands*{#1}{captions}%
2698 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2699 \EndBabelCommands
2700 \fi
2701 \ifx\bbbl@KVP@date\@nnil\else
2702 \StartBabelCommands*{#1}{date}%
2703 \bbbl@savetoday
2704 \bbbl@savestate
2705 \EndBabelCommands
2706 \fi
2707 % == hyphenrules (also in new) ==
2708 \ifx\bbbl@lbfkflag\@empty
2709 \bbbl@provide@hyphens{#1}%
2710 \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.)

```

2711 \def\bbbl@load@basic#1{%
2712 \ifcase\bbbl@howloaded\or\or
2713 \ifcase\csname bbl@llevel@\language\endcsname
2714 \bbbl@csarg\let\lname@\language\relax
2715 \fi
2716 \fi
2717 \bbbl@ifunset{bbbl@lname@#1}%
2718 {\def\BabelBeforeIni##1##2{%
2719 \begingroup
2720 \let\bbbl@ini@captions@aux\@gobbletwo
2721 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2722 \bbbl@read@ini{##1}1%
2723 \ifx\bbbl@initoload\relax\endinput\fi
2724 \endgroup}%
2725 \begingroup % boxed, to avoid extra spaces:
2726 \ifx\bbbl@initoload\relax
2727 \bbbl@input@texini{##1}%
2728 \else
2729 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
2730 \fi
2731 \endgroup}%

```

2732 {}}

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```

2733 \def\bb1@provide@hyphens#1{%
2734   \@tempcnta\m@ne % a flag
2735   \ifx\bb1@KVP@hyphenrules\@nnil\else
2736     \bb1@replace\bb1@KVP@hyphenrules{ }{,}%
2737     \bb1@foreach\bb1@KVP@hyphenrules{%
2738       \ifnum\@tempcnta=\m@ne % if not yet found
2739         \bb1@ifsamestring{##1}{+}%
2740         {\bb1@carg\addlanguage{l@##1}}%
2741         {}%
2742         \bb1@ifunset{l@##1}% After a possible +
2743         {}%
2744         {\@tempcnta\@nameuse{l@##1}}%
2745       \fi}%
2746     \ifnum\@tempcnta=\m@ne
2747       \bb1@warning{%
2748         Requested 'hyphenrules=' for '\language' not found.\\%
2749         Using the default value. Reported}%
2750     \fi
2751   \fi
2752   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2753     \ifx\bb1@KVP@captions@\@nnil % TODO. Hackish. See above.
2754       \bb1@ifunset{\bb1@hyphr@#1}{}% use value in ini, if exists
2755       {\bb1@exp{\bb1@ifblank{\bb1@cs{hyphr@#1}}}%
2756        {}%
2757        {\bb1@ifunset{l@bb1@cl{hyphr}}}%
2758        {}% if hyphenrules found:
2759        {\@tempcnta\@nameuse{l@bb1@cl{hyphr}}}}}%
2760     \fi
2761   \fi
2762   \bb1@ifunset{l@#1}%
2763   {\ifnum\@tempcnta=\m@ne
2764     \bb1@carg\adddialect{l@#1}\language
2765   \else
2766     \bb1@carg\adddialect{l@#1}\@tempcnta
2767   \fi}%
2768   {\ifnum\@tempcnta=\m@ne\else
2769     \global\bb1@carg\chardef{l@#1}\@tempcnta
2770   \fi}}

```

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

```

2771 \def\bb1@input@texini#1{%
2772   \bb1@bsphack
2773   \bb1@exp{%
2774     \catcode`\\\%=14 \catcode`\\\%=0
2775     \catcode`\\\{=1 \catcode`\\\}=2
2776     \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2777     \catcode`\\\%=\the\catcode`\% \relax
2778     \catcode`\\\=\the\catcode`\% \relax
2779     \catcode`\\\{=\the\catcode`\% \relax
2780     \catcode`\\\}= \the\catcode`\% \relax}%
2781   \bb1@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 \bb1@read@ini.

```

2782 \def\bb1@inline#1\bb1@inline{%
2783   \@ifnextchar[\bb1@iniset{\@ifnextchar\bb1@iniskip\bb1@inistore}#1\@@% ]
2784   \def\bb1@iniset[#1]#2\@@{\def\bb1@section{#1}}
2785   \def\bb1@iniskip#1\@@{% if starts with ;
2786   \def\bb1@inistore#1=#2\@@{% full (default)

```

```

2787 \bbl@trim@def\bbl@tempa{#1}%
2788 \bbl@trim\toks@{#2}%
2789 \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2790 \ifin@ \else
2791   \bbl@xin@{,identification/include.}%
2792   {,\bbl@section/\bbl@tempa}%
2793   \ifin@ \edef\bbl@required@inis{\the\toks@}\fi
2794   \bbl@exp{%
2795     \\g@addto@macro\\bbl@inidata{%
2796       \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2797   \fi}
2798 \def\bbl@inistore@min#1=#2@@{% minimal (maybe set in \bbl@read@ini)
2799 \bbl@trim@def\bbl@tempa{#1}%
2800 \bbl@trim\toks@{#2}%
2801 \bbl@xin@{.identification.}{.\bbl@section.}%
2802 \ifin@
2803   \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2804     \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2805   \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.

```

2806 \def\bbl@loop@ini{%
2807   \loop
2808     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2809     \endlinechar\m@ne
2810     \read\bbl@readstream to \bbl@line
2811     \endlinechar\^^M
2812     \ifx\bbl@line\empty\else
2813       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2814     \fi
2815     \repeat}
2816 \ifx\bbl@readstream\undefined
2817   \csname newread\endcsname\bbl@readstream
2818 \fi
2819 \def\bbl@read@ini#1#2{%
2820   \global\let\bbl@extend@ini\gobble
2821   \openin\bbl@readstream=babel-#1.ini
2822   \ifeof\bbl@readstream
2823     \bbl@error
2824     {There is no ini file for the requested language\\%
2825       (#1: \language). Perhaps you misspelled it or your\\%
2826       installation is not complete.}%
2827     {Fix the name or reinstall babel.}%
2828   \else
2829     % == Store ini data in \bbl@inidata ==
2830     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2831     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2832     \bbl@info{Importing
2833       \ifcase#2font and identification \or basic \fi
2834       data for \language\\%
2835       from babel-#1.ini. Reported}%
2836     \ifnum#2=\z@
2837       \global\let\bbl@inidata\empty
2838       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2839     \fi
2840     \def\bbl@section{identification}%
2841     \let\bbl@required@inis\empty
2842     \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%

```

```

2843 \bbl@inistore load.level=#2\@@
2844 \bbl@loop@ini
2845 \ifx\bbl@required@inis\@empty\else
2846   \bbl@replace\bbl@required@inis{ }{,}%
2847   \bbl@foreach\bbl@required@inis{
2848     \openin\bbl@readstream=##1.ini
2849     \bbl@loop@ini}%
2850 \fi
2851 % == Process stored data ==
2852 \bbl@csarg\xdef{lini@\languagename}{#1}%
2853 \bbl@read@ini@aux
2854 % == 'Export' data ==
2855 \bbl@ini@exports{#2}%
2856 \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2857 \global\let\bbl@inidata\@empty
2858 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
2859 \bbl@toGLOBAL\bbl@ini@loaded
2860 \fi
2861 \closein\bbl@readstream}
2862 \def\bbl@read@ini@aux{%
2863   \let\bbl@savestrings\@empty
2864   \let\bbl@savetoday\@empty
2865   \let\bbl@savestate\@empty
2866   \def\bbl@elt##1##2##3{%
2867     \def\bbl@section{##1}%
2868     \in@{=date.}{=##1}% Find a better place
2869     \ifin@
2870       \bbl@ifunset{bbl@inikv@##1}%
2871       {\bbl@ini@calendar{##1}}%
2872       {}%
2873   \fi
2874   \in@{=identification/extension.}{=##1/##2}%
2875   \ifin@
2876     \bbl@ini@extension{##2}%
2877   \fi
2878   \bbl@ifunset{bbl@inikv@##1}{}%
2879   {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2880 \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.

```

2881 \def\bbl@extend@ini@aux#1{%
2882   \bbl@startcommands*{#1}{captions}%
2883   % Activate captions/... and modify exports
2884   \bbl@csarg\def{inikv@captions.licr}##1##2{%
2885     \setlocalecaption{#1}{##1}{##2}%
2886   \def\bbl@inikv@captions##1##2{%
2887     \bbl@ini@captions@aux{##1}{##2}%
2888   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2889   \def\bbl@exportkey##1##2##3{%
2890     \bbl@ifunset{bbl@kv@##2}{}%
2891     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2892       \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@kv@##2>}%
2893     \fi}}%
2894   % As with \bbl@read@ini, but with some changes
2895   \bbl@read@ini@aux
2896   \bbl@ini@exports\tw@
2897   % Update inidata@lang by pretending the ini is read.
2898   \def\bbl@elt##1##2##3{%
2899     \def\bbl@section{##1}%
2900     \bbl@inline##2=##3\bbl@inline}%
2901   \csname bbl@inidata@#1\endcsname
2902   \global\bbl@csarg\let{inidata@#1}\bbl@inidata

```

```

2903 \StartBabelCommands*{#1}{date}% And from the import stuff
2904 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2905 \bbl@savetoday
2906 \bbl@savedate
2907 \bbl@endcommands}

```

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

```

2908 \def\bbl@ini@calendar#1{%
2909 \lowercase{\def\bbl@tempa{=#1=}}%
2910 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2911 \bbl@replace\bbl@tempa{=date.}{}%
2912 \in@{.licr=}{#1=}%
2913 \ifin@
2914 \ifcase\bbl@engine
2915 \bbl@replace\bbl@tempa{.licr=}{}%
2916 \else
2917 \let\bbl@tempa\relax
2918 \fi
2919 \fi
2920 \ifx\bbl@tempa\relax\else
2921 \bbl@replace\bbl@tempa{=}{}%
2922 \ifx\bbl@tempa\@empty\else
2923 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2924 \fi
2925 \bbl@exp{%
2926 \def<\bbl@inikv@#1>####1####2{%
2927 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2928 \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).

```

2929 \def\bbl@renewinikey#1/#2\@#3{%
2930 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2931 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2932 \bbl@trim\toks@{#3}% value
2933 \bbl@exp{%
2934 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2935 \\g@addto@macro\\bbl@inidata{%
2936 \\\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.

```

2937 \def\bbl@exportkey#1#2#3{%
2938 \bbl@ifunset{\bbl@kv@#2}%
2939 {\bbl@csarg\gdef{#1@\language}\bbl@tempa{#3}}%
2940 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
2941 \bbl@csarg\gdef{#1@\language}\bbl@tempa{#3}}%
2942 \else
2943 \bbl@exp{\global\let<\bbl@#1@\language>\<\bbl@kv@#2>}%
2944 \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.

```

2945 \def\bbl@iniwarning#1{%
2946 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2947 {\bbl@warning{%
2948 From babel-\bbl@cs{lini@\language}.ini:\%
2949 \bbl@cs{@kv@identification.warning#1}\%
2950 Reported }}}
2951 %
2952 \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.

```

2953 \def\bbl@ini@extension#1{%
2954   \def\bbl@tempa{#1}%
2955   \bbl@replace\bbl@tempa{extension.}{}%
2956   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2957   \bbl@ifunset\bbl@info@#1{%
2958     {\bbl@csarg\xdef\info@#1}{ext/\bbl@tempa}%
2959     \bbl@exp{%
2960       \\g@addto@macro\\bbl@moreinfo{%
2961         \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
2962     {}%
2963 \let\bbl@moreinfo\@empty
2964 %
2965 \def\bbl@ini@exports#1{%
2966   % Identification always exported
2967   \bbl@iniwarning{}%
2968   \ifcase\bbl@engine
2969     \bbl@iniwarning{.pdflatex}%
2970   \or
2971     \bbl@iniwarning{.lualatex}%
2972   \or
2973     \bbl@iniwarning{.xelatex}%
2974   \fi%
2975   \bbl@exportkey{llevel}{identification.load.level}{}%
2976   \bbl@exportkey{elname}{identification.name.english}{}%
2977   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2978     {\csname bbl@elname@\language\endcsname}}%
2979   \bbl@exportkey{tbcpr}{identification.tag.bcp47}{}%
2980   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2981   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2982   \bbl@exportkey{esname}{identification.script.name}{}%
2983   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2984     {\csname bbl@esname@\language\endcsname}}%
2985   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2986   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2987   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2988   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2989   \bbl@moreinfo
2990   % Also maps bcp47 -> language
2991   \ifbbl@bcptoname
2992     \bbl@csarg\xdef\bcp@map@{\bbl@cl{tbcpr}}{\language}%
2993   \fi
2994   % Conditional
2995   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
2996     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2997     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2998     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2999     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3000     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3001     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3002     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3003     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3004     \bbl@exportkey{intsp}{typography.intraspace}{}%
3005     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3006     \bbl@exportkey{chrng}{characters.ranges}{}%
3007     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3008     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3009   \ifnum#1=\tw@         % only (re)new
3010     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3011     \bbl@toglobal\bbl@savetoday
3012     \bbl@toglobal\bbl@savestate
3013     \bbl@savestrings

```

```

3014 \fi
3015 \fi}

```

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

```

3016 \def\bbl@inikv#1#2{%      key=value
3017 \toks@{#2}%              This hides #'s from ini values
3018 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3019 \let\bbl@inikv@identification\bbl@inikv
3020 \let\bbl@inikv@date\bbl@inikv
3021 \let\bbl@inikv@typography\bbl@inikv
3022 \let\bbl@inikv@characters\bbl@inikv
3023 \let\bbl@inikv@numbers\bbl@inikv

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

3024 \def\bbl@inikv@counters#1#2{%
3025 \bbl@ifsamestring{#1}{digits}%
3026 {\bbl@error{The counter name 'digits' is reserved for mapping\%
3027 decimal digits}%
3028 {Use another name.}}%
3029 {}%
3030 \def\bbl@tempc{#1}%
3031 \bbl@trim@def{\bbl@tempb*}{#2}%
3032 \in@{.1$}{#1$}%
3033 \ifin@
3034 \bbl@replace\bbl@tempc{.1}{}%
3035 \bbl@csarg\protected@xdef{cnt@ \bbl@tempc @\language name}{%
3036 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3037 \fi
3038 \in@{.F.}{#1}%
3039 \ifin@\else\in@{.S.}{#1}\fi
3040 \ifin@
3041 \bbl@csarg\protected@xdef{cnt@#1@\language name}{\bbl@tempb*}%
3042 \else
3043 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3044 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3045 \bbl@csarg{\global\expandafter\let}{cnt@#1@\language name}\bbl@tempa
3046 \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.

```

3047 \ifcase\bbl@engine
3048 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3049 \bbl@ini@captions@aux{#1}{#2}}
3050 \else
3051 \def\bbl@inikv@captions#1#2{%
3052 \bbl@ini@captions@aux{#1}{#2}}
3053 \fi

```

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

```

3054 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3055 \bbl@replace\bbl@tempa{.template}{}%
3056 \def\bbl@toreplace{#1}{}%
3057 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3058 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3059 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3060 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3061 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3062 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3063 \ifin@

```

```

3064 \@nameuse{bbl@patch\bbl@tempa}%
3065 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3066 \fi
3067 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3068 \ifin@
3069 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3070 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3071     \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\language}%
3072     {[fnum@\bbl@tempa]}%
3073     {\@nameuse{bbl@\bbl@tempa fmt@\\language}}}%
3074 \fi}
3075 \def\bbl@ini@captions@aux#1#2{%
3076 \bbl@trim\def\bbl@tempa{#1}%
3077 \bbl@xin@{.template}{\bbl@tempa}%
3078 \ifin@
3079 \bbl@ini@captions@template{#2}\language
3080 \else
3081 \bbl@ifblank{#2}%
3082 {\bbl@exp{%
3083     \toks@{\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3084     {\bbl@trim\toks@{#2}}}%
3085 \bbl@exp{%
3086     \\\bbl@add\\bbl@savestrings{%
3087         \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3088     \toks@\expandafter{\bbl@captionslist}%
3089     \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}}%
3090 \ifin@else
3091 \bbl@exp{%
3092     \\\bbl@add\<bbl@extracaps@\language>{\<\bbl@tempa name>}%
3093     \\\bbl@tglobal\<bbl@extracaps@\language>}%
3094 \fi
3095 \fi}

```

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

```

3096 \def\bbl@list@the{%
3097 part,chapter,section,subsection,subsubsection,paragraph,%
3098 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3099 table,page,footnote,mpfootnote,mpfn}
3100 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3101 \bbl@ifunset{bbl@map@#1\language}%
3102 {\@nameuse{#1}}%
3103 {\@nameuse{bbl@map@#1\language}}}
3104 \def\bbl@inikv@labels#1#2{%
3105 \in@{.map}{#1}%
3106 \ifin@
3107 \ifx\bbl@KVP@labels\@nnil\else
3108 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3109 \ifin@
3110 \def\bbl@tempc{#1}%
3111 \bbl@replace\bbl@tempc{.map}{}%
3112 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3113 \bbl@exp{%
3114     \gdef\<bbl@map@\bbl@tempc @\language>%
3115     {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3116 \bbl@foreach\bbl@list@the{%
3117     \bbl@ifunset{the##1}{}%
3118     {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3119         \bbl@exp{%
3120             \\\bbl@sreplace\<the##1>%
3121             {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3122             \\\bbl@sreplace\<the##1>%
3123             {\<\@empty @\bbl@tempc>\<c##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3124     \expandafter\ifx\cname the##1\endcsname\bbl@tempd\else

```



```

3125         \toks@ \expandafter \expandafter \expandafter {%
3126             \csname the##1 \endcsname}%
3127         \expandafter \xdef \csname the##1 \endcsname {\the \toks@}%
3128     \fi}}%
3129 \fi
3130 \fi
3131 %
3132 \else
3133 %
3134 % The following code is still under study. You can test it and make
3135 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3136 % language dependent.
3137 \in@{enumerate.}{#1}%
3138 \ifin@
3139     \def \bbl@tempa{#1}%
3140     \bbl@replace \bbl@tempa {enumerate.}{}%
3141     \def \bbl@toreplace{#2}%
3142     \bbl@replace \bbl@toreplace {[ ]}{\nobreakspace{}}%
3143     \bbl@replace \bbl@toreplace {[ ]}{\csname the}%
3144     \bbl@replace \bbl@toreplace [ ]{\endcsname{}}%
3145     \toks@ \expandafter {\bbl@toreplace}%
3146     % TODO. Execute only once:
3147     \bbl@exp{%
3148         \\ \bbl@add \<extras \language name>{%
3149             \\ \babel@save \<labelenum \romannumeral \bbl@tempa>%
3150             \def \<labelenum \romannumeral \bbl@tempa> {\the \toks@}%
3151             \\ \bbl@to global \<extras \language name>%
3152         \fi
3153     \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.

```

3154 \def \bbl@chapttype{chapter}
3155 \ifx \@makechapterhead \@undefined
3156     \let \bbl@patchchapter \relax
3157 \else \ifx \thechapter \@undefined
3158     \let \bbl@patchchapter \relax
3159 \else \ifx \ps@headings \@undefined
3160     \let \bbl@patchchapter \relax
3161 \else
3162     \def \bbl@patchchapter{%
3163         \global \let \bbl@patchchapter \relax
3164         \gdef \bbl@chfmt{%
3165             \bbl@ifunset{\bbl@bbl@chapttype fmt@ \language name}%
3166                 {\@chapapp \space \thechapter}
3167                 {\@nameuse{\bbl@bbl@chapttype fmt@ \language name}}}
3168         \bbl@add \appendix {\def \bbl@chapttype{appendix}}% Not harmful, I hope
3169         \bbl@sreplace \ps@headings {\@chapapp \thechapter} {\bbl@chfmt}%
3170         \bbl@sreplace \chaptermark {\@chapapp \thechapter} {\bbl@chfmt}%
3171         \bbl@sreplace \@makechapterhead {\@chapapp \space \thechapter} {\bbl@chfmt}%
3172         \bbl@to global \appendix
3173         \bbl@to global \ps@headings
3174         \bbl@to global \chaptermark
3175         \bbl@to global \@makechapterhead}
3176     \let \bbl@patchappendix \bbl@patchchapter
3177 \fi \fi \fi
3178 \ifx \@part \@undefined
3179     \let \bbl@patchpart \relax
3180 \else
3181     \def \bbl@patchpart{%
3182         \global \let \bbl@patchpart \relax

```

```

3183 \gdef\bbl@partformat{%
3184 \bbl@ifunset\bbl@partfmt@\language\name}%
3185 {\partname\nobreakspace\thepart}
3186 {\@nameuse\bbl@partfmt@\language\name}}
3187 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3188 \bbl@tglobal\@part}
3189 \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

```

3190 \let\bbl@calendar\@empty
3191 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3192 \def\bbl@localedate#1#2#3#4{%
3193 \begingroup
3194 \edef\bbl@they{#2}%
3195 \edef\bbl@them{#3}%
3196 \edef\bbl@thed{#4}%
3197 \edef\bbl@tempe{%
3198 \bbl@ifunset\bbl@calpr@\language\name}{\bbl@cl{calpr}},%
3199 #1}%
3200 \bbl@replace\bbl@tempe{ }{}%
3201 \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3202 \bbl@replace\bbl@tempe{convert}{convert=}%
3203 \let\bbl@ld@calendar\@empty
3204 \let\bbl@ld@variant\@empty
3205 \let\bbl@ld@convert\relax
3206 \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld@##1}{##2}%
3207 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3208 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3209 \ifx\bbl@ld@calendar\@empty\else
3210 \ifx\bbl@ld@convert\relax\else
3211 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3212 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3213 \fi
3214 \fi
3215 \@nameuse\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3216 \edef\bbl@calendar{% Used in \month..., too
3217 \bbl@ld@calendar
3218 \ifx\bbl@ld@variant\@empty\else
3219 .\bbl@ld@variant
3220 \fi}%
3221 \bbl@cased
3222 {\@nameuse\bbl@date@\language\name @\bbl@calendar}%
3223 \bbl@they\bbl@them\bbl@thed}%
3224 \endgroup}
3225 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3226 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3227 \bbl@trim@def\bbl@tempa{#1.#2}%
3228 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3229 {\bbl@trim@def\bbl@tempa{#3}%
3230 \bbl@trim\toks@{#5}%
3231 \@temptokena\expandafter{\bbl@savedate}%
3232 \bbl@exp{% Reverse order - in ini last wins
3233 \def\\bbl@savedate{%
3234 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3235 \the\@temptokena}}}%
3236 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3237 {\lowercase{\def\bbl@tempb{#6}}}%
3238 \bbl@trim@def\bbl@toreplace{#5}%
3239 \bbl@TG@date
3240 \global\bbl@csarg\let{date@\language\name @\bbl@tempb}\bbl@toreplace
3241 \ifx\bbl@savetoday\@empty
3242 \bbl@exp{% TODO. Move to a better place.

```

```

3243      \\\AfterBabelCommands{%
3244      \def<\language name date>{\\protect<\language name date >}%
3245      \\\newcommand<\language name date >[4][ ]{%
3246      \\\bbl@usedategroupttrue
3247      <bbl@ensure@language name>{%
3248      \\\localedate[####1]{####2}{####3}{####4}}}%
3249      \def\\bbl@savetoday{%
3250      \\\SetString\\today{%
3251      <\language name date>[convert]%
3252      {\\the\year}{\\the\month}{\\the\day}}}%
3253      \fi}%
3254      {}}}

```

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).

```

3255 \let\bbl@calendar\@empty
3256 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3257   \@nameuse{bbl@ca#2}#1\@@}
3258 \newcommand\BabelDateSpace{\nobreakspace}
3259 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3260 \newcommand\BabelDated[1]{\number#1}
3261 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3262 \newcommand\BabelDateM[1]{\number#1}
3263 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3264 \newcommand\BabelDateMMMM[1]{%
3265   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3266 \newcommand\BabelDatey[1]{\number#1}%
3267 \newcommand\BabelDateyy[1]{%
3268   \ifnum#1<10 0\number#1 %
3269   \else\ifnum#1<100 \number#1 %
3270   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3271   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3272   \else
3273     \bbl@error
3274     {Currently two-digit years are restricted to the\
3275      range 0-9999.}%
3276     {There is little you can do. Sorry.}%
3277   \fi\fi\fi\fi}}
3278 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3279 \def\bbl@replace@finish@iii#1{%
3280   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3281   \def\bbl@TG@@date{%
3282     \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3283     \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3284     \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3285     \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3286     \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3287     \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3288     \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3289     \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3290     \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3291     \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3292     \bbl@replace\bbl@toreplace{[y]}{\bbl@datecntr[####1]}%
3293     \bbl@replace\bbl@toreplace{[m]}{\bbl@datecntr[####2]}%
3294     \bbl@replace\bbl@toreplace{[d]}{\bbl@datecntr[####3]}%
3295     \bbl@replace@finish@iii\bbl@toreplace}
3296 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3297 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3298 \let\bbl@release@transforms\@empty

```

```

3299 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3300 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3301 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3302   #1[#2]{#3}{#4}{#5}}
3303 \begingroup % A hack. TODO. Don't require an specific order
3304   \catcode\%=12
3305   \catcode\&=14
3306   \gdef\bbl@transforms#1#2#3{%&
3307     \directlua{
3308       local str = [==[#2]==]
3309       str = str:gsub('%.%d+%.%d+$', '')
3310       token.set_macro('babeltempa', str)
3311     }&
3312     \def\babeltempc{}&
3313     \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&
3314     \ifin@
3315       \bbl@xin@{:\babeltempa,},{,\bbl@KVP@transforms,}&
3316     \fi
3317     \ifin@
3318       \bbl@foreach\bbl@KVP@transforms{%&
3319         \bbl@xin@{:\babeltempa,},{,##1,}&
3320         \ifin@ & font:font:transform syntax
3321         \directlua{
3322           local t = {}
3323           for m in string.gmatch('##1'..' ':'(.)') do
3324             table.insert(t, m)
3325           end
3326           table.remove(t)
3327           token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3328         }&
3329         \fi}&
3330       \in@{.0$}{#2$}&
3331       \ifin@
3332         \directlua{%& (\attribute) syntax
3333           local str = string.match([[ \bbl@KVP@transforms]],
3334             '%([^(%-)%)^[^%)]-\babeltempa')
3335           if str == nil then
3336             token.set_macro('babeltempb', '')
3337           else
3338             token.set_macro('babeltempb', ',attribute=' .. str)
3339           end
3340         }&
3341       \toks@{#3}&
3342       \bbl@exp{%&
3343         \\g@addto@macro\\bbl@release@transforms{%&
3344           \relax & Closes previous \bbl@transforms@aux
3345           \\bbl@transforms@aux
3346           \\#1{label=\babeltempa\babeltempb\babeltempc}&
3347           {\language\the\toks@}}&
3348       \else
3349         \g@addto@macro\bbl@release@transforms{, {#3}}&
3350       \fi
3351     \fi}
3352 \endgroup

```

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

```

3353 \def\bbl@provide@lsys#1{%
3354   \bbl@ifunset{bbl@lname@#1}%
3355     {\bbl@load@info{#1}}%
3356     {}%
3357   \bbl@csarg\let{lsys@#1}\@empty
3358   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%

```

```

3359 \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3360 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3361 \bbl@ifunset{\bbl@lname@#1}{}%
3362 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3363 \ifcase\bbl@engine\or\or
3364 \bbl@ifunset{\bbl@prehc@#1}{}%
3365 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3366 {}%
3367 {\ifx\bbl@xenohyph\undefined
3368 \global\let\bbl@xenohyph\bbl@xenohyph@d
3369 \ifx\AtBeginDocument\@notprerr
3370 \expandafter\@secondoftwo % to execute right now
3371 \fi
3372 \AtBeginDocument{%
3373 \bbl@patchfont{\bbl@xenohyph}%
3374 \expandafter\select@language\expandafter{\language}%
3375 \fi}}%
3376 \fi
3377 \bbl@csarg\bbl@tglobal{lsys@#1}}
3378 \def\bbl@xenohyph@d{%
3379 \bbl@ifset{\bbl@prehc@language}%
3380 {\ifnum\hyphenchar\font=\defaultthyphenchar
3381 \iffontchar\font\bbl@c1{prehc}\relax
3382 \hyphenchar\font\bbl@c1{prehc}\relax
3383 \else\iffontchar\font"200B
3384 \hyphenchar\font"200B
3385 \else
3386 \bbl@warning
3387 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3388 in the current font, and therefore the hyphen\\%
3389 will be printed. Try changing the fontspec's\\%
3390 'HyphenChar' to another value, but be aware\\%
3391 this setting is not safe (see the manual).\\%
3392 Reported}%
3393 \hyphenchar\font\defaultthyphenchar
3394 \fi\fi
3395 \fi}%
3396 {\hyphenchar\font\defaultthyphenchar}}
3397 % \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).

```

3398 \def\bbl@load@info#1{%
3399 \def\BabelBeforeIni##1##2{%
3400 \begingroup
3401 \bbl@read@ini{##1}0%
3402 \endinput % babel- .tex may contain onlypreamble's
3403 \endgroup}% boxed, to avoid extra spaces:
3404 {\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 \TeX . Non-digits characters are kept. The first macro is the generic “localized” command.

```

3405 \def\bbl@setdigits#1#2#3#4#5{%
3406 \bbl@exp{%
3407 \def<\language digits>####1{ % ie, \langdigits
3408 \<bbl@digits@language>####1\\\nil}%
3409 \let<\bbl@cntr@digits@language>\<\language digits>%
3410 \def<\language counter>####1{ % ie, \langcounter
3411 \\\expandafter\<bbl@counter@language>%
3412 \\\csname c#####1\endcsname}%
3413 \def<\bbl@counter@language>####1{ % ie, \bbl@counter@lang

```



```

3470     {\bbl@cs{ctr@#1.F.\number#5#6#7#8@\languagename}}
3471 \def\bbl@alphnum@invalid#1{%
3472   \bbl@error{Alphabetic numeral too large (#1)}%
3473   {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.

```

3474 \def\bbl@localeinfo#1#2{%
3475   \bbl@ifunset{bbl@info@#2}{#1}%
3476   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3477    {\bbl@cs{csname bbl@info@#2\endcsname @\languagename}}}}
3478 \newcommand\localeinfo[1]{%
3479   \ifx*#1@empty % TODO. A bit hackish to make it expandable.
3480     \bbl@afterelse\bbl@localeinfo{}%
3481   \else
3482     \bbl@localeinfo
3483     {\bbl@error{I've found no info for the current locale.\%
3484               The corresponding ini file has not been loaded\%
3485               Perhaps it doesn't exist}%
3486      {See the manual for details.}}%
3487     {#1}%
3488   \fi}
3489 % \@namedef{bbl@info@name.locale}{lname}
3490 \@namedef{bbl@info@tag.ini}{lini}
3491 \@namedef{bbl@info@name.english}{elname}
3492 \@namedef{bbl@info@name.opentype}{lname}
3493 \@namedef{bbl@info@tag.bcp47}{tbc}
3494 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3495 \@namedef{bbl@info@tag.opentype}{lotf}
3496 \@namedef{bbl@info@script.name}{esname}
3497 \@namedef{bbl@info@script.name.opentype}{sname}
3498 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3499 \@namedef{bbl@info@script.tag.opentype}{sotf}
3500 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3501 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3502 % Extensions are dealt with in a special way
3503 % Now, an internal \LaTeX{} macro:
3504 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}

```

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

```

3505 <{*More package options}> \equiv
3506 \DeclareOption{ensureinfo=off}{}
3507 <{/More package options}>
3508 %
3509 \let\bbl@ensureinfo@gobble
3510 \newcommand\BabelEnsureInfo{%
3511   \ifx\InputIfFileExists\undefined\else
3512     \def\bbl@ensureinfo##1{%
3513       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
3514   \fi
3515   \bbl@foreach\bbl@loaded{%
3516     \def\languagename{##1}%
3517     \bbl@ensureinfo{##1}}}%
3518 \@ifpackagewith{babel}{ensureinfo=off}{}%
3519 {\AtEndOfPackage{% Test for plain.
3520   \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.

```

3521 \newcommand\getlocaleproperty{%
3522   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3523 \def\bbl@getproperty@s#1#2#3{%
3524   \let#1\relax

```

```

3525 \def\bbl@elt##1##2##3{%
3526 \bbl@ifsamestring{##1/##2}{#3}%
3527 {\providecommand#1{##3}%
3528 \def\bbl@elt####1####2####3{}}%
3529 {}}%
3530 \bbl@cs{inidata@#2}}%
3531 \def\bbl@getproperty@x#1#2#3{%
3532 \bbl@getproperty@s{#1}{#2}{#3}%
3533 \ifx#1\relax
3534 \bbl@error
3535 {Unknown key for locale '#2':\%
3536 #3\}%
3537 \string#1 will be set to \relax}%
3538 {Perhaps you misspelled it.}%
3539 \fi}
3540 \let\bbl@ini@loaded\@empty
3541 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

8 Adjusting the Babel behavior

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

```

3542 \newcommand\babeladjust[1]{% TODO. Error handling.
3543 \bbl@forkv{#1}{%
3544 \bbl@ifunset{bbl@ADJ@##1@##2}%
3545 {\bbl@cs{ADJ@##1}{##2}}%
3546 {\bbl@cs{ADJ@##1@##2}}}
3547 %
3548 \def\bbl@adjust@lua#1#2{%
3549 \ifvmode
3550 \ifnum\currentgrouplevel=\z@
3551 \directlua{ Babel.#2 }%
3552 \expandafter\expandafter\expandafter\@gobble
3553 \fi
3554 \fi
3555 {\bbl@error % The error is gobbled if everything went ok.
3556 {Currently, #1 related features can be adjusted only\%
3557 in the main vertical list.}%
3558 {Maybe things change in the future, but this is what it is.}}}
3559 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3560 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3561 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3562 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3563 \@namedef{bbl@ADJ@bidi.text@on}{%
3564 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3565 \@namedef{bbl@ADJ@bidi.text@off}{%
3566 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3567 \@namedef{bbl@ADJ@bidi.math@on}{%
3568 \let\bbl@noamsmath\@empty}
3569 \@namedef{bbl@ADJ@bidi.math@off}{%
3570 \let\bbl@noamsmath\relax}
3571 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3572 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3573 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3574 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3575 %
3576 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3577 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3578 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3579 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3580 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3581 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3582 \@namedef{bbl@ADJ@linebreak.cjk@off}{%

```



```

3583 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3584 \@namedef{bbl@ADJ@justify.arabic@on}{%
3585 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3586 \@namedef{bbl@ADJ@justify.arabic@off}{%
3587 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3588 %
3589 \def\bbl@adjust@layout#1{%
3590 \ifvmode
3591 #1%
3592 \expandafter\@gobble
3593 \fi
3594 {\bbl@error % The error is gobbled if everything went ok.
3595 {Currently, layout related features can be adjusted only\\%
3596 in vertical mode.}%
3597 {Maybe things change in the future, but this is what it is.}}}
3598 \@namedef{bbl@ADJ@layout.tabular@on}{%
3599 \ifnum\bbl@tabular@mode=\tw@
3600 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3601 \else
3602 \chardef\bbl@tabular@mode\@ne
3603 \fi}
3604 \@namedef{bbl@ADJ@layout.tabular@off}{%
3605 \ifnum\bbl@tabular@mode=\tw@
3606 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3607 \else
3608 \chardef\bbl@tabular@mode\z@
3609 \fi}
3610 \@namedef{bbl@ADJ@layout.lists@on}{%
3611 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3612 \@namedef{bbl@ADJ@layout.lists@off}{%
3613 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3614 %
3615 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3616 \bbl@bcpallowedtrue}
3617 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3618 \bbl@bcpallowedfalse}
3619 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3620 \def\bbl@bcp@prefix{#1}}
3621 \def\bbl@bcp@prefix{bcp47-}
3622 \@namedef{bbl@ADJ@autoload.options}#1{%
3623 \def\bbl@autoload@options{#1}}
3624 \let\bbl@autoload@bcptions\@empty
3625 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3626 \def\bbl@autoload@bcptions{#1}}
3627 \newif\ifbbl@bcptoname
3628 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3629 \bbl@bcptonametrue
3630 \BabelEnsureInfo}
3631 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3632 \bbl@bcptonamefalse}
3633 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3634 \directlua{ Babel.ignore_pre_char = function(node)
3635 return (node.lang == \the\csname l@nohyphenation\endcsname)
3636 end }}
3637 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3638 \directlua{ Babel.ignore_pre_char = function(node)
3639 return false
3640 end }}
3641 \@namedef{bbl@ADJ@select.write@shift}{%
3642 \let\bbl@restorelastskip\relax
3643 \def\bbl@savelastskip{%
3644 \let\bbl@restorelastskip\relax
3645 \ifvmode

```

```

3646 \ifdim\lastskip=\z@
3647 \let\bbl@restorelastskip\nobreak
3648 \else
3649 \bbl@exp{%
3650 \def\bbl@restorelastskip{%
3651 \skip@=\the\lastskip
3652 \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3653 \fi
3654 \fi}}
3655 \@namedef{bbl@ADJ@select.write@keep}{%
3656 \let\bbl@restorelastskip\relax
3657 \let\bbl@savelastskip\relax}
3658 \@namedef{bbl@ADJ@select.write@omit}{%
3659 \AddBabelHook{babel-select}{beforestart}{%
3660 \expandafter\babel@aux\expandafter\bbl@main@language}}}%
3661 \let\bbl@restorelastskip\relax
3662 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3663 \@namedef{bbl@ADJ@select.encoding@off}{%
3664 \let\bbl@encoding@select@off@empty}

```

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

```

3665 \ifx\directlua\@undefined\else
3666 \ifx\bbl@luapatterns\@undefined
3667 \input luababel.def
3668 \fi
3669 \fi

```

Continue with \LaTeX .

```

3670 </package | core>
3671 <*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.

```

3672 <(*More package options)> ≡
3673 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3674 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3675 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3676 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3677 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3678 </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.

```

3679 \bbl@trace{Cross referencing macros}
3680 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3681 \def\@newl@bel#1#2#3{%
3682 {\@safe@activetrue
3683 \bbl@ifunset{#1@#2}%
3684 \relax
3685 {\gdef\@multiplelabels{%
3686 \@latex@warning@no@line{There were multiply-defined labels}}%
3687 \@latex@warning@no@line{Label `#2' multiply defined}}%
3688 \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.

```

3689 \CheckCommand*\@testdef[3]{%
3690   \def\reserved@a{#3}%
3691   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3692   \else
3693     \@tempwattrue
3694   \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 `\@newlabel` 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.

```

3695 \def\@testdef#1#2#3{% TODO. With @samestring?
3696   \@safe@activetrue
3697   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3698   \def\bbl@tempb{#3}%
3699   \@safe@activetrue
3700   \ifx\bbl@tempa\relax
3701   \else
3702     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3703   \fi
3704   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3705   \ifx\bbl@tempa\bbl@tempb
3706   \else
3707     \@tempwattrue
3708   \fi}
3709 \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.

```

3710 \bbl@xin@{R}\bbl@opt@safe
3711 \ifin@
3712   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3713   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3714   {\expandafter\strip@prefix\meaning\ref}%
3715 \ifin@
3716   \bbl@redefine\@kernel@ref#1{%
3717     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3718   \bbl@redefine\@kernel@pageref#1{%
3719     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3720   \bbl@redefine\@kernel@sref#1{%
3721     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3722   \bbl@redefine\@kernel@spageref#1{%
3723     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3724   \else
3725     \bbl@redefineroobust\ref#1{%
3726       \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3727     \bbl@redefineroobust\pageref#1{%
3728       \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3729   \fi
3730 \else
3731   \let\org@ref\ref
3732   \let\org@pageref\pageref
3733 \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.

```

3734 \bbl@xin@{B}\bbl@opt@safe
3735 \ifin@
3736 \bbl@redefine\@citex[#1]#2{%
3737 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3738 \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.

```

3739 \AtBeginDocument{%
3740 \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.)

```

3741 \def\@citex[#1][#2]#3{%
3742 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3743 \org@@citex[#1][#2]{\@tempa}}%
3744 }{}}

```

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

```

3745 \AtBeginDocument{%
3746 \ifpackageloaded{cite}{%
3747 \def\@citex[#1]#2{%
3748 \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3749 }{}}

```

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

```

3750 \bbl@redefine\nocite#1{%
3751 \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

3752 \bbl@redefine\bibcite{%
3753 \bbl@cite@choice
3754 \bibcite}

```

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

```

3755 \def\bbl@bibcite#1#2{%
3756 \org@bibcite{#1}{\@safe@activesfalse#2}}

```

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

```

3757 \def\bbl@cite@choice{%
3758 \global\let\bibcite\bbl@bibcite
3759 \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}}%
3760 \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}}%
3761 \global\let\bbl@cite@choice\relax}

```

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

```

3762 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3763 \bbl@redefine\@bibitem#1{%
3764   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3765 \else
3766   \let\org@nocite\nocite
3767   \let\org@@citex\@citex
3768   \let\org@bibcite\@bibcite
3769   \let\org@@bibitem\@bibitem
3770 \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.

```

3771 \bbl@trace{Marks}
3772 \IfBabelLayout{sectioning}
3773   {\ifx\bbl@opt@headfoot\@nnil
3774     \g@addto@macro\resetactivechars{%
3775       \set@typeset@protect
3776       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3777       \let\protect\@noexpand
3778       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3779         \edef\thepage{%
3780           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3781       \fi}%
3782   \fi}
3783 {\ifbbl@single\else
3784   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3785   \markright#1{%
3786     \bbl@ifblank{#1}%
3787     {\org@markright{}}%
3788     {\toks@{#1}%
3789       \bbl@exp{%
3790         \\org@markright{\\protect\\foreignlanguage{\language}\language}%
3791         {\\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

3792   \ifx\@mkboth\markboth
3793     \def\bbl@tempc{\let\@mkboth\markboth}%
3794   \else
3795     \def\bbl@tempc{}%
3796   \fi
3797   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3798   \markboth#1#2{%
3799     \protected@edef\bbl@tempb##1{%
3800       \protect\foreignlanguage
3801       {\language}\protect\bbl@restore@actives##1}%
3802     \bbl@ifblank{#1}%
3803     {\toks@{}}%
3804     {\toks@\expandafter{\bbl@tempb{#1}}}%
3805     \bbl@ifblank{#2}%
3806     {\@temptokena{}}%
3807     {\@temptokena\expandafter{\bbl@tempb{#2}}}%

```

```

3808      \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3809      \bbl@tempc
3810      \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.

```

3811 \bbl@trace{Preventing clashes with other packages}
3812 \ifx\org@ref\@undefined\else
3813   \bbl@xin@{R}\bbl@opt@safe
3814   \ifin@
3815     \AtBeginDocument{%
3816       \@ifpackageloaded{ifthen}{%
3817         \bbl@redefine@long\ifthenelse#1#2#3{%
3818           \let\bbl@temp@pref\pageref
3819           \let\pageref\org@pageref
3820           \let\bbl@temp@ref\ref
3821           \let\ref\org@ref
3822           \@safe@activestru
3823           \org@ifthenelse{#1}%
3824             {\let\pageref\bbl@temp@pref
3825              \let\ref\bbl@temp@ref
3826              \@safe@activesfalse
3827              #2}%
3828             {\let\pageref\bbl@temp@pref
3829              \let\ref\bbl@temp@ref
3830              \@safe@activesfalse
3831              #3}%
3832           }%
3833         }{}%
3834       }
3835 \fi

```

8.3.2 varioref

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

```

3836 \AtBeginDocument{%
3837   \@ifpackageloaded{varioref}{%
3838     \bbl@redefine\@vpageref#1[#2]#3{%
3839       \@safe@activestru
3840       \org@@@vpageref{#1}[#2]{#3}%
3841       \@safe@activesfalse}%
3842     \bbl@redefine\vrefpagenum#1#2{%
3843       \@safe@activestru

```

```

3844 \org@vrefpagenum{#1}{#2}%
3845 \@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.

```

3846 \expandafter\def\csname Ref \endcsname#1{%
3847 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3848 }{}%
3849 }
3850 \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.

```

3851 \AtEndOfPackage{%
3852 \AtBeginDocument{%
3853 \ifpackageloaded{hhline}%
3854 {\expandafter\ifx\csname normal@char\string\endcsname\relax
3855 \else
3856 \makeatletter
3857 \def\@currname{hhline}\input{hhline.sty}\makeatother
3858 \fi}%
3859 {}}

```

`\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.

```

3860 \def\substitutefontfamily#1#2#3{%
3861 \lowercase{\immediate\openout15=#1#2.fd\relax}%
3862 \immediate\write15{%
3863 \string\ProvidesFile{#1#2.fd}%
3864 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3865 \space generated font description file]^{}
3866 \string\DeclareFontFamily{#1}{#2}{}}{}
3867 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}{}
3868 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}{}
3869 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}{}
3870 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}{}
3871 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}{}
3872 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}{}
3873 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}{}
3874 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}{}
3875 }%
3876 \closeout15
3877 }
3878 \@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

```
3879 \bbl@trace{Encoding and fonts}
3880 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3881 \newcommand\BabelNonText{TS1,T3,TS3}
3882 \let\org@TeX\TeX
3883 \let\org@LaTeX\LaTeX
3884 \let\ensureascii\@firstofone
3885 \AtBeginDocument{%
3886   \def\@elt#1{,#1,}%
3887   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3888   \let\@elt\relax
3889   \let\bbl@tempb\@empty
3890   \def\bbl@tempc{OT1}%
3891   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3892     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3893   \bbl@foreach\bbl@tempa{%
3894     \bbl@xin@{#1}{\BabelNonASCII}%
3895     \ifin@
3896       \def\bbl@tempb{#1}% Store last non-ascii
3897     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3898       \ifin@\else
3899         \def\bbl@tempc{#1}% Store last ascii
3900       \fi
3901     \fi}%
3902   \ifx\bbl@tempb\@empty\else
3903     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3904     \ifin@\else
3905       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3906     \fi
3907     \edef\ensureascii#1{%
3908       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3909     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3910     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3911   \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.

```
3912 \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.

```
3913 \AtBeginDocument{%
3914   \@ifpackageloaded{fontspec}%
3915   {\xdef\latinencoding{%
3916     \ifx\UTFencname\undefined
3917       EU\ifcase\bbl@engine\or2\or1\fi
3918     \else
3919       \UTFencname
3920     \fi}}%
3921   {\gdef\latinencoding{OT1}%
3922     \ifx\cf@encoding\bbl@t@one
3923       \xdef\latinencoding{\bbl@t@one}%
3924     \else
3925       \def\@elt#1{,#1,}%
3926       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3927       \let\@elt\relax
3928       \bbl@xin@{,T1,}\bbl@tempa
```



```

3929      \ifin@
3930      \xdef\latinencoding{\bbl@t@one}%
3931      \fi
3932      \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.

```

3933 \DeclareRobustCommand{\latintext}{%
3934   \fontencoding{\latinencoding}\selectfont
3935   \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.

```

3936 \ifx\@undefined\DeclareTextFontCommand
3937   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3938 \else
3939   \DeclareTextFontCommand{\textlatin}{\latintext}
3940 \fi

```

For several functions, we need to execute some code with `\selectfont`. With \LaTeX 2021-06-01, there is a hook for this purpose.

```

3941 \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.

```

3942 \bbl@trace{Loading basic (internal) bidi support}
3943 \ifodd\bbl@engine
3944 \else % TODO. Move to txtbabel
3945   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3946     \bbl@error
3947     {The bidi method 'basic' is available only in\%
3948      luatex. I'll continue with 'bidi=default', so\%
3949      expect wrong results}%
3950     {See the manual for further details.}%
3951     \let\bbl@beforeforeign\leavevmode
3952     \AtEndOfPackage{%
3953       \EnableBabelHook{babel-bidi}%
3954       \bbl@xebidipar}
3955   \fi\fi
3956   \def\bbl@loadxebidi#1{%
3957     \ifx\RTLfootnotetext\@undefined
3958       \AtEndOfPackage{%

```

```

3959      \EnableBabelHook{babel-bidi}%
3960      \bbl@loadfontspec % bidi needs fontspec
3961      \usepackage#1{bidi}}%
3962    \fi}
3963    \ifnum\bbl@bidimode>200
3964      \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3965        \bbl@tentative{bidi=bidi}
3966        \bbl@loadxebidi{}
3967      \or
3968        \bbl@loadxebidi{[rldocument]}
3969      \or
3970        \bbl@loadxebidi{}
3971      \fi
3972    \fi
3973  \fi
3974  % TODO? Separate:
3975  \ifnum\bbl@bidimode=\@ne
3976    \let\bbl@beforeforeign\leavevmode
3977    \ifodd\bbl@engine
3978      \newattribute\bbl@attr@dir
3979      \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3980      \bbl@exp{\output{\bodydir\pagedir\the\output}}
3981    \fi
3982    \AtEndOfPackage{%
3983      \EnableBabelHook{babel-bidi}%
3984      \ifodd\bbl@engine\else
3985        \bbl@xebidipar
3986      \fi}
3987  \fi

```

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

```

3988 \bbl@trace{Macros to switch the text direction}
3989 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3990 \def\bbl@rscripts{% TODO. Base on codes ??
3991   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3992   Old Hungarian,Lydian,Mandaean,Manichaean,%
3993   Meroitic Cursive,Meroitic,Old North Arabian,%
3994   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3995   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3996   Old South Arabian,}%
3997 \def\bbl@provide@dirs#1{%
3998   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3999   \ifin@
4000     \global\bbl@csarg\chardef{wdir@#1}\@ne
4001     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4002     \ifin@
4003       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4004     \fi
4005   \else
4006     \global\bbl@csarg\chardef{wdir@#1}\z@
4007   \fi
4008   \ifodd\bbl@engine
4009     \bbl@csarg\ifcase{wdir@#1}%
4010       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4011     \or
4012       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4013     \or
4014       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4015     \fi
4016   \fi}
4017 \def\bbl@switchdir{%
4018   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%

```

```

4019 \bbl@ifunset{\bbl@wdir\language}{\bbl@provide@dirs\language}}{}%
4020 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
4021 \def\bbl@setdirs#1{% TODO - math
4022 \ifcase\bbl@select@type % TODO - strictly, not the right test
4023 \bbl@bodydir{#1}%
4024 \bbl@pardir{#1}% <- Must precede \bbl@textdir
4025 \fi
4026 \bbl@textdir{#1}}
4027 % TODO. Only if \bbl@bidimode > 0?:
4028 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4029 \DisableBabelHook{babel-bidi}

```

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

```

4030 \ifodd\bbl@engine % luatex=1
4031 \else % pdftex=0, xetex=2
4032 \newcount\bbl@dirlevel
4033 \chardef\bbl@thetextdir\z@
4034 \chardef\bbl@thepardir\z@
4035 \def\bbl@textdir#1{%
4036 \ifcase#1\relax
4037 \chardef\bbl@thetextdir\z@
4038 \bbl@textdir@i\beginL\endL
4039 \else
4040 \chardef\bbl@thetextdir@ne
4041 \bbl@textdir@i\beginR\endR
4042 \fi}
4043 \def\bbl@textdir@i#1#2{%
4044 \ifhmode
4045 \ifnum\currentgrouplevel>\z@
4046 \ifnum\currentgrouplevel=\bbl@dirlevel
4047 \bbl@error{Multiple bidi settings inside a group}%
4048 {I'll insert a new group, but expect wrong results.}%
4049 \bgroup\aftergroup#2\aftergroup\egroup
4050 \else
4051 \ifcase\currentgrouptype\or % 0 bottom
4052 \aftergroup#2% 1 simple {}
4053 \or
4054 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4055 \or
4056 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4057 \or\or\or % vbox vtop align
4058 \or
4059 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4060 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4061 \or
4062 \aftergroup#2% 14 \begingroup
4063 \else
4064 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4065 \fi
4066 \fi
4067 \bbl@dirlevel\currentgrouplevel
4068 \fi
4069 #1%
4070 \fi}
4071 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4072 \let\bbl@bodydir@gobble
4073 \let\bbl@pagedir@gobble
4074 \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).

```

4075 \def\bbl@xebidipar{%
4076 \let\bbl@xebidipar\relax

```

```

4077 \TeXeTstate\@ne
4078 \def\bbl@xeverypar{%
4079   \ifcase\bbl@thepardir
4080     \ifcase\bbl@thetextdir\else\beginR\fi
4081   \else
4082     {\setbox\z@\lastbox\beginR\box\z@}%
4083   \fi}%
4084 \let\bbl@severypar\everypar
4085 \newtoks\everypar
4086 \everypar=\bbl@severypar
4087 \bbl@severypar{\bbl@xeverypar\the\everypar}}
4088 \ifnum\bbl@bidimode>200
4089   \let\bbl@textdir@i\@gobbletwo
4090   \let\bbl@xebidipar\@empty
4091   \AddBabelHook{bidi}{foreign}{%
4092     \def\bbl@tempa{\def\BabelText####1}%
4093     \ifcase\bbl@thetextdir
4094       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4095     \else
4096       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4097     \fi}
4098   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4099 \fi
4100 \fi

```

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

```

4101 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4102 \AtBeginDocument{%
4103   \ifx\pdfstringdefDisableCommands\@undefined\else
4104     \ifx\pdfstringdefDisableCommands\relax\else
4105       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4106     \fi
4107   \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`.

```

4108 \bbl@trace{Local Language Configuration}
4109 \ifx\loadlocalcfg\@undefined
4110   \ifpackagewith{babel}{noconfigs}%
4111     {\let\loadlocalcfg\@gobble}%
4112   {\def\loadlocalcfg#1{%
4113     \InputIfFileExists{#1.cfg}%
4114     {\typeout{*****^^J%
4115               * Local config file #1.cfg used^^J%
4116             *}}%
4117     \@empty}}
4118 \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).

```

4119 \bbl@trace{Language options}
4120 \let\bbl@afterlang\relax
4121 \let\BabelModifiers\relax
4122 \let\bbl@loaded\@empty

```

```

4123 \def\bbl@load@language#1{%
4124   \InputIfFileExists{#1.ldf}%
4125   {\edef\bbl@loaded{\CurrentOption
4126     \ifx\bbl@loaded\empty\else,\bbl@loaded\fi}%
4127     \expandafter\let\expandafter\bbl@afterlang
4128     \csname\CurrentOption.ldf-h@k\endcsname
4129     \expandafter\let\expandafter\BabelModifiers
4130     \csname\bbl@mod@\CurrentOption\endcsname}%
4131   {\bbl@error{%
4132     Unknown option '\CurrentOption'. Either you misspelled it\\%
4133     or the language definition file \CurrentOption.ldf was not found}}%
4134     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4135     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4136     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.

```

4137 \def\bbl@try@load@lang#1#2#3{%
4138   \IfFileExists{\CurrentOption.ldf}%
4139   {\bbl@load@language{\CurrentOption}}%
4140   {#1\bbl@load@language{#2}#3}}
4141 %
4142 \DeclareOption{hebrew}{%
4143   \input{rlbabel.def}%
4144   \bbl@load@language{hebrew}}
4145 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4146 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4147 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4148 \DeclareOption{polutonikogreek}{%
4149   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4150 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4151 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4152 \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.

```

4153 \ifx\bbl@opt@config\@nnil
4154   \@ifpackagewith{babel}{noconfigs}{}%
4155   {\InputIfFileExists{bblopts.cfg}%
4156     {\typeout{*****^J%
4157       * Local config file bblopts.cfg used^^J%
4158       *}}%
4159     {}}%
4160 \else
4161   \InputIfFileExists{\bbl@opt@config.cfg}%
4162   {\typeout{*****^J%
4163     * Local config file \bbl@opt@config.cfg used^^J%
4164     *}}%
4165   {\bbl@error{%
4166     Local config file '\bbl@opt@config.cfg' not found}%
4167     Perhaps you misspelled it.}}%
4168 \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.

```

4169 \ifx\bbl@opt@main\@nnil

```

```

4170 \ifnum\bbbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4171 \let\bbbl@tempb\@empty
4172 \edef\bbbl@tempa{\@classoptionslist,\bbbl@language@opts}%
4173 \bbbl@foreach\bbbl@tempa{\edef\bbbl@tempb{#1,\bbbl@tempb}}%
4174 \bbbl@foreach\bbbl@tempb{% \bbbl@tempb is a reversed list
4175 \ifx\bbbl@opt@main\@nnil % ie, if not yet assigned
4176 \ifodd\bbbl@iniflag % = *=
4177 \IfFileExists{babel-#1.tex}{\def\bbbl@opt@main{#1}}}%
4178 \else % n +=
4179 \IfFileExists{#1.ldf}{\def\bbbl@opt@main{#1}}}%
4180 \fi
4181 \fi}%
4182 \fi
4183 \else
4184 \bbbl@info{Main language set with 'main='. Except if you have%%
4185 problems, prefer the default mechanism for setting%%
4186 the main language. Reported}
4187 \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).

```

4188 \ifx\bbbl@opt@main\@nnil\else
4189 \bbbl@ncarg\let\bbbl@loadmain{ds@\bbbl@opt@main}%
4190 \expandafter\let\csname ds@\bbbl@opt@main\endcsname\relax
4191 \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.

```

4192 \bbbl@foreach\bbbl@language@opts{%
4193 \def\bbbl@tempa{#1}%
4194 \ifx\bbbl@tempa\bbbl@opt@main\else
4195 \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4196 \bbbl@ifunset{ds@#1}%
4197 {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4198 {}%
4199 \else % + * (other = ini)
4200 \DeclareOption{#1}{%
4201 \bbbl@ldfinit
4202 \babelprovide[import]{#1}%
4203 \bbbl@afterldf{}}%
4204 \fi
4205 \fi}%
4206 \bbbl@foreach\@classoptionslist{%
4207 \def\bbbl@tempa{#1}%
4208 \ifx\bbbl@tempa\bbbl@opt@main\else
4209 \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4210 \bbbl@ifunset{ds@#1}%
4211 {\IfFileExists{#1.ldf}%
4212 {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4213 {}}%
4214 {}%
4215 \else % + * (other = ini)
4216 \IfFileExists{babel-#1.tex}%
4217 {\DeclareOption{#1}{%
4218 \bbbl@ldfinit
4219 \babelprovide[import]{#1}%
4220 \bbbl@afterldf{}}}%
4221 {}%
4222 \fi
4223 \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):

```

4224 \def\AfterBabelLanguage#1{%
4225   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4226 \DeclareOption*{}
4227 \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.

```

4228 \bbl@trace{Option 'main'}
4229 \ifx\bbl@opt@main\@nnil
4230   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4231   \let\bbl@tempc\@empty
4232   \edef\bbl@templ{\bbl@loaded,}
4233   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4234   \bbl@for\bbl@tempb\bbl@tempa{%
4235     \edef\bbl@tempd{\bbl@tempb,%}
4236     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4237     \bbl@xin{\bbl@tempd}{\bbl@templ}%
4238     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4239   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4240   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4241   \ifx\bbl@tempb\bbl@tempc\else
4242     \bbl@warning{%
4243       Last declared language option is '\bbl@tempc',\%
4244       but the last processed one was '\bbl@tempb'.\%
4245       The main language can't be set as both a global\%
4246       and a package option. Use 'main=\bbl@tempc' as\%
4247       option. Reported}
4248   \fi
4249 \else
4250   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4251     \bbl@ldfinit
4252     \let\CurrentOption\bbl@opt@main
4253     \bbl@exp{% \bbl@opt@provide = empty if *
4254       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4255     \bbl@afterldf{}
4256     \DeclareOption{\bbl@opt@main}{}
4257   \else % case 0,2 (main is ldf)
4258     \ifx\bbl@loadmain\relax
4259       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4260     \else
4261       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4262     \fi
4263     \ExecuteOptions{\bbl@opt@main}
4264     \@namedef{ds@\bbl@opt@main}{}%
4265   \fi
4266   \DeclareOption*{}
4267   \ProcessOptions*
4268 \fi
4269 \def\AfterBabelLanguage{%
4270   \bbl@error
4271   {Too late for \string\AfterBabelLanguage}%
4272   {Languages have been loaded, so I can do nothing}}

```

In order to catch the case where the user didn't specify a language we check whether \bbl@main@language, has become defined. If not, the nil language is loaded.

```

4273 \ifx\bbl@main@language\undefined
4274   \bbl@info{%

```

```

4275   You haven't specified a language as a class or package\\%
4276   option. I'll load 'nil'. Reported}
4277   \bbl@load@language{nil}
4278 \fi
4279 </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

```

4280 <*kernel>
4281 \let\bbl@onlyswitch\@empty
4282 \input babel.def
4283 \let\bbl@onlyswitch\@undefined
4284 </kernel>
4285 <*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.

```

4286 <<Make sure ProvidesFile is defined>>
4287 \ProvidesFile{hyphen.cfg}[<<date>>] <<version>> Babel hyphens]
4288 \xdef\bbl@format{\jobname}
4289 \def\bbl@version{<<version>>}
4290 \def\bbl@date{<<date>>}
4291 \ifx\AtBeginDocument\@undefined
4292   \def\@empty{}
4293 \fi
4294 <<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.

```

4295 \def\process@line#1#2 #3 #4 {%
4296   \ifx=#1%
4297     \process@synonym{#2}%
4298   \else
4299     \process@language{#1#2}{#3}{#4}%
4300   \fi
4301   \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.

```

4302 \toks@{}
4303 \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.

```

4304 \def\process@synonym#1{%
4305   \ifnum\last@language=\m@ne
4306     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4307   \else
4308     \expandafter\chardef\csname l@#1\endcsname\last@language
4309     \wlog{\string\l@#1=\string\language\the\last@language}%
4310     \expandafter\let\csname #1hyphenmins\endcsname
4311       \csname\language\hyphenmins\endcsname
4312     \let\bbl@elt\relax
4313     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4314   \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 `\langhyphenmins` 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.

```

4315 \def\process@language#1#2#3{%
4316   \expandafter\addlanguage\csname l@#1\endcsname
4317   \expandafter\language\csname l@#1\endcsname
4318   \edef\language{#1}%
4319   \bbl@hook@everylanguage{#1}%
4320   % > luatex
4321   \bbl@get@enc#1::\@@@
4322   \begingroup
4323     \lefthyphenmin\m@ne
4324     \bbl@hook@loadpatterns{#2}%
4325     % > luatex
4326     \ifnum\lefthyphenmin=\m@ne
4327     \else
4328       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4329         \the\lefthyphenmin\the\righthyphenmin}%
4330     \fi
4331   \endgroup
4332   \def\bbl@tempa{#3}%
4333   \ifx\bbl@tempa\empty\else
4334     \bbl@hook@loadexceptions{#3}%
4335     % > luatex
4336   \fi
4337   \let\bbl@elt\relax

```

```

4338 \edef\bb1@languages{%
4339   \bb1@languages\bb1@elt{#1}{\the\language}{#2}{\bb1@tempa}}%
4340 \ifnum\the\language=\z@
4341   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4342     \set@hyphenmins\tw@\thr@\relax
4343   \else
4344     \expandafter\expandafter\expandafter\set@hyphenmins
4345       \csname #1hyphenmins\endcsname
4346   \fi
4347   \the\toks@
4348   \toks@{}%
4349 \fi}

```

\bb1@get@enc The macro \bb1@get@enc extracts the font encoding from the language name and stores it in \bb1@hyph@enc. It uses delimited arguments to achieve this.

```

4350 \def\bb1@get@enc#1:#2:#3\@@{\def\bb1@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.

```

4351 \def\bb1@hook@everylanguage#1{%
4352 \def\bb1@hook@loadpatterns#1{\input #1\relax}
4353 \let\bb1@hook@loadexceptions\bb1@hook@loadpatterns
4354 \def\bb1@hook@loadkernel#1{%
4355   \def\addlanguage{\csname newlanguage\endcsname}%
4356   \def\adddialect##1##2{%
4357     \global\chardef##1##2\relax
4358     \wlog{\string##1 = a dialect from \string\language##2}}%
4359   \def\iflanguage##1{%
4360     \expandafter\ifx\csname l@##1\endcsname\relax
4361       \nolannerr{##1}%
4362     \else
4363       \ifnum\csname l@##1\endcsname=\language
4364         \expandafter\expandafter\expandafter\@firstoftwo
4365       \else
4366         \expandafter\expandafter\expandafter\@secondoftwo
4367       \fi
4368     \fi}%
4369   \def\providehyphenmins##1##2{%
4370     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4371       \namedef{##1hyphenmins}{##2}%
4372     \fi}%
4373   \def\set@hyphenmins##1##2{%
4374     \lefthyphenmin##1\relax
4375     \righthyphenmin##2\relax}%
4376   \def\selectlanguage{%
4377     \errhelp{Selecting a language requires a package supporting it}%
4378     \errmessage{Not loaded}}%
4379   \let\foreignlanguage\selectlanguage
4380   \let\otherlanguage\selectlanguage
4381   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4382   \def\bb1@usehooks##1##2{% TODO. Temporary!!
4383     \def\setlocale{%
4384       \errhelp{Find an armchair, sit down and wait}%
4385       \errmessage{Not yet available}}%
4386     \let\uselocale\setlocale
4387     \let\locale\setlocale
4388     \let\selectlocale\setlocale
4389     \let\localename\setlocale
4390     \let\textlocale\setlocale
4391     \let\textlanguage\setlocale
4392     \let\languagetext\setlocale}
4393 \begingroup

```

```

4394 \def\AddBabelHook#1#2{%
4395   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4396     \def\next{\toks1}%
4397   \else
4398     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4399   \fi
4400   \next}
4401 \ifx\directlua\@undefined
4402   \ifx\XeTeXinputencoding\@undefined\else
4403     \input xebabel.def
4404   \fi
4405 \else
4406   \input luababel.def
4407 \fi
4408 \openin1 = babel-\bbl@format.cfg
4409 \ifeof1
4410 \else
4411   \input babel-\bbl@format.cfg\relax
4412 \fi
4413 \closein1
4414 \endgroup
4415 \bbl@hook@loadkernel{switch.def}

```

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

```

4416 \openin1 = language.dat

```

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

```

4417 \def\language{english}%
4418 \ifeof1
4419   \message{I couldn't find the file language.dat,\space
4420     I will try the file hyphen.tex}
4421   \input hyphen.tex\relax
4422   \chardef\l@english\z@
4423 \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.

```

4424 \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.

```

4425 \loop
4426   \endlinechar\m@ne
4427   \read1 to \bbl@line
4428   \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.

```

4429   \if T\ifeof1F\fi T\relax
4430   \ifx\bbl@line\@empty\else
4431     \edef\bbl@line{\bbl@line\space\space\space}%
4432     \expandafter\process@line\bbl@line\relax
4433   \fi
4434 \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.

```

4435 \begingroup
4436 \def\bbl@elt#1#2#3#4{%

```

```

4437 \global\language=#2\relax
4438 \gdef\language{#1}%
4439 \def\bbl@elt##1##2##3##4{}}%
4440 \bbl@languages
4441 \endgroup
4442 \fi
4443 \closein1

```

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

```

4444 \if\the\toks@/\else
4445 \errhelp{language.dat loads no language, only synonyms}
4446 \errmessage{Orphan language synonym}
4447 \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.

```

4448 \let\bbl@line\@undefined
4449 \let\process@line\@undefined
4450 \let\process@synonym\@undefined
4451 \let\process@language\@undefined
4452 \let\bbl@get@enc\@undefined
4453 \let\bbl@hyph@enc\@undefined
4454 \let\bbl@tempa\@undefined
4455 \let\bbl@hook@loadkernel\@undefined
4456 \let\bbl@hook@everylanguage\@undefined
4457 \let\bbl@hook@loadpatterns\@undefined
4458 \let\bbl@hook@loadexceptions\@undefined
4459 \</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].

```

4460 <{*More package options}> ≡
4461 \chardef\bbl@bidimode\z@
4462 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4463 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4464 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4465 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4466 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4467 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4468 <{/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 (and mostly useless) message.

```

4469 <{*Font selection}> ≡
4470 \bbl@trace{Font handling with fontspec}
4471 \ifx\ExplSyntaxOn\@undefined\else
4472 \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4473 \in@{, #1, }{, no-script, language-not-exist,}%
4474 \ifin@ \else\bbl@tempfs@nx{#1}{#2}\fi}
4475 \def\bbl@fs@warn@nx#1#2#3{%
4476 \in@{, #1, }{, no-script, language-not-exist,}%
4477 \ifin@ \else\bbl@tempfs@nx{#1}{#2}{#3}\fi}
4478 \def\bbl@loadfontspec{%
4479 \let\bbl@loadfontspec\relax

```

```

4480 \ifx\fontspec\@undefined
4481 \usepackage{fontspec}%
4482 \fi}%
4483 \fi
4484 \@onlypreamble\babelfont
4485 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4486 \bbl@foreach{#1}{%
4487 \expandafter\ifx\csname babel-##1\endcsname\relax
4488 \IfFileExists{babel-##1.tex}%
4489 {\babelprovide{##1}}%
4490 {}%
4491 \fi}%
4492 \edef\bbl@tempa{#1}%
4493 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4494 \bbl@loadfontspec
4495 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4496 \bbl@bblfont}
4497 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4498 \bbl@ifunset{\bbl@tempb family}%
4499 {\bbl@providfam{\bbl@tempb}}%
4500 {}%
4501 % For the default font, just in case:
4502 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4503 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4504 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4505 \bbl@exp{%
4506 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4507 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4508 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4509 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4510 \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:

```

4511 \def\bbl@providfam#1{%
4512 \bbl@exp{%
4513 \\\newcommand\<#1default>{}% Just define it
4514 \\\bbl@add@list\\bbl@font@fams{#1}%
4515 \\\DeclareRobustCommand\<#1family>{%
4516 \\\not@math@alphabet\<#1family>\relax
4517 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4518 \\\fontfamily\<#1default>%
4519 \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4520 \\\selectfont}%
4521 \\\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.

```

4522 \def\bbl@nostdfont#1{%
4523 \bbl@ifunset{\bbl@WFF@\f@family}%
4524 {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
4525 \bbl@infowarn{The current font is not a babel standard family:\\%
4526 #1%
4527 \fontname\font\\%
4528 There is nothing intrinsically wrong with this warning, and\\%
4529 you can ignore it altogether if you do not need these\\%
4530 families. But if they are used in the document, you should be\\%
4531 aware 'babel' will not set Script and Language for them, so\\%
4532 you may consider defining a new family with \string\babelfont.\\%
4533 See the manual for further details about \string\babelfont.\\%
4534 Reported}}
4535 {}}%
4536 \gdef\bbl@switchfont{%
4537 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4538 \bbl@exp{% eg Arabic -> arabic

```

```

4539 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4540 \bbl@foreach\bbl@font@fams{%
4541 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4542 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4543 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4544 {}% 123=F - nothing!
4545 {\bbl@exp{% 3=T - from generic
4546 \global\let\<bbl@##1dflt@\language>%
4547 \<bbl@##1dflt@>}}}%
4548 {\bbl@exp{% 2=T - from script
4549 \global\let\<bbl@##1dflt@\language>%
4550 \<bbl@##1dflt@*\bbl@tempa>}}}%
4551 {}}% 1=T - language, already defined
4552 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4553 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4554 \bbl@ifunset{\bbl@##1dflt@\language}%
4555 {\bbl@cs{famrst@##1}%
4556 \global\bbl@csarg\let{famrst@##1}\relax}%
4557 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4558 \\bbl@add\\originalTeX{%
4559 \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4560 \<##1default>\<##1family>{##1}}}%
4561 \\bbl@font@set\<bbl@##1dflt@\language>% the main part!
4562 \<##1default>\<##1family>}}}%
4563 \bbl@ifrestoring{{\bbl@tempa}}}%

```

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

```

4564 \ifx\fb@family\undefined\else % if latex
4565 \ifcase\bbl@engine % if pdftex
4566 \let\bbl@ckeckstdfonts\relax
4567 \else
4568 \def\bbl@ckeckstdfonts{%
4569 \begingroup
4570 \global\let\bbl@ckeckstdfonts\relax
4571 \let\bbl@tempa\@empty
4572 \bbl@foreach\bbl@font@fams{%
4573 \bbl@ifunset{\bbl@##1dflt@}%
4574 {\@nameuse{##1family}%
4575 \bbl@csarg\gdef{WFF@\fb@family}{}}% Flag
4576 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \fb@family\\}%
4577 \space\space\fontname\font\\}%
4578 \bbl@csarg\xdef{##1dflt@}{\fb@family}%
4579 \expandafter\xdef\csname ##1default\endcsname{\fb@family}}%
4580 {}}%
4581 \ifx\bbl@tempa\@empty\else
4582 \bbl@infowarn{The following font families will use the default\\%
4583 settings for all or some languages:\\%
4584 \bbl@tempa
4585 There is nothing intrinsically wrong with it, but\\%
4586 'babel' will no set Script and Language, which could\\%
4587 be relevant in some languages. If your document uses\\%
4588 these families, consider redefining them with \string\babelfont.\\%
4589 Reported}%
4590 \fi
4591 \endgroup}
4592 \fi
4593 \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.

```

4594 \def\bb1@font@set#1#2#3{% eg \bb1@rmdflt@lang \rmdefault \rmfamily
4595   \bb1@xin@{<>}{#1}%
4596   \ifin@
4597     \bb1@exp{\bb1@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4598   \fi
4599   \bb1@exp{%
4600     \def\#2{#1}%          eg, \rmdefault{\bb1@rmdflt@lang}
4601     \bb1@ifsamestring{#2}{\f@family}%
4602     {\#3%
4603       \bb1@ifsamestring{\f@series}{\bfdefault}{\bfseries}}}%
4604     \let\bb1@tempa\relax}%
4605   {}}
4606 %   TODO - next should be global?, but even local does its job. I'm
4607 %   still not sure -- must investigate:
4608 \def\bb1@fontspec@set#1#2#3#4{% eg \bb1@rmdflt@lang fnt-opt fnt-nme \xxfamily
4609   \let\bb1@tempa\bb1@mapselect
4610   \let\bb1@mapselect\relax
4611   \let\bb1@temp@fam#4%          eg, '\rmfamily', to be restored below
4612   \let#4@empty %          Make sure \renewfontfamily is valid
4613   \bb1@exp{%
4614     \let\bb1@temp@pfam<\bb1@stripslash#4\space>% eg, '\rmfamily '
4615     \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bb1@cl{sname}}}%
4616     {\newfontscript{\bb1@cl{sname}}{\bb1@cl{sotf}}}%
4617     \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bb1@cl{lname}}}%
4618     {\newfontlanguage{\bb1@cl{lname}}{\bb1@cl{lotf}}}%
4619     \let\bb1@tempfs@nx<__fontspec_warning:nx>%
4620     \let<__fontspec_warning:nx>\bb1@fs@warn@nx
4621     \let\bb1@tempfs@nxx<__fontspec_warning:nxx>%
4622     \let<__fontspec_warning:nxx>\bb1@fs@warn@nxx
4623     \renewfontfamily\#4%
4624     [\bb1@cl{lsys},#2]{#3}% ie \bb1@exp{..}{#3}
4625   \bb1@exp{%
4626     \let<__fontspec_warning:nx>\bb1@tempfs@nx
4627     \let<__fontspec_warning:nxx>\bb1@tempfs@nxx}%
4628   \begingroup
4629     #4%
4630     \xdef#1{\f@family}%          eg, \bb1@rmdflt@lang{FreeSerif(0)}
4631   \endgroup
4632   \let#4\bb1@temp@fam
4633   \bb1@exp{\let<\bb1@stripslash#4\space>\bb1@temp@pfam
4634   \let\bb1@mapselect\bb1@tempa}%

```

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

```

4635 \def\bb1@font@rst#1#2#3#4{%
4636   \bb1@csarg\def{famrst@#4}{\bb1@font@set{#1}#2#3}}

```

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

```

4637 \def\bb1@font@fams{rm,sf,tt}
4638 <{/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.

```

4639 <{*Footnote changes}> ≡
4640 \bb1@trace{Bidi footnotes}
4641 \ifnum\bb1@bidimode>\z@
4642   \def\bb1@footnote#1#2#3{%
4643     \@ifnextchar[%

```

```

4644      {\bbl@footnote@o{#1}{#2}{#3}}%
4645      {\bbl@footnote@x{#1}{#2}{#3}}}
4646 \long\def\bbl@footnote@x#1#2#3#4{%
4647   \bgroup
4648   \select@language@x{\bbl@main@language}%
4649   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4650   \egroup}
4651 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4652   \bgroup
4653   \select@language@x{\bbl@main@language}%
4654   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4655   \egroup}
4656 \def\bbl@footnotetext#1#2#3{%
4657   \@ifnextchar[%
4658     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4659     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4660 \long\def\bbl@footnotetext@x#1#2#3#4{%
4661   \bgroup
4662   \select@language@x{\bbl@main@language}%
4663   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4664   \egroup}
4665 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4666   \bgroup
4667   \select@language@x{\bbl@main@language}%
4668   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4669   \egroup}
4670 \def\BabelFootnote#1#2#3#4{%
4671   \ifx\bbl@fn@footnote\undefined
4672     \let\bbl@fn@footnote\footnote
4673   \fi
4674   \ifx\bbl@fn@footnotetext\undefined
4675     \let\bbl@fn@footnotetext\footnotetext
4676   \fi
4677   \bbl@ifblank{#2}%
4678   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4679    \@namedef{\bbl@stripslash#1text}%
4680    {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4681   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4682    \@namedef{\bbl@stripslash#1text}%
4683    {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4684 \fi
4685 <\/Footnote changes>

```

Now, the code.

```

4686 <*xetex>
4687 \def\BabelStringsDefault{unicode}
4688 \let\xebbl@stop\relax
4689 \AddBabelHook{xetex}{encodedcommands}{%
4690   \def\bbl@tempa{#1}%
4691   \ifx\bbl@tempa\empty
4692     \XeTeXinputencoding"bytes"%
4693   \else
4694     \XeTeXinputencoding"#1"%
4695   \fi
4696   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4697 \AddBabelHook{xetex}{stopcommands}{%
4698   \xebbl@stop
4699   \let\xebbl@stop\relax}
4700 \def\bbl@intraspace#1 #2 #3\@{%
4701   \bbl@csarg\gdef{xeisp@languagename}%
4702   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4703 \def\bbl@intrapenalty#1\@{%
4704   \bbl@csarg\gdef{xeipn@languagename}%

```



```

4705 {\XeTeXlinebreakpenalty #1\relax}}
4706 \def\bbl@provide@intraspace{%
4707 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4708 \ifin@ \else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}}\fi
4709 \ifin@
4710 \bbl@ifunset{\bbl@intsp@{language}}}%
4711 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\empty\else
4712 \ifx\bbl@KVP@intraspace@nnil
4713 \bbl@exp{%
4714 \bbl@intraspace\bbl@cl{intsp}\@@}%
4715 \fi
4716 \ifx\bbl@KVP@intrapenalty@nnil
4717 \bbl@intrapenalty0\@@
4718 \fi
4719 \fi
4720 \ifx\bbl@KVP@intraspace@nnil\else % We may override the ini
4721 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4722 \fi
4723 \ifx\bbl@KVP@intrapenalty@nnil\else
4724 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4725 \fi
4726 \bbl@exp{%
4727 % TODO. Execute only once (but redundant):
4728 \bbl@add\<extras\language>{%
4729 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4730 \<bbl@xeisp@{language}>%
4731 \<bbl@xeipn@{language}>}%
4732 \bbl@tglobal\<extras\language>%
4733 \bbl@add\<noextras\language>{%
4734 \XeTeXlinebreaklocale ""}%
4735 \bbl@tglobal\<noextras\language>}%
4736 \ifx\bbl@ispacesize@undefined
4737 \gdef\bbl@ispacesize{\bbl@cl{xeisp}}}%
4738 \ifx\AtBeginDocument\@notprerr
4739 \expandafter\@secondoftwo % to execute right now
4740 \fi
4741 \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4742 \fi}%
4743 \fi}
4744 \ifx\DisableBabelHook\@undefined\endinput\fi
4745 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4746 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
4747 \DisableBabelHook{babel-fontspec}
4748 <<Font selection>>
4749 \def\bbl@provide@extra#1{}
4750 </xetex>

```

12.2 Layout

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

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

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

```

4751 <*xetex | texxet>
4752 \providecommand\bbl@provide@intraspace{}
4753 \bbl@trace{Redefinitions for bidi layout}
4754 \def\bbl@sspre@caption{%
4755 \bbl@exp{\everyhbox{\bbl@texmdir\bbl@cs{wdir@\bbl@main@language}}}}
4756 \ifx\bbl@opt@layout@nnil\else % if layout=..
4757 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4758 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}

```

```

4759 \ifx\bb1@beforeforeign\leavevmode % A poor test for bidi=
4760 \def\hangfrom#1{%
4761 \setbox\@tempboxa\hbox{{#1}}%
4762 \hangindent\ifcase\bb1@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4763 \noindent\box\@tempboxa}
4764 \def\raggedright{%
4765 \let\\\@centercr
4766 \bb1@startskip\z@skip
4767 \@rightskip\flushglue
4768 \bb1@endskip\@rightskip
4769 \parindent\z@
4770 \parfillskip\bb1@startskip}
4771 \def\raggedleft{%
4772 \let\\\@centercr
4773 \bb1@startskip\flushglue
4774 \bb1@endskip\z@skip
4775 \parindent\z@
4776 \parfillskip\bb1@endskip}
4777 \fi
4778 \IfBabelLayout{lists}
4779 {\bb1@sreplace\list
4780 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bb1@listleftmargin}%
4781 \def\bb1@listleftmargin{%
4782 \ifcase\bb1@thepardir\leftmargin\else\rightmargin\fi}%
4783 \ifcase\bb1@engine
4784 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4785 \def\p@enumiii{\p@enumii}\theenumii{}\fi
4786 \fi
4787 \bb1@sreplace\@verbatim
4788 {\leftskip\@totalleftmargin}%
4789 {\bb1@startskip\textwidth
4790 \advance\bb1@startskip-\linewidth}%
4791 \bb1@sreplace\@verbatim
4792 {\rightskip\z@skip}%
4793 {\bb1@endskip\z@skip}}%
4794 {}
4795 \IfBabelLayout{contents}
4796 {\bb1@sreplace\@dottedtocline{\leftskip}{\bb1@startskip}%
4797 \bb1@sreplace\@dottedtocline{\rightskip}{\bb1@endskip}}
4798 {}
4799 \IfBabelLayout{columns}
4800 {\bb1@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bb1@outputbox}%
4801 \def\bb1@outputbox#1{%
4802 \hb@xt@\textwidth{%
4803 \hskip\columnwidth
4804 \hfil
4805 {\normalcolor\vrule \@width\columnseprule}%
4806 \hfil
4807 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4808 \hskip-\textwidth
4809 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4810 \hskip\columnsep
4811 \hskip\columnwidth}}}%
4812 {}
4813 <<Footnote changes>>
4814 \IfBabelLayout{footnotes}%
4815 {\BabelFootnote\footnote\languagename{}\{}}%
4816 \BabelFootnote\localfootnote\languagename{}\{}}%
4817 \BabelFootnote\mainfootnote{}\{}}%
4818 {}

```

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.

```

4819 \IfBabelLayout{counters*}%
4820   {\bbl@add\bbl@opt@layout{.counters.}%
4821     \AddToHook{shipout/before}{%
4822       \let\bbl@tempa\babelsublr
4823       \let\babelsublr\@firstofone
4824       \let\bbl@save@thepage\thepage
4825       \protected@edef\thepage{\thepage}%
4826       \let\babelsublr\bbl@tempa}%
4827     \AddToHook{shipout/after}{%
4828       \let\thepage\bbl@save@thepage}}}%
4829 \IfBabelLayout{counters}%
4830   {\let\bbl@latinarabic=\@arabic
4831     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4832     \let\bbl@asciroman=\@roman
4833     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4834     \let\bbl@asciiRoman=\@Roman
4835     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
4836 \fi % end if layout
4837 </xetex | texxet>

```

12.3 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff.

```

4838 <*texxet>
4839 \def\bbl@provide@extra#1{%
4840   % == auto-select encoding ==
4841   \ifx\bbl@encoding@select@off\@empty\else
4842     \bbl@ifunset{\bbl@encoding@#1}%
4843     {\def\@elt##1{,##1,}%
4844       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4845       \count\@z@
4846       \bbl@foreach\bbl@tempe{%
4847         \def\bbl@tempd{##1}% Save last declared
4848         \advance\count\@ne}%
4849       \ifnum\count\@z@>\@ne
4850         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4851         \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4852         \bbl@replace\bbl@tempa{ }{,}%
4853         \global\bbl@csarg\let{encoding@#1}\@empty
4854         \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
4855         \ifin@ \else % if main encoding included in ini, do nothing
4856           \let\bbl@tempb\relax
4857           \bbl@foreach\bbl@tempa{%
4858             \ifx\bbl@tempb\relax
4859               \bbl@xin@{,##1,}{,\bbl@tempe,}%
4860               \ifin@\def\bbl@tempb{##1}\fi
4861             \fi}%
4862           \ifx\bbl@tempb\relax\else
4863             \bbl@exp{%
4864               \global\<\bbl@add>\<\bbl@preextras@#1>{\<\bbl@encoding@#1>}%
4865               \gdef\<\bbl@encoding@#1>{%
4866                 \\\babel@save\\\f@encoding
4867                 \\\bbl@add\\\originalTeX{\\\selectfont}%
4868                 \\\fontencoding{\bbl@tempb}%
4869                 \\\selectfont}}}%
4870             \fi
4871           \fi
4872         \fi}%
4873     }%
4874   \fi}
4875 </texxet>

```

12.4 LuaTeX

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

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

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

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

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

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

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the base option); (2) at `hyphen.cfg`, to modify some macros; (3) in the middle of `plain.def` and `babel.sty`, by `babel.def`, with the commands and other definitions for luatex (eg, `\babelpatterns`).

```
4876 (*luatex)
4877 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4878 \bbl@trace{Read language.dat}
4879 \ifx\bbl@readstream\@undefined
4880 \csname newread\endcsname\bbl@readstream
4881 \fi
4882 \begingroup
4883 \toks@{}
4884 \count@ \z@ % 0=start, 1=0th, 2=normal
4885 \def\bbl@process@line#1#2 #3 #4 {%
4886 \ifx=#1%
4887 \bbl@process@synonym{#2}%
4888 \else
4889 \bbl@process@language{#1#2}{#3}{#4}%
4890 \fi
4891 \ignorespaces}
4892 \def\bbl@manylang{%
4893 \ifnum\bbl@last>\@ne
4894 \bbl@info{Non-standard hyphenation setup}%
4895 \fi
4896 \let\bbl@manylang\relax}
4897 \def\bbl@process@language#1#2#3{%
4898 \ifcase\count@
4899 \@ifundefined{zth@#1}{\count@ \tw@}{\count@ \@ne}%
4900 \or
4901 \count@ \tw@
4902 \fi
4903 \ifnum\count@=\tw@
4904 \expandafter\addlanguage\csname l@#1\endcsname
```

```

4905     \language\allocationnumber
4906     \chardef\bbl@last\allocationnumber
4907     \bbl@manylang
4908     \let\bbl@elt\relax
4909     \xdef\bbl@languages{%
4910         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4911     \fi
4912     \the\toks@
4913     \toks@{}}
4914 \def\bbl@process@synonym@aux#1#2{%
4915     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4916     \let\bbl@elt\relax
4917     \xdef\bbl@languages{%
4918         \bbl@languages\bbl@elt{#1}{#2}{}}}%
4919 \def\bbl@process@synonym#1{%
4920     \ifcase\count@
4921     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4922     \or
4923     \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4924     \else
4925     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4926     \fi}
4927 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4928     \chardef\l@english\z@
4929     \chardef\l@USenglish\z@
4930     \chardef\bbl@last\z@
4931     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4932     \gdef\bbl@languages{%
4933         \bbl@elt{english}{0}{hyphen.tex}}%
4934         \bbl@elt{USenglish}{0}{}}
4935 \else
4936     \global\let\bbl@languages@format\bbl@languages
4937     \def\bbl@elt#1#2#3#4{% Remove all except language 0
4938         \ifnum#2>\z@\else
4939             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4940             \fi}%
4941     \xdef\bbl@languages{\bbl@languages}%
4942     \fi
4943     \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4944     \bbl@languages
4945     \openin\bbl@readstream=language.dat
4946     \ifeof\bbl@readstream
4947         \bbl@warning{I couldn't find language.dat. No additional\%
4948             patterns loaded. Reported}%
4949     \else
4950         \loop
4951             \endlinechar\m@ne
4952             \read\bbl@readstream to \bbl@line
4953             \endlinechar\^^M
4954             \if T\ifeof\bbl@readstream F\fi T\relax
4955             \ifx\bbl@line\@empty\else
4956                 \edef\bbl@line{\bbl@line\space\space\space}%
4957                 \expandafter\bbl@process@line\bbl@line\relax
4958             \fi
4959         \repeat
4960     \fi
4961     \closein\bbl@readstream
4962 \endgroup
4963 \bbl@trace{Macros for reading patterns files}
4964 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4965 \ifx\babelcatcodetablenum\@undefined
4966     \ifx\newcatcodetable\@undefined
4967         \def\babelcatcodetablenum{5211}

```

```

4968 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4969 \else
4970 \newcatcodetable\babelcatcodetablenum
4971 \newcatcodetable\bbl@pattcodes
4972 \fi
4973 \else
4974 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4975 \fi
4976 \def\bbl@luapatterns#1#2{%
4977 \bbl@get@enc#1::\@@@
4978 \setbox\z@\hbox\bgroup
4979 \begingroup
4980 \savecatcodetable\babelcatcodetablenum\relax
4981 \initcatcodetable\bbl@pattcodes\relax
4982 \catcodetable\bbl@pattcodes\relax
4983 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4984 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~ =13
4985 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4986 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4987 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4988 \catcode\`=12 \catcode\'=12 \catcode\"=12
4989 \input #1\relax
4990 \catcodetable\babelcatcodetablenum\relax
4991 \endgroup
4992 \def\bbl@tempa{#2}%
4993 \ifx\bbl@tempa\empty\else
4994 \input #2\relax
4995 \fi
4996 \egroup}%
4997 \def\bbl@patterns@lua#1{%
4998 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4999 \csname l@#1\endcsname
5000 \edef\bbl@tempa{#1}%
5001 \else
5002 \csname l@#1:\f@encoding\endcsname
5003 \edef\bbl@tempa{#1:\f@encoding}%
5004 \fi\relax
5005 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5006 \@ifundefined{bbl@hyphendata@the\language}%
5007 {\def\bbl@elt##1##2##3##4{%
5008 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5009 \def\bbl@tempb{##3}%
5010 \ifx\bbl@tempb\empty\else % if not a synonymous
5011 \def\bbl@tempc{##3}{##4}%
5012 \fi
5013 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5014 \fi}%
5015 \bbl@languages
5016 \@ifundefined{bbl@hyphendata@the\language}%
5017 {\bbl@info{No hyphenation patterns were set for\%
5018 language '\bbl@tempa'. Reported}}%
5019 {\expandafter\expandafter\expandafter\bbl@luapatterns
5020 \csname bbl@hyphendata@the\language\endcsname}}}%
5021 \endinput\fi
5022 % Here ends \ifx\AddBabelHook\@undefined
5023 % A few lines are only read by hyphen.cfg
5024 \ifx\DisableBabelHook\@undefined
5025 \AddBabelHook{luatex}{everylanguage}{%
5026 \def\process@language##1##2##3{%
5027 \def\process@line####1####2 ####3 ####4 {}}%
5028 \AddBabelHook{luatex}{loadpatterns}{%
5029 \input #1\relax
5030 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname

```

```

5031      {{#1}}{}}
5032 \AddBabelHook{luatex}{loadexceptions}{%
5033   \input #1\relax
5034   \def\bbl@tempb##1##2{{##1}{#1}}%
5035   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5036     {\expandafter\expandafter\expandafter\bbl@tempb
5037       \csname bbl@hyphendata@the\language\endcsname}}
5038 \endinput\fi
5039 % Here stops reading code for hyphen.cfg
5040 % The following is read the 2nd time it's loaded
5041 \begingroup % TODO - to a lua file
5042 \catcode`\%=12
5043 \catcode`\'=12
5044 \catcode`\%=12
5045 \catcode`\:=12
5046 \directlua{
5047   Babel = Babel or {}
5048   function Babel.bytes(line)
5049     return line:gsub("(.)",
5050       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5051   end
5052   function Babel.begin_process_input()
5053     if luatexbase and luatexbase.add_to_callback then
5054       luatexbase.add_to_callback('process_input_buffer',
5055         Babel.bytes, 'Babel.bytes')
5056     else
5057       Babel.callback = callback.find('process_input_buffer')
5058       callback.register('process_input_buffer', Babel.bytes)
5059     end
5060   end
5061   function Babel.end_process_input ()
5062     if luatexbase and luatexbase.remove_from_callback then
5063       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5064     else
5065       callback.register('process_input_buffer', Babel.callback)
5066     end
5067   end
5068   function Babel.addpatterns(pp, lg)
5069     local lg = lang.new(lg)
5070     local pats = lang.patterns(lg) or ''
5071     lang.clear_patterns(lg)
5072     for p in pp:gmatch('[^%s]+') do
5073       ss = ''
5074       for i in string.utfcharacters(p:gsub('%d', '')) do
5075         ss = ss .. '%d?' .. i
5076       end
5077       ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5078       ss = ss:gsub('%.%d%?$', '%%.')
5079       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5080       if n == 0 then
5081         tex.sprint(
5082           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5083           .. p .. [[{ }]])
5084         pats = pats .. ' ' .. p
5085       else
5086         tex.sprint(
5087           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5088           .. p .. [[{ }]])
5089       end
5090     end
5091     lang.patterns(lg, pats)
5092   end
5093   Babel.characters = Babel.characters or {}

```

```

5094 Babel.ranges = Babel.ranges or {}
5095 function Babel.hlist_has_bidi(head)
5096   local has_bidi = false
5097   local ranges = Babel.ranges
5098   for item in node.traverse(head) do
5099     if item.id == node.id'glyph' then
5100       local itemchar = item.char
5101       local chardata = Babel.characters[itemchar]
5102       local dir = chardata and chardata.d or nil
5103       if not dir then
5104         for nn, et in ipairs(ranges) do
5105           if itemchar < et[1] then
5106             break
5107           elseif itemchar <= et[2] then
5108             dir = et[3]
5109             break
5110           end
5111         end
5112       end
5113       if dir and (dir == 'al' or dir == 'r') then
5114         has_bidi = true
5115       end
5116     end
5117   end
5118   return has_bidi
5119 end
5120 function Babel.set_chranges_b (script, chrng)
5121   if chrng == '' then return end
5122   texio.write('Replacing ' .. script .. ' script ranges')
5123   Babel.script_blocks[script] = {}
5124   for s, e in string.gmatch(chrng..' ', '(-.)%.(-.)%s') do
5125     table.insert(
5126       Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5127   end
5128 end
5129 function Babel.discard_sublr(str)
5130   if str:find( [[\string\indexentry]] ) and
5131     str:find( [[\string\babelsublr]] ) then
5132     str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5133       function(m) return m:sub(2,-2) end )
5134   end
5135   return str
5136 end
5137 }
5138 \endgroup
5139 \ifx\newattribute\@undefined\else
5140   \newattribute\bbl@attr@locale
5141   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5142   \AddBabelHook{luatex}{beforeextras}{%
5143     \setattribute\bbl@attr@locale\localeid}
5144 \fi
5145 \def\BabelStringsDefault{unicode}
5146 \let\luabbl@stop\relax
5147 \AddBabelHook{luatex}{encodedcommands}{%
5148   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5149   \ifx\bbl@tempa\bbl@tempb\else
5150     \directlua{Babel.begin_process_input()}%
5151     \def\luabbl@stop{%
5152       \directlua{Babel.end_process_input()}}%
5153   \fi}%
5154 \AddBabelHook{luatex}{stopcommands}{%
5155   \luabbl@stop
5156   \let\luabbl@stop\relax}

```



```

5157 \AddBabelHook{luatex}{patterns}{%
5158   \@ifundefined{bbl@hyphendata@the\language}%
5159     {\def\bbl@elt##1##2##3##4{%
5160       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5161       \def\bbl@tempb{##3}%
5162       \ifx\bbl@tempb\@empty\else % if not a synonymous
5163         \def\bbl@tempc{##3}{##4}}%
5164       \fi
5165       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5166     \fi}%
5167   \bbl@languages
5168   \@ifundefined{bbl@hyphendata@the\language}%
5169     {\bbl@info{No hyphenation patterns were set for\%
5170       language '#2'. Reported}}%
5171     {\expandafter\expandafter\expandafter\bbl@luapatterns
5172       \csname bbl@hyphendata@the\language\endcsname}}}%
5173   \@ifundefined{bbl@patterns@}{}%
5174   \begingroup
5175     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5176     \ifin@ \else
5177       \ifx\bbl@patterns@\@empty\else
5178         \directlua{ Babel.addpatterns(
5179           [[\bbl@patterns@]], \number\language) }%
5180       \fi
5181       \@ifundefined{bbl@patterns@#1}%
5182       \@empty
5183       {\directlua{ Babel.addpatterns(
5184         [[\space\csname bbl@patterns@#1\endcsname]],
5185         \number\language) }}%
5186       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5187     \fi
5188   \endgroup}%
5189   \bbl@exp{%
5190     \bbl@ifunset{bbl@prehc@\languagename}{}%
5191     {\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}}%
5192     {\prehyphenchar=\bbl@cl{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.

```

5193 \@onlypreamble\babelpatterns
5194 \AtEndOfPackage{%
5195   \newcommand\babelpatterns[2][\@empty]{%
5196     \ifx\bbl@patterns\relax
5197       \let\bbl@patterns@\@empty
5198     \fi
5199     \ifx\bbl@pttnlist\@empty\else
5200       \bbl@warning{%
5201         You must not intermingle \string\selectlanguage\space and\%
5202         \string\babelpatterns\space or some patterns will not\%
5203         be taken into account. Reported}%
5204       \fi
5205       \ifx\@empty#1%
5206         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5207       \else
5208         \edef\bbl@tempb{\zap@space#1 \@empty}%
5209         \bbl@for\bbl@tempa\bbl@tempb{%
5210           \bbl@fixname\bbl@tempa
5211           \bbl@iflanguage\bbl@tempa{%
5212             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5213               \@ifundefined{bbl@patterns@\bbl@tempa}%
5214               \@empty
5215               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%

```

```

5216         #2}}}%
5217     \fi}}

```

12.5 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.

```

5218 % TODO - to a lua file
5219 \directlua{
5220     Babel = Babel or {}
5221     Babel.linebreaking = Babel.linebreaking or {}
5222     Babel.linebreaking.before = {}
5223     Babel.linebreaking.after = {}
5224     Babel.locale = {} % Free to use, indexed by \localeid
5225     function Babel.linebreaking.add_before(func, pos)
5226         tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5227         if pos == nil then
5228             table.insert(Babel.linebreaking.before, func)
5229         else
5230             table.insert(Babel.linebreaking.before, pos, func)
5231         end
5232     end
5233     function Babel.linebreaking.add_after(func)
5234         tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5235         table.insert(Babel.linebreaking.after, func)
5236     end
5237 }
5238 \def\bbl@intraspace#1 #2 #3\@{%
5239     \directlua{
5240         Babel = Babel or {}
5241         Babel.intraspaces = Babel.intraspaces or {}
5242         Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5243             {b = #1, p = #2, m = #3}
5244         Babel.locale_props[\the\localeid].intraspace = %
5245             {b = #1, p = #2, m = #3}
5246     }}
5247 \def\bbl@intrapenalty#1\@{%
5248     \directlua{
5249         Babel = Babel or {}
5250         Babel.intrapenalties = Babel.intrapenalties or {}
5251         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5252         Babel.locale_props[\the\localeid].intrapenalty = #1
5253     }}
5254 \begingroup
5255 \catcode`\%=12
5256 \catcode`\^=14
5257 \catcode`\'=12
5258 \catcode`\~=12
5259 \gdef\bbl@seaintraspace{^
5260     \let\bbl@seaintraspace\relax
5261     \directlua{
5262         Babel = Babel or {}
5263         Babel.sea_enabled = true
5264         Babel.sea_ranges = Babel.sea_ranges or {}
5265         function Babel.set_chrngs (script, chrng)
5266             local c = 0
5267             for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5268                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5269                 c = c + 1
5270             end
5271         end

```

```

5272 function Babel.sea_disc_to_space (head)
5273   local sea_ranges = Babel.sea_ranges
5274   local last_char = nil
5275   local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5276   for item in node.traverse(head) do
5277     local i = item.id
5278     if i == node.id'glyph' then
5279       last_char = item
5280     elseif i == 7 and item.subtype == 3 and last_char
5281       and last_char.char > 0x0C99 then
5282       quad = font.getfont(last_char.font).size
5283       for lg, rg in pairs(sea_ranges) do
5284         if last_char.char > rg[1] and last_char.char < rg[2] then
5285           lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5286           local intraspace = Babel.intraspaces[lg]
5287           local intrapenalty = Babel.intrapealties[lg]
5288           local n
5289           if intrapenalty ~= 0 then
5290             n = node.new(14, 0)      ^% penalty
5291             n.penalty = intrapenalty
5292             node.insert_before(head, item, n)
5293           end
5294           n = node.new(12, 13)      ^% (glue, spaceskip)
5295           node.setglue(n, intraspace.b * quad,
5296             intraspace.p * quad,
5297             intraspace.m * quad)
5298           node.insert_before(head, item, n)
5299           node.remove(head, item)
5300         end
5301       end
5302     end
5303   end
5304 end
5305 }^^
5306 \bbl@luahyphenate}

```

12.6 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.

```

5307 \catcode`\%=14
5308 \gdef\bbl@cjkintraspaces{%
5309   \let\bbl@cjkintraspaces\relax
5310   \directlua{
5311     Babel = Babel or {}
5312     require('babel-data-cjk.lua')
5313     Babel.cjk_enabled = true
5314     function Babel.cjk_linebreak(head)
5315       local GLYPH = node.id'glyph'
5316       local last_char = nil
5317       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5318       local last_class = nil
5319       local last_lang = nil
5320
5321       for item in node.traverse(head) do
5322         if item.id == GLYPH then
5323
5324           local lang = item.lang
5325

```

```

5326     local LOCALE = node.get_attribute(item,
5327         Babel.attr_locale)
5328     local props = Babel.locale_props[LOCALE]
5329
5330     local class = Babel.cjk_class[item.char].c
5331
5332     if props.cjk_quotes and props.cjk_quotes[item.char] then
5333         class = props.cjk_quotes[item.char]
5334     end
5335
5336     if class == 'cp' then class = 'cl' end % ]] as CL
5337     if class == 'id' then class = 'I' end
5338
5339     local br = 0
5340     if class and last_class and Babel.cjk_breaks[last_class][class] then
5341         br = Babel.cjk_breaks[last_class][class]
5342     end
5343
5344     if br == 1 and props.linebreak == 'c' and
5345         lang ~= \the\l@nohyphenation\space and
5346         last_lang ~= \the\l@nohyphenation then
5347         local intrapenalty = props.intrapenalty
5348         if intrapenalty ~= 0 then
5349             local n = node.new(14, 0) % penalty
5350             n.penalty = intrapenalty
5351             node.insert_before(head, item, n)
5352         end
5353         local intraspace = props.intraspace
5354         local n = node.new(12, 13) % (glue, spaceskip)
5355         node.setglue(n, intraspace.b * quad,
5356             intraspace.p * quad,
5357             intraspace.m * quad)
5358         node.insert_before(head, item, n)
5359     end
5360
5361     if font.getfont(item.font) then
5362         quad = font.getfont(item.font).size
5363     end
5364     last_class = class
5365     last_lang = lang
5366     else % if penalty, glue or anything else
5367         last_class = nil
5368     end
5369 end
5370 lang.hyphenate(head)
5371 end
5372 }%
5373 \bbl@luahyphenate}
5374 \gdef\bbl@luahyphenate{%
5375 \let\bbl@luahyphenate\relax
5376 \directlua{
5377     luatexbase.add_to_callback('hyphenate',
5378     function (head, tail)
5379         if Babel.linebreaking.before then
5380             for k, func in ipairs(Babel.linebreaking.before) do
5381                 func(head)
5382             end
5383         end
5384         if Babel.cjk_enabled then
5385             Babel.cjk_linebreak(head)
5386         end
5387         lang.hyphenate(head)
5388         if Babel.linebreaking.after then

```

```

5389     for k, func in ipairs(Babel.linebreaking.after) do
5390         func(head)
5391     end
5392 end
5393 if Babel.sea_enabled then
5394     Babel.sea_disc_to_space(head)
5395 end
5396 end,
5397 'Babel.hyphenate')
5398 }
5399 }
5400 \endgroup
5401 \def\bbl@provide@intraspace{%
5402     \bbl@ifunset{\bbl@intsp@{language}}{%
5403         {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5404             \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5405         \ifin@           % cjk
5406             \bbl@cjk@intraspace
5407             \directlua{
5408                 Babel = Babel or {}
5409                 Babel.locale_props = Babel.locale_props or {}
5410                 Babel.locale_props[\the\localeid].linebreak = 'c'
5411             }%
5412             \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@@}%
5413             \ifx\bbl@KVP@intrapenalty\@nnil
5414                 \bbl@intrapenalty0@@
5415             \fi
5416         \else           % sea
5417             \bbl@sea@intraspace
5418             \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@@}%
5419             \directlua{
5420                 Babel = Babel or {}
5421                 Babel.sea_ranges = Babel.sea_ranges or {}
5422                 Babel.set_chranges('\bbl@cl{sbcpr}',
5423                                     '\bbl@cl{chrng}')
5424             }%
5425             \ifx\bbl@KVP@intrapenalty\@nnil
5426                 \bbl@intrapenalty0@@
5427             \fi
5428         \fi
5429     \fi
5430     \ifx\bbl@KVP@intrapenalty\@nnil\else
5431         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5432     \fi}}

```

12.7 Arabic justification

```

5433 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5434 \def\bblar@chars{%
5435     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5436     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5437     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5438 \def\bblar@elongated{%
5439     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5440     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5441     0649,064A}
5442 \begin{group}
5443     \catcode`\_ =11 \catcode`\:=11
5444     \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5445 \end{group}
5446 \gdef\bbl@arabicjust{%
5447     \let\bbl@arabicjust\relax
5448     \newattribute\bblar@kashida

```

```

5449 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5450 \bblar@kashida=\z@
5451 \bbl@patchfont{\bbl@parsejalt}}%
5452 \directlua{
5453   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5454   Babel.arabic.elong_map[\the\localeid] = {}
5455   luatexbase.add_to_callback('post_linebreak_filter',
5456     Babel.arabic.justify, 'Babel.arabic.justify')
5457   luatexbase.add_to_callback('hpack_filter',
5458     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5459 }%
5460 % Save both node lists to make replacement. TODO. Save also widths to
5461 % make computations
5462 \def\bblar@fetchjalt#1#2#3#4{%
5463   \bbl@exp{\bbl@foreach{#1}}{%
5464     \bbl@ifunset{bblar@JE@##1}%
5465       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5466       {\setbox\z@\hbox{^^^^200d\char"@nameuse{bblar@JE@##1}#2}}%
5467   \directlua{%
5468     local last = nil
5469     for item in node.traverse(tex.box[0].head) do
5470       if item.id == node.id'glyph' and item.char > 0x600 and
5471         not (item.char == 0x200D) then
5472         last = item
5473       end
5474     end
5475     Babel.arabic.#3['##1#4'] = last.char
5476   }}
5477 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5478 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5479 % positioning?
5480 \gdef\bbl@parsejalt{%
5481   \ifx\addfontfeature\undefined\else
5482     \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
5483     \ifin@
5484       \directlua{%
5485         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5486           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5487           tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5488         end
5489       }%
5490     \fi
5491   \fi}
5492 \gdef\bbl@parsejalti{%
5493   \begingroup
5494     \let\bbl@parsejalt\relax % To avoid infinite loop
5495     \edef\bbl@tempb{\fontid\font}%
5496     \bblar@nofswarn
5497     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5498     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5499     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5500     \addfontfeature{RawFeature+=jalt}%
5501     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5502     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5503     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5504     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5505     \directlua{%
5506       for k, v in pairs(Babel.arabic.from) do
5507         if Babel.arabic.dest[k] and
5508           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5509           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5510             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5511         end

```

```

5512         end
5513     }%
5514 \endgroup}
5515 %
5516 \begingroup
5517 \catcode`#=11
5518 \catcode`~=11
5519 \directlua{
5520
5521 Babel.arabic = Babel.arabic or {}
5522 Babel.arabic.from = {}
5523 Babel.arabic.dest = {}
5524 Babel.arabic.justify_factor = 0.95
5525 Babel.arabic.justify_enabled = true
5526
5527 function Babel.arabic.justify(head)
5528     if not Babel.arabic.justify_enabled then return head end
5529     for line in node.traverse_id(node.id'hlist', head) do
5530         Babel.arabic.justify_hlist(head, line)
5531     end
5532     return head
5533 end
5534
5535 function Babel.arabic.justify_hbox(head, gc, size, pack)
5536     local has_inf = false
5537     if Babel.arabic.justify_enabled and pack == 'exactly' then
5538         for n in node.traverse_id(12, head) do
5539             if n.stretch_order > 0 then has_inf = true end
5540         end
5541         if not has_inf then
5542             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5543         end
5544     end
5545     return head
5546 end
5547
5548 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5549     local d, new
5550     local k_list, k_item, pos_inline
5551     local width, width_new, full, k_curr, wt_pos, goal, shift
5552     local subst_done = false
5553     local elong_map = Babel.arabic.elong_map
5554     local last_line
5555     local GLYPH = node.id'glyph'
5556     local KASHIDA = Babel.attr_kashida
5557     local LOCALE = Babel.attr_locale
5558
5559     if line == nil then
5560         line = {}
5561         line.glue_sign = 1
5562         line.glue_order = 0
5563         line.head = head
5564         line.shift = 0
5565         line.width = size
5566     end
5567
5568     % Exclude last line. todo. But-- it discards one-word lines, too!
5569     % ? Look for glue = 12:15
5570     if (line.glue_sign == 1 and line.glue_order == 0) then
5571         elongs = {} % Stores elongated candidates of each line
5572         k_list = {} % And all letters with kashida
5573         pos_inline = 0 % Not yet used
5574

```

```

5575 for n in node.traverse_id(GLYPH, line.head) do
5576     pos_inline = pos_inline + 1 % To find where it is. Not used.
5577
5578     % Elongated glyphs
5579     if elong_map then
5580         local locale = node.get_attribute(n, LOCALE)
5581         if elong_map[locale] and elong_map[locale][n.font] and
5582             elong_map[locale][n.font][n.char] then
5583             table.insert(elongs, {node = n, locale = locale} )
5584             node.set_attribute(n.prev, KASHIDA, 0)
5585         end
5586     end
5587
5588     % Tatwil
5589     if Babel.kashida_wts then
5590         local k_wt = node.get_attribute(n, KASHIDA)
5591         if k_wt > 0 then % todo. parameter for multi inserts
5592             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5593         end
5594     end
5595
5596 end % of node.traverse_id
5597
5598 if #elongs == 0 and #k_list == 0 then goto next_line end
5599 full = line.width
5600 shift = line.shift
5601 goal = full * Babel.arabic.justify_factor % A bit crude
5602 width = node.dimensions(line.head) % The 'natural' width
5603
5604 % == Elongated ==
5605 % Original idea taken from 'chickenize'
5606 while (#elongs > 0 and width < goal) do
5607     subst_done = true
5608     local x = #elongs
5609     local curr = elongs[x].node
5610     local oldchar = curr.char
5611     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5612     width = node.dimensions(line.head) % Check if the line is too wide
5613     % Substitute back if the line would be too wide and break:
5614     if width > goal then
5615         curr.char = oldchar
5616         break
5617     end
5618     % If continue, pop the just substituted node from the list:
5619     table.remove(elongs, x)
5620 end
5621
5622 % == Tatwil ==
5623 if #k_list == 0 then goto next_line end
5624
5625 width = node.dimensions(line.head) % The 'natural' width
5626 k_curr = #k_list
5627 wt_pos = 1
5628
5629 while width < goal do
5630     subst_done = true
5631     k_item = k_list[k_curr].node
5632     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5633         d = node.copy(k_item)
5634         d.char = 0x0640
5635         line.head, new = node.insert_after(line.head, k_item, d)
5636         width_new = node.dimensions(line.head)
5637         if width > goal or width == width_new then

```



```

5638         node.remove(line.head, new) % Better compute before
5639         break
5640     end
5641     width = width_new
5642 end
5643 if k_curr == 1 then
5644     k_curr = #k_list
5645     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5646 else
5647     k_curr = k_curr - 1
5648 end
5649 end
5650
5651 ::next_line::
5652
5653 % Must take into account marks and ins, see luatex manual.
5654 % Have to be executed only if there are changes. Investigate
5655 % what's going on exactly.
5656 if subst_done and not gc then
5657     d = node.hpack(line.head, full, 'exactly')
5658     d.shift = shift
5659     node.insert_before(head, line, d)
5660     node.remove(head, line)
5661 end
5662 end % if process line
5663 end
5664 }
5665 \endgroup
5666 \fi\fi % Arabic just block

```

12.8 Common stuff

```

5667 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5668 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5669 \DisableBabelHook{babel-fontspec}
5670 <Font selection>

```

12.9 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.

```

5671 % TODO - to a lua file
5672 \directlua{
5673 Babel.script_blocks = {
5674   ['dflt'] = {},
5675   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5676             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5677   ['Armn'] = {{0x0530, 0x058F}},
5678   ['Beng'] = {{0x0980, 0x09FF}},
5679   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5680   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5681   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5682             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5683   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5684   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5685             {0xAB00, 0xAB2F}},
5686   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5687   % Don't follow strictly Unicode, which places some Coptic letters in
5688   % the 'Greek and Coptic' block

```

```

5689 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5690 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5691             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5692             {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5693             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5694             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5695             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5696 ['Hebr'] = {{0x0590, 0x05FF}},
5697 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5698             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5699 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5700 ['Knda'] = {{0x0C80, 0x0CFF}},
5701 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5702             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5703             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5704 ['Lao'] = {{0x0E80, 0x0EFF}},
5705 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5706             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5707             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5708 ['Mahj'] = {{0x11150, 0x1117F}},
5709 ['Mlym'] = {{0x0D00, 0x0D7F}},
5710 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5711 ['Orya'] = {{0x0B00, 0x0B7F}},
5712 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5713 ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5714 ['Taml'] = {{0x0B80, 0x0BFF}},
5715 ['Telu'] = {{0x0C00, 0x0C7F}},
5716 ['Tfng'] = {{0x2D30, 0x2D7F}},
5717 ['Thai'] = {{0x0E00, 0x0E7F}},
5718 ['Tibt'] = {{0x0F00, 0x0FFF}},
5719 ['Vaii'] = {{0xA500, 0xA63F}},
5720 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5721 }
5722
5723 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5724 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5725 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5726
5727 function Babel.locale_map(head)
5728   if not Babel.locale_mapped then return head end
5729
5730   local LOCALE = Babel.attr_locale
5731   local GLYPH = node.id('glyph')
5732   local inmath = false
5733   local toloc_save
5734   for item in node.traverse(head) do
5735     local toloc
5736     if not inmath and item.id == GLYPH then
5737       % Optimization: build a table with the chars found
5738       if Babel.chr_to_loc[item.char] then
5739         toloc = Babel.chr_to_loc[item.char]
5740       else
5741         for lc, maps in pairs(Babel.loc_to_scr) do
5742           for _, rg in pairs(maps) do
5743             if item.char >= rg[1] and item.char <= rg[2] then
5744               Babel.chr_to_loc[item.char] = lc
5745               toloc = lc
5746               break
5747             end
5748           end
5749         end
5750       end
5751       % Now, take action, but treat composite chars in a different

```

```

5752 % fashion, because they 'inherit' the previous locale. Not yet
5753 % optimized.
5754 if not toloc and
5755     (item.char >= 0x0300 and item.char <= 0x036F) or
5756     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5757     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5758     toloc = toloc_save
5759 end
5760 if toloc and Babel.locale_props[toloc] and
5761     Babel.locale_props[toloc].letters and
5762     tex.getcatcode(item.char) \string~= 11 then
5763     toloc = nil
5764 end
5765 if toloc and toloc > -1 then
5766     if Babel.locale_props[toloc].lg then
5767         item.lang = Babel.locale_props[toloc].lg
5768         node.set_attribute(item, LOCALE, toloc)
5769     end
5770     if Babel.locale_props[toloc]['/'..item.font] then
5771         item.font = Babel.locale_props[toloc]['/'..item.font]
5772     end
5773     toloc_save = toloc
5774 end
5775 elseif not inmath and item.id == 7 then % Apply recursively
5776     item.replace = item.replace and Babel.locale_map(item.replace)
5777     item.pre      = item.pre and Babel.locale_map(item.pre)
5778     item.post     = item.post and Babel.locale_map(item.post)
5779 elseif item.id == node.id'math' then
5780     inmath = (item.subtype == 0)
5781 end
5782 end
5783 return head
5784 end
5785 }

```

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

```

5786 \newcommand\babelcharproperty[1]{%
5787   \count@=#1\relax
5788   \ifvmode
5789     \expandafter\bbl@chprop
5790   \else
5791     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5792               vertical mode (preamble or between paragraphs)}%
5793     {See the manual for futher info}%
5794   \fi}
5795 \newcommand\bbl@chprop[3][\the\count@]{%
5796   \@tempcnta=#1\relax
5797   \bbl@ifunset{\bbl@chprop@#2}%
5798   {\bbl@error{No property named '#2'. Allowed values are\\%
5799             direction (bc), mirror (bmg), and linebreak (lb)}%
5800   {See the manual for futher info}}%
5801   {}%
5802   \loop
5803     \bbl@cs{chprop@#2}{#3}%
5804     \ifnum\count@<\@tempcnta
5805       \advance\count@\@ne
5806     \repeat}
5807 \def\bbl@chprop@direction#1{%
5808   \directlua{
5809     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5810     Babel.characters[\the\count@]['d'] = '#1'
5811   }}

```

```

5812 \let\bbl@chprop@bc\bbl@chprop@direction
5813 \def\bbl@chprop@mirror#1{%
5814   \directlua{
5815     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5816     Babel.characters[\the\count@]['m'] = '\number#1'
5817   }}
5818 \let\bbl@chprop@bmg\bbl@chprop@mirror
5819 \def\bbl@chprop@linebreak#1{%
5820   \directlua{
5821     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5822     Babel.cjk_characters[\the\count@]['c'] = '#1'
5823   }}
5824 \let\bbl@chprop@lb\bbl@chprop@linebreak
5825 \def\bbl@chprop@locale#1{%
5826   \directlua{
5827     Babel.chr_to_loc = Babel.chr_to_loc or {}
5828     Babel.chr_to_loc[\the\count@] =
5829       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5830   }}

```

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.

```

5831 \directlua{
5832   Babel.nohyphenation = \the\l@nohyphenation
5833 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the {*n*} syntax. For example, pre={1}{1}- becomes function(*m*) return *m*[1]..*m*[1]..'-' end, where *m* are the matches returned after applying the pattern. With a mapped capture the functions are similar to function(*m*) return Babel.capt_map(*m*[1],1) end, where the last argument identifies the mapping to be applied to *m*[1]. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

```

5834 \begingroup
5835 \catcode`\~ = 12
5836 \catcode`\% = 12
5837 \catcode`\& = 14
5838 \catcode`\| = 12
5839 \gdef\babelprehyphenation{&&
5840   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}[]]}
5841 \gdef\babelposthyphenation{&&
5842   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}[]]}
5843 \gdef\bbl@postlinebreak{\bbl@settransform{2}}[] && WIP
5844 \gdef\bbl@settransform#1[#2]#3#4#5{&&
5845   \ifcase#1
5846     \bbl@activateprehyphen
5847   \or
5848     \bbl@activateposthyphen
5849   \fi
5850 \begingroup
5851   \def\babeltempa{\bbl@add@list\babeltempb}&&
5852   \let\babeltempb\@empty
5853   \def\bbl@tempa{#5}&&
5854   \bbl@replace\bbl@tempa{,}{,}&& TODO. Ugly trick to preserve {}
5855   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&&
5856     \bbl@ifsamestring{##1}{remove}&&
5857     {\bbl@add@list\babeltempb{nil}}&&
5858     {\directlua{
5859       local rep = {[##1]=}
5860       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5861       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5862       rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)

```

```

5863         if #1 == 0 or #1 == 2 then
5864             rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5865                 'space = {' .. '%2, %3, %4' .. '}')
5866             rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5867                 'spacefactor = {' .. '%2, %3, %4' .. '}')
5868             rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5869         else
5870             rep = rep:gsub('(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5871             rep = rep:gsub('(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5872             rep = rep:gsub('(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5873         end
5874         tex.print([[\\string\\babeltempa{}}] .. rep .. [[]]])
5875     }}&%
5876 \\bbl@foreach\\babeltempb{&%
5877     \\bbl@forkv{##1}{&%
5878         \\in{####1},{,nil,step,data,remove,insert,string,no,pre,&%
5879             no,post,penalty,kashida,space,spacefactor,}&%
5880         \\ifin@\\else
5881             \\bbl@error
5882             {Bad option '####1' in a transform.\\&%
5883                 I'll ignore it but expect more errors}&%
5884             {See the manual for further info.}&%
5885             \\fi}&%
5886     \\let\\bbl@kv@attribute\\relax
5887     \\let\\bbl@kv@label\\relax
5888     \\let\\bbl@kv@font@empty
5889     \\bbl@forkv{#2}{\\bbl@csarg\\edef{kv##1}{##2}}&%
5890     \\ifx\\bbl@kv@font@empty\\else\\bbl@settransfont\\fi
5891     \\ifx\\bbl@kv@attribute\\relax
5892         \\ifx\\bbl@kv@label\\relax\\else
5893             \\bbl@exp{\\bbl@trim@def\\bbl@kv@font@empty{\\bbl@kv@font@empty}}&%
5894             \\bbl@replace\\bbl@kv@font@empty{,}&%
5895             \\edef\\bbl@kv@attribute{\\bbl@ATR@\\bbl@kv@label @#3@\\bbl@kv@font@empty}&%
5896             \\count@\\z@
5897             \\def\\bbl@elt##1##2##3{&%
5898                 \\bbl@ifsamestring{#3,\\bbl@kv@label}{##1,##2}&%
5899                 {\\bbl@ifsamestring{\\bbl@kv@font@empty}{##3}&%
5900                     {\\count@\\@ne}&%
5901                     {\\bbl@error
5902                         {Transforms cannot be re-assigned to different\\&%
5903                             fonts. The conflict is in '\\bbl@kv@label'.\\&%
5904                             Apply the same fonts or use a different label}&%
5905                         {See the manual for further details.}}}&%
5906                 }}&%
5907             \\bbl@transfont@list
5908             \\ifnum\\count@=\\z@
5909                 \\bbl@exp{\\global\\bbl@add\\bbl@transfont@list
5910                     {\\bbl@elt{#3}{\\bbl@kv@label}{\\bbl@kv@font@empty}}}&%
5911             \\fi
5912             \\bbl@ifunset{\\bbl@kv@attribute}&%
5913             {\\global\\bbl@carg\\newattribute{\\bbl@kv@attribute}}&%
5914             {}&%
5915             \\global\\bbl@carg\\setattribute{\\bbl@kv@attribute}\\@ne
5916         \\fi
5917     \\else
5918         \\edef\\bbl@kv@attribute{\\expandafter\\bbl@stripslash\\bbl@kv@attribute}&%
5919     \\fi
5920     \\directlua{
5921         local lbkr = Babel.linebreaking.replacements[#1]
5922         local u = unicode.utf8
5923         local id, attr, label
5924         if #1 == 0 or #1 == 2 then
5925             id = \\the\\csname bbl@id@#3\\endcsname\\space

```

```

5926     else
5927         id = \the\csname l@#3\endcsname\space
5928     end
5929     \ifx\bbl@kv@attribute\relax
5930         attr = -1
5931     \else
5932         attr = luatexbase.registernumber'\bbl@kv@attribute'
5933     \fi
5934     \ifx\bbl@kv@label\relax\else    %% Same refs:
5935         label = [==[\bbl@kv@label]==]
5936     \fi
5937     %% Convert pattern:
5938     local patt = string.gsub([==[#4]==], '%s', '')
5939     if #1 == 0 or #1 == 2 then
5940         patt = string.gsub(patt, '|', ' ')
5941     end
5942     if not u.find(patt, '()', nil, true) then
5943         patt = '()' .. patt .. '()'
5944     end
5945     if #1 == 1 then
5946         patt = string.gsub(patt, '%(%)%', '^()')
5947         patt = string.gsub(patt, '%$(%)', '()$')
5948     end
5949     patt = u.gsub(patt, '{{(.)}}',
5950         function (n)
5951             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5952         end)
5953     patt = u.gsub(patt, '{{(%x%x%x%x+)}}',
5954         function (n)
5955             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5956         end)
5957     lbkr[id] = lbkr[id] or {}
5958     table.insert(lbkr[id],
5959         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5960 }&%
5961 \endgroup}
5962 \endgroup
5963 \let\bbl@transfont@list\@empty
5964 \def\bbl@settransfont{%
5965     \global\let\bbl@settransfont\relax % Execute only once
5966     \gdef\bbl@transfont{%
5967         \def\bbl@elt####1####2####3{%
5968             \bbl@ifblank{####3}%
5969             {\count@tw@}% Do nothing if no fonts
5970             {\count@z@
5971             \bbl@vforeach{####3}{%
5972                 \def\bbl@tempd{#####1}%
5973                 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
5974                 \ifx\bbl@tempd\bbl@tempe
5975                     \count@ne
5976                 \else\ifx\bbl@tempd\bbl@transfam
5977                     \count@ne
5978                 \fi\fi}%
5979             \ifcase\count@
5980                 \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
5981             \or
5982                 \bbl@csarg\setattribute{ATR####2@####1@####3}\@ne
5983             \fi}}%
5984             \bbl@transfont@list}%
5985     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
5986     \gdef\bbl@transfam{-unknown-}%
5987     \bbl@foreach\bbl@font@fams{%
5988         \AddToHook{##1family}{\def\bbl@transfam{##1}}%

```

```

5989 \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
5990 {\xdef\bbl@transfam{##1}}}%
5991 {}}}
5992 \DeclareRobustCommand\enablelocaletransform[1]{%
5993 \bbl@ifunset{\bbl@ATR@#1@language @}{%
5994 {\bbl@error
5995 {'#1' for '\language' cannot be enabled.\%
5996 Maybe there is a typo or it's a font-dependent transform}%
5997 {See the manual for further details.}}%
5998 {\bbl@csarg\setattribute{ATR@#1@language @}{\@ne}}
5999 \DeclareRobustCommand\disablelocaletransform[1]{%
6000 \bbl@ifunset{\bbl@ATR@#1@language @}{%
6001 {\bbl@error
6002 {'#1' for '\language' cannot be disabled.\%
6003 Maybe there is a typo or it's a font-dependent transform}%
6004 {See the manual for further details.}}%
6005 {\bbl@csarg\unsetattribute{ATR@#1@language @}}}%
6006 \def\bbl@activateposthyphen{%
6007 \let\bbl@activateposthyphen\relax
6008 \directlua{
6009 require('babel-transforms.lua')
6010 Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6011 }}
6012 \def\bbl@activateprehyphen{%
6013 \let\bbl@activateprehyphen\relax
6014 \directlua{
6015 require('babel-transforms.lua')
6016 Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6017 }}

```

12.10 Bidi

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

```

6018 \def\bbl@activate@preotf{%
6019 \let\bbl@activate@preotf\relax % only once
6020 \directlua{
6021 Babel = Babel or {}
6022 %
6023 function Babel.pre_otfload_v(head)
6024 if Babel.numbers and Babel.digits_mapped then
6025 head = Babel.numbers(head)
6026 end
6027 if Babel.bidi_enabled then
6028 head = Babel.bidi(head, false, dir)
6029 end
6030 return head
6031 end
6032 %
6033 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6034 if Babel.numbers and Babel.digits_mapped then
6035 head = Babel.numbers(head)
6036 end
6037 if Babel.bidi_enabled then
6038 head = Babel.bidi(head, false, dir)
6039 end
6040 return head
6041 end
6042 %
6043 luatexbase.add_to_callback('pre_linebreak_filter',
6044 Babel.pre_otfload_v,
6045 'Babel.pre_otfload_v',

```

```

6046     luatexbase.priority_in_callback('pre_linebreak_filter',
6047     'luaotfload.node_processor') or nil)
6048 %
6049     luatexbase.add_to_callback('hpack_filter',
6050     Babel.pre_otfload_h,
6051     'Babel.pre_otfload_h',
6052     luatexbase.priority_in_callback('hpack_filter',
6053     'luaotfload.node_processor') or nil)
6054 }}

```

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=`.

```

6055 \ifnum\bbl@bidimode>\@ne % Excludes default=1
6056   \let\bbl@beforeforeign\leavevmode
6057   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6058   \RequirePackage{luatexbase}
6059   \bbl@activate@preotf
6060   \directlua{
6061     require('babel-data-bidi.lua')
6062     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6063       require('babel-bidi-basic.lua')
6064     \or
6065       require('babel-bidi-basic-r.lua')
6066     \fi}
6067   \newattribute\bbl@attr@dir
6068   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6069   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6070 \fi
6071 \chardef\bbl@thetextdir\z@
6072 \chardef\bbl@thepardir\z@
6073 \def\bbl@getluadir#1{%
6074   \directlua{
6075     if tex.#1dir == 'TLT' then
6076       tex.sprint('0')
6077     elseif tex.#1dir == 'TRT' then
6078       tex.sprint('1')
6079     end}}
6080 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6081   \ifcase#3\relax
6082     \ifcase\bbl@getluadir{#1}\relax\else
6083       #2 TLT\relax
6084     \fi
6085   \else
6086     \ifcase\bbl@getluadir{#1}\relax
6087       #2 TRT\relax
6088     \fi
6089   \fi}
6090 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6091 \def\bbl@thedir{0}
6092 \def\bbl@textdir#1{%
6093   \bbl@setluadir{text}\textdir{#1}%
6094   \chardef\bbl@thetextdir#1\relax
6095   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6096   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6097 \def\bbl@pardir#1{% Used twice
6098   \bbl@setluadir{par}\pardir{#1}%
6099   \chardef\bbl@thepardir#1\relax}
6100 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6101 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6102 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

RTL text inside math needs special attention. It affects not only to actual math stuff, but also to ‘tabular’, which is based on a fake math.


```

6103 \ifnum\bbl@bidimode>\z@
6104 \def\bbl@insidemath{0}%
6105 \def\bbl@everymath{\def\bbl@insidemath{1}}
6106 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6107 \frozen@everymath\expandafter{%
6108   \expandafter\bbl@everymath\the\frozen@everymath}
6109 \frozen@everydisplay\expandafter{%
6110   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6111 \AtBeginDocument{
6112   \directlua{
6113     function Babel.math_box_dir(head)
6114       if not (token.get_macro('bbl@insidemath') == '0') then
6115         if Babel.hlist_has_bidi(head) then
6116           local d = node.new(node.id'dir')
6117           d.dir = '+TRT'
6118           node.insert_before(head, node.has_glyph(head), d)
6119           for item in node.traverse(head) do
6120             node.set_attribute(item,
6121               Babel.attr_dir, token.get_macro('bbl@thedir'))
6122           end
6123         end
6124       end
6125       return head
6126     end
6127     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6128       "Babel.math_box_dir", 0)
6129   }%
6130 \fi

```

12.11 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

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

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

6131 \bbl@trace{Redefinitions for bidi layout}
6132 %
6133 <<(*More package options)>> ≡
6134 \chardef\bbl@eqnpos\z@
6135 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6136 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6137 <</More package options>>
6138 %
6139 \ifnum\bbl@bidimode>\z@
6140   \ifx\matheqdirmode\undefined\else
6141     \matheqdirmode\@ne % A luatex primitive
6142   \fi
6143   \let\bbl@eqnodir\relax
6144   \def\bbl@eqdel{()}
6145   \def\bbl@eqnum{%
6146     {\normalfont\normalcolor
6147       \expandafter\@firstoftwo\bbl@eqdel
6148       \theequation
6149       \expandafter\@secondoftwo\bbl@eqdel}}
6150   \def\bbl@puteqno#1{\eqno\hbox{#1}}

```

```

6151 \def\bb@putleqno#1{\leqno\hbox{#1}}
6152 \def\bb@eqno@flip#1{%
6153   \ifdim\predisplaysize=-\maxdimen
6154     \eqno
6155     \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6156   \else
6157     \leqno\hbox{#1}%
6158   \fi}
6159 \def\bb@leqno@flip#1{%
6160   \ifdim\predisplaysize=-\maxdimen
6161     \leqno
6162     \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6163   \else
6164     \eqno\hbox{#1}%
6165   \fi}
6166 \AtBeginDocument{%
6167   \ifx\bb@noamsmath\relax\else
6168     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6169       \AddToHook{env/equation/begin}{%
6170         \ifnum\bb@thetextdir>\z@
6171           \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6172           \let\@eqnnum\bb@eqnum
6173           \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6174           \chardef\bb@thetextdir\z@
6175           \bb@add\normalfont{\bb@eqnodir}%
6176           \ifcase\bb@eqnpos
6177             \let\bb@puteqno\bb@eqno@flip
6178           \or
6179             \let\bb@puteqno\bb@leqno@flip
6180           \fi
6181         \fi}%
6182       \ifnum\bb@eqnpos=\tw@\else
6183         \def\endequation{\bb@puteqno{\@eqnnum}$$\@ignoretrue}%
6184       \fi
6185       \AddToHook{env/eqnarray/begin}{%
6186         \ifnum\bb@thetextdir>\z@
6187           \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6188           \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6189           \chardef\bb@thetextdir\z@
6190           \bb@add\normalfont{\bb@eqnodir}%
6191           \ifnum\bb@eqnpos=\@ne
6192             \def\@eqnnum{%
6193               \setbox\z@\hbox{\bb@eqnum}%
6194               \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6195             \else
6196               \let\@eqnnum\bb@eqnum
6197             \fi
6198           \fi}
6199       % Hack. YA luatex bug?:
6200       \expandafter\bb@sreplace\csname] \endcsname{$$$}\eqno\kern.001pt$$$}%
6201     \else % amstex
6202       \bb@exp{% Hack to hide maybe undefined conditionals:
6203         \chardef\bb@eqnpos=0%
6204         \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6205       \ifnum\bb@eqnpos=\@ne
6206         \let\bb@ams@lap\hbox
6207       \else
6208         \let\bb@ams@lap\llap
6209       \fi
6210       \ExplSyntaxOn
6211       \bb@sreplace\intertext@{\normalbaselines}%
6212       {\normalbaselines
6213         \ifx\bb@eqnodir\relax\else\bb@pardir\@ne\bb@eqnodir\fi}%

```

```

6214 \ExplSyntaxOff
6215 \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6216 \ifx\bbl@ams@lap\hbox % leqno
6217   \def\bbl@ams@flip#1{%
6218     \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6219 \else % eqno
6220   \def\bbl@ams@flip#1{%
6221     \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6222 \fi
6223 \def\bbl@ams@preset#1{%
6224   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6225   \ifnum\bbl@thetextdir>\z@
6226     \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6227     \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6228     \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6229   \fi}%
6230 \ifnum\bbl@eqnpos=\tw@%
6231   \def\bbl@ams@equation{%
6232     \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6233     \ifnum\bbl@thetextdir>\z@
6234       \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6235       \chardef\bbl@thetextdir\z@
6236       \bbl@add\normalfont{\bbl@eqnodir}%
6237       \ifcase\bbl@eqnpos
6238         \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6239       \or
6240         \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6241       \fi
6242     \fi}%
6243   \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6244   \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6245 \fi
6246 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6247 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6248 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6249 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6250 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6251 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6252 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6253 % Hackish, for proper alignment. Don't ask me why it works!:
6254 \bbl@exp{% Avoid a 'visible' conditional
6255   \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6256 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6257 \AddToHook{env/split/before}{%
6258   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6259   \ifnum\bbl@thetextdir>\z@
6260     \bbl@ifsamestring\@currenvir{equation}%
6261     {\ifx\bbl@ams@lap\hbox % leqno
6262       \def\bbl@ams@flip#1{%
6263         \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}%
6264       \else
6265         \def\bbl@ams@flip#1{%
6266           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6267         \fi}%
6268       {}%
6269     \fi}%
6270 \fi\fi}
6271 \fi
6272 \def\bbl@provide@extra#1{%
6273   % == Counters: mapdigits ==
6274   % Native digits
6275   \ifx\bbl@KVP@mapdigits\@nnil\else
6276     \bbl@ifunset{\bbl@dgnat\@languagename}{}%

```

```

6277     {\RequirePackage{luatexbase}%
6278      \bbl@activate@preotf
6279      \directlua{
6280        Babel = Babel or {}  %% -> presets in luababel
6281        Babel.digits_mapped = true
6282        Babel.digits = Babel.digits or {}
6283        Babel.digits[\the\localeid] =
6284          table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6285        if not Babel.numbers then
6286          function Babel.numbers(head)
6287            local LOCALE = Babel.attr_locale
6288            local GLYPH = node.id'glyph'
6289            local inmath = false
6290            for item in node.traverse(head) do
6291              if not inmath and item.id == GLYPH then
6292                local temp = node.get_attribute(item, LOCALE)
6293                if Babel.digits[temp] then
6294                  local chr = item.char
6295                  if chr > 47 and chr < 58 then
6296                    item.char = Babel.digits[temp][chr-47]
6297                  end
6298                end
6299                elseif item.id == node.id'math' then
6300                  inmath = (item.subtype == 0)
6301                end
6302              end
6303            return head
6304          end
6305        end
6306      } }%
6307 \fi
6308 % == transforms ==
6309 \ifx\bbl@KVP@transforms\@nnil\else
6310   \def\bbl@elt##1##2##3{%
6311     \in@{$transforms.}{$##1}%
6312     \ifin@
6313       \def\bbl@tempa{##1}%
6314       \bbl@replace\bbl@tempa{transforms.}{}%
6315       \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6316     \fi}%
6317   \csname bbl@inidata@\language\endcsname
6318   \bbl@release@transforms\relax % \relax closes the last item.
6319 \fi}
6320 % Start tabular here:
6321 \def\localerestoredirs{%
6322   \ifcase\bbl@thetextdir
6323     \ifnum\textdirection=\z@\else\textdir TLT\fi
6324   \else
6325     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6326   \fi
6327   \ifcase\bbl@thepardir
6328     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6329   \else
6330     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6331   \fi}
6332 \IfBabelLayout{tabular}%
6333   {\chardef\bbl@tabular@mode\tw@}% All RTL
6334   {\IfBabelLayout{notabular}%
6335     {\chardef\bbl@tabular@mode\z@}%
6336     {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6337 \ifnum\bbl@bidimode>\@ne
6338   \ifnum\bbl@tabular@mode=\@ne
6339     \let\bbl@parabefore\relax

```

```

6340 \AddToHook{para/before}{\bbl@parabefore}
6341 \AtBeginDocument{%
6342   \bbl@replace\@tabular{$}{\%
6343     \def\bbl@insidemath{0}%
6344     \def\bbl@parabefore{\localerestoredirs}}%
6345   \ifnum\bbl@tabular@mode=\@ne
6346     \bbl@ifunset{\@tabclassz}{\%
6347       \bbl@exp{\% Hide conditionals
6348         \\\bbl@sreplace\\ \@tabclassz
6349           {\<ifcase>\\ \@chnum}%
6350           {\\\localerestoredirs\<ifcase>\\ \@chnum}}}%
6351   \ifpackageloaded{colortbl}%
6352     {\bbl@sreplace\@classz
6353       {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6354   {\ifpackageloaded{array}%
6355     {\bbl@exp{\% Hide conditionals
6356       \\\bbl@sreplace\\ \@classz
6357         {\<ifcase>\\ \@chnum}%
6358         {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6359         \\\bbl@sreplace\\ \@classz
6360         {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6361     {}}%
6362   \fi}
6363 \fi
6364 \AtBeginDocument{%
6365   \ifpackageloaded{multicol}%
6366     {\toks@\expandafter{\multi@column@out}%
6367      \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6368     {}}
6369 \fi
6370 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

OMEGA provided a companion to `\mathdir` (`\nextfakemath`) for those cases where we did not want it to be applied, so that the writing direction of the main text was left unchanged. `\bbl@nextfake` is an attempt to emulate it, because `luatex` has removed it without an alternative. Also, `\hangindent` does not honour direction changes by default, so we need to redefine `\@hangfrom`.

```

6371 \ifnum\bbl@bidimode>\z@
6372   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6373     \bbl@exp{%
6374       \def\\ \bbl@insidemath{0}%
6375       \mathdir\the\bodydir
6376       #1% Once entered in math, set boxes to restore values
6377       \<ifmmode>%
6378         \everyvbox{%
6379           \the\everyvbox
6380           \bodydir\the\bodydir
6381           \mathdir\the\mathdir
6382           \everyhbox{\the\everyhbox}%
6383           \everyvbox{\the\everyvbox}}%
6384         \everyhbox{%
6385           \the\everyhbox
6386           \bodydir\the\bodydir
6387           \mathdir\the\mathdir
6388           \everyhbox{\the\everyhbox}%
6389           \everyvbox{\the\everyvbox}}%
6390       \<fi>}}%
6391   \def\@hangfrom#1{%
6392     \setbox\@tempboxa\hbox{#1}%
6393     \hangindent\wd\@tempboxa
6394     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6395       \shapemode\@ne
6396     \fi
6397     \noindent\box\@tempboxa}

```

```

6398 \fi
6399 \IfBabelLayout{tabular}
6400 {\let\bbl@OL@tabular\@tabular
6401 \bbl@replace\@tabular{$}\bbl@nextfake$}%
6402 \let\bbl@NL@tabular\@tabular
6403 \AtBeginDocument{%
6404 \ifx\bbl@NL@tabular\@tabular\else
6405 \bbl@replace\@tabular{$}\bbl@nextfake$}%
6406 \let\bbl@NL@tabular\@tabular
6407 \fi}}
6408 {}
6409 \IfBabelLayout{lists}
6410 {\let\bbl@OL@list\list
6411 \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6412 \let\bbl@NL@list\list
6413 \def\bbl@listparshape#1#2#3{%
6414 \parshape #1 #2 #3 %
6415 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6416 \shapemode\tw@
6417 \fi}}
6418 {}
6419 \IfBabelLayout{graphics}
6420 {\let\bbl@pictresetdir\relax
6421 \def\bbl@pictsetdir#1{%
6422 \ifcase\bbl@thetextdir
6423 \let\bbl@pictresetdir\relax
6424 \else
6425 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6426 \or\textdir TLT
6427 \else\bodydir TLT \textdir TLT
6428 \fi
6429 % \text\par\dir required in pgf:
6430 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6431 \fi}%
6432 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6433 \directlua{
6434 Babel.get_picture_dir = true
6435 Babel.picture_has_bidi = 0
6436 %
6437 function Babel.picture_dir (head)
6438 if not Babel.get_picture_dir then return head end
6439 if Babel.hlist_has_bidi(head) then
6440 Babel.picture_has_bidi = 1
6441 end
6442 return head
6443 end
6444 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6445 "Babel.picture_dir")
6446 }%
6447 \AtBeginDocument{%
6448 \def\LS@rot{%
6449 \setbox\@outputbox\vbox{%
6450 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6451 \long\def\put(#1,#2)#3{%
6452 \@killglue
6453 % Try:
6454 \ifx\bbl@pictresetdir\relax
6455 \def\bbl@tempc{0}%
6456 \else
6457 \directlua{
6458 Babel.get_picture_dir = true
6459 Babel.picture_has_bidi = 0
6460 }%

```

```

6461 \setbox\z@\hb@xt@\z@{%
6462 \@defaultunitsset\@tempdimc{#1}\unitlength
6463 \kern\@tempdimc
6464 #3\hss}% TODO: #3 executed twice (below). That's bad.
6465 \edef\bb1@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6466 \fi
6467 % Do:
6468 \@defaultunitsset\@tempdimc{#2}\unitlength
6469 \raise\@tempdimc\hb@xt@\z@{%
6470 \@defaultunitsset\@tempdimc{#1}\unitlength
6471 \kern\@tempdimc
6472 {\ifnum\bb1@tempc>\z@\bb1@pictresetdir\fi#3}\hss}%
6473 \ignorespaces}%
6474 \MakeRobust\put}%
6475 \AtBeginDocument
6476 {\AddToHook{cmd/diagbox@pict/before}{\let\bb1@pictsetdir\@gobble}%
6477 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6478 \AddToHook{env/pgfpicture/begin}{\bb1@pictsetdir\@ne}%
6479 \bb1@add\pgfinterruptpicture{\bb1@pictresetdir}%
6480 \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir\z@}%
6481 \fi
6482 \ifx\tikzpicture\undefined\else
6483 \AddToHook{env/tikzpicture/begin}{\bb1@pictsetdir\tw@}%
6484 \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
6485 \bb1@sreplace\tikz{\begingroup}{\begingroup\bb1@pictsetdir\tw@}%
6486 \fi
6487 \ifx\tcolorbox\undefined\else
6488 \def\tcb@drawing@env@begin{%
6489 \csname tcb@before@\tcb@split@state\endcsname
6490 \bb1@pictsetdir\tw@
6491 \begin{\kvtcb@graphenv}%
6492 \tcb@bbdraw%
6493 \tcb@apply@graph@patches
6494 }%
6495 \def\tcb@drawing@env@end{%
6496 \end{\kvtcb@graphenv}%
6497 \bb1@pictresetdir
6498 \csname tcb@after@\tcb@split@state\endcsname
6499 }%
6500 \fi
6501 }}
6502 {}

```

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.

```

6503 \IfBabelLayout{counters*}%
6504 {\bb1@add\bb1@opt@layout{.counters.}%
6505 \directlua{
6506 \lua{
6507 \lua{
6508 }
6509 }
6509 \IfBabelLayout{counters}%
6510 {\let\bb1@OL@@@textsuperscript\@textsuperscript
6511 \bb1@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6512 \let\bb1@latinarabic=\@arabic
6513 \let\bb1@OL@@@arabic\@arabic
6514 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
6515 \ifpackagewith{babel}{bidi=default}%
6516 {\let\bb1@asciroman=\@roman
6517 \let\bb1@OL@@@roman\@roman
6518 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
6519 \let\bb1@asciiRoman=\@Roman

```

```

6520 \let\bbl@OL@@roman\@Roman
6521 \def\@Roman#1{\babelsublr{\ensureascii\bbl@asciiRoman#1}}}%
6522 \let\bbl@OL@labelenumii\labelenumii
6523 \def\labelenumii{}\theenumii{%
6524 \let\bbl@OL@p@enumiii\p@enumiii
6525 \def\p@enumiii{\p@enumii}\theenumii{}}{}{}{}
6526 <<Footnote changes>>
6527 \IfBabelLayout{footnotes}%
6528 {\let\bbl@OL@footnote\footnote
6529 \BabelFootnote\footnote\language{}}{}%
6530 \BabelFootnote\localfootnote\language{}}{}%
6531 \BabelFootnote\mainfootnote{}}{}{}
6532 {}

```

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.

```

6533 \IfBabelLayout{extras}%
6534 {\let\bbl@OL@underline\underline
6535 \bbl@sreplace\underline{$\@underline}{\bbl@nextfake$\@underline}%
6536 \let\bbl@OL@LaTeX2e\LaTeX2e
6537 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6538 \if b\expandafter\car\series\@nil\boldmath\fi
6539 \babelsublr{%
6540 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
6541 {}
6542 </luatex>

```

12.12 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.

```

6543 <*transforms>
6544 Babel.linebreaking.replacements = {}
6545 Babel.linebreaking.replacements[0] = {} -- pre
6546 Babel.linebreaking.replacements[1] = {} -- post
6547 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6548
6549 -- Discretionaries contain strings as nodes
6550 function Babel.str_to_nodes(fn, matches, base)
6551 local n, head, last
6552 if fn == nil then return nil end
6553 for s in string.utfvalues(fn(matches)) do
6554 if base.id == 7 then
6555 base = base.replace
6556 end
6557 n = node.copy(base)
6558 n.char = s
6559 if not head then
6560 head = n
6561 else
6562 last.next = n
6563 end
6564 last = n
6565 end

```



```

6566 return head
6567 end
6568
6569 Babel.fetch_subtext = {}
6570
6571 Babel.ignore_pre_char = function(node)
6572   return (node.lang == Babel.nohyphenation)
6573 end
6574
6575 -- Merging both functions doesn't seem feasible, because there are too
6576 -- many differences.
6577 Babel.fetch_subtext[0] = function(head)
6578   local word_string = ''
6579   local word_nodes = {}
6580   local lang
6581   local item = head
6582   local inmath = false
6583
6584   while item do
6585
6586     if item.id == 11 then
6587       inmath = (item.subtype == 0)
6588     end
6589
6590     if inmath then
6591       -- pass
6592
6593     elseif item.id == 29 then
6594       local locale = node.get_attribute(item, Babel.attr_locale)
6595
6596       if lang == locale or lang == nil then
6597         lang = lang or locale
6598         if Babel.ignore_pre_char(item) then
6599           word_string = word_string .. Babel.us_char
6600         else
6601           word_string = word_string .. unicode.utf8.char(item.char)
6602         end
6603         word_nodes[#word_nodes+1] = item
6604       else
6605         break
6606       end
6607
6608     elseif item.id == 12 and item.subtype == 13 then
6609       word_string = word_string .. ' '
6610       word_nodes[#word_nodes+1] = item
6611
6612     -- Ignore leading unrecognized nodes, too.
6613     elseif word_string ~= '' then
6614       word_string = word_string .. Babel.us_char
6615       word_nodes[#word_nodes+1] = item -- Will be ignored
6616     end
6617
6618     item = item.next
6619   end
6620
6621   -- Here and above we remove some trailing chars but not the
6622   -- corresponding nodes. But they aren't accessed.
6623   if word_string:sub(-1) == ' ' then
6624     word_string = word_string:sub(1,-2)
6625   end
6626   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6627   return word_string, word_nodes, item, lang
6628 end

```

```

6629
6630 Babel.fetch_subtext[1] = function(head)
6631   local word_string = ''
6632   local word_nodes = {}
6633   local lang
6634   local item = head
6635   local inmath = false
6636
6637   while item do
6638
6639     if item.id == 11 then
6640       inmath = (item.subtype == 0)
6641     end
6642
6643     if inmath then
6644       -- pass
6645
6646     elseif item.id == 29 then
6647       if item.lang == lang or lang == nil then
6648         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6649           lang = lang or item.lang
6650           word_string = word_string .. unicode.utf8.char(item.char)
6651           word_nodes[#word_nodes+1] = item
6652         end
6653       else
6654         break
6655       end
6656
6657     elseif item.id == 7 and item.subtype == 2 then
6658       word_string = word_string .. '='
6659       word_nodes[#word_nodes+1] = item
6660
6661     elseif item.id == 7 and item.subtype == 3 then
6662       word_string = word_string .. '|'
6663       word_nodes[#word_nodes+1] = item
6664
6665     -- (1) Go to next word if nothing was found, and (2) implicitly
6666     -- remove leading USs.
6667     elseif word_string == '' then
6668       -- pass
6669
6670     -- This is the responsible for splitting by words.
6671     elseif (item.id == 12 and item.subtype == 13) then
6672       break
6673
6674     else
6675       word_string = word_string .. Babel.us_char
6676       word_nodes[#word_nodes+1] = item -- Will be ignored
6677     end
6678
6679     item = item.next
6680   end
6681
6682   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6683   return word_string, word_nodes, item, lang
6684 end
6685
6686 function Babel.pre_hyphenate_replace(head)
6687   Babel.hyphenate_replace(head, 0)
6688 end
6689
6690 function Babel.post_hyphenate_replace(head)
6691   Babel.hyphenate_replace(head, 1)

```

```

6692 end
6693
6694 Babel.us_char = string.char(31)
6695
6696 function Babel.hyphenate_replace(head, mode)
6697     local u = unicode.utf8
6698     local lbkr = Babel.linebreaking.replacements[mode]
6699     if mode == 2 then mode = 0 end -- WIP
6700
6701     local word_head = head
6702
6703     while true do -- for each subtext block
6704
6705         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6706
6707         if Babel.debug then
6708             print()
6709             print((mode == 0) and '@@@<' or '@@@>', w)
6710         end
6711
6712         if nw == nil and w == '' then break end
6713
6714         if not lang then goto next end
6715         if not lbkr[lang] then goto next end
6716
6717         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6718         -- loops are nested.
6719         for k=1, #lbkr[lang] do
6720             local p = lbkr[lang][k].pattern
6721             local r = lbkr[lang][k].replace
6722             local attr = lbkr[lang][k].attr or -1
6723
6724             if Babel.debug then
6725                 print('*****', p, mode)
6726             end
6727
6728             -- This variable is set in some cases below to the first *byte*
6729             -- after the match, either as found by u.match (faster) or the
6730             -- computed position based on sc if w has changed.
6731             local last_match = 0
6732             local step = 0
6733
6734             -- For every match.
6735             while true do
6736                 if Babel.debug then
6737                     print('====')
6738                 end
6739                 local new -- used when inserting and removing nodes
6740
6741                 local matches = { u.match(w, p, last_match) }
6742
6743                 if #matches < 2 then break end
6744
6745                 -- Get and remove empty captures (with ()'s, which return a
6746                 -- number with the position), and keep actual captures
6747                 -- (from (...)), if any, in matches.
6748                 local first = table.remove(matches, 1)
6749                 local last = table.remove(matches, #matches)
6750                 -- Non re-fetched substrings may contain \31, which separates
6751                 -- subsubstrings.
6752                 if string.find(w:sub(first, last-1), Babel.us_char) then break end
6753
6754                 local save_last = last -- with A()BC()D, points to D

```

```

6755
6756 -- Fix offsets, from bytes to unicode. Explained above.
6757 first = u.len(w:sub(1, first-1)) + 1
6758 last = u.len(w:sub(1, last-1)) -- now last points to C
6759
6760 -- This loop stores in a small table the nodes
6761 -- corresponding to the pattern. Used by 'data' to provide a
6762 -- predictable behavior with 'insert' (w_nodes is modified on
6763 -- the fly), and also access to 'remove'd nodes.
6764 local sc = first-1 -- Used below, too
6765 local data_nodes = {}
6766
6767 local enabled = true
6768 for q = 1, last-first+1 do
6769     data_nodes[q] = w_nodes[sc+q]
6770     if enabled
6771         and attr > -1
6772         and not node.has_attribute(data_nodes[q], attr)
6773     then
6774         enabled = false
6775     end
6776 end
6777
6778 -- This loop traverses the matched substring and takes the
6779 -- corresponding action stored in the replacement list.
6780 -- sc = the position in substr nodes / string
6781 -- rc = the replacement table index
6782 local rc = 0
6783
6784 while rc < last-first+1 do -- for each replacement
6785     if Babel.debug then
6786         print('.....', rc + 1)
6787     end
6788     sc = sc + 1
6789     rc = rc + 1
6790
6791     if Babel.debug then
6792         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6793         local ss = ''
6794         for itt in node.traverse(head) do
6795             if itt.id == 29 then
6796                 ss = ss .. unicode.utf8.char(itt.char)
6797             else
6798                 ss = ss .. '{' .. itt.id .. '}'
6799             end
6800         end
6801         print('*****', ss)
6802     end
6803
6804     local crep = r[rc]
6805     local item = w_nodes[sc]
6806     local item_base = item
6807     local placeholder = Babel.us_char
6808     local d
6809
6810     if crep and crep.data then
6811         item_base = data_nodes[crep.data]
6812     end
6813
6814     if crep then
6815         step = crep.step or 0
6816     end
6817

```

```

6818
6819     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6820         last_match = save_last      -- Optimization
6821         goto next
6822
6823     elseif crep == nil or crep.remove then
6824         node.remove(head, item)
6825         table.remove(w_nodes, sc)
6826         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6827         sc = sc - 1  -- Nothing has been inserted.
6828         last_match = utf8.offset(w, sc+1+step)
6829         goto next
6830
6831     elseif crep and crep.kashida then -- Experimental
6832         node.set_attribute(item,
6833             Babel.attr_kashida,
6834             crep.kashida)
6835         last_match = utf8.offset(w, sc+1+step)
6836         goto next
6837
6838     elseif crep and crep.string then
6839         local str = crep.string(matches)
6840         if str == '' then -- Gather with nil
6841             node.remove(head, item)
6842             table.remove(w_nodes, sc)
6843             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6844             sc = sc - 1  -- Nothing has been inserted.
6845         else
6846             local loop_first = true
6847             for s in string.utfvalues(str) do
6848                 d = node.copy(item_base)
6849                 d.char = s
6850                 if loop_first then
6851                     loop_first = false
6852                     head, new = node.insert_before(head, item, d)
6853                     if sc == 1 then
6854                         word_head = head
6855                     end
6856                     w_nodes[sc] = d
6857                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6858                 else
6859                     sc = sc + 1
6860                     head, new = node.insert_before(head, item, d)
6861                     table.insert(w_nodes, sc, new)
6862                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6863                 end
6864                 if Babel.debug then
6865                     print('.....', 'str')
6866                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6867                 end
6868             end -- for
6869             node.remove(head, item)
6870         end -- if ''
6871         last_match = utf8.offset(w, sc+1+step)
6872         goto next
6873
6874     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6875         d = node.new(7, 0)  -- (disc, discretionary)
6876         d.pre    = Babel.str_to_nodes(crep.pre, matches, item_base)
6877         d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6878         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6879         d.attr = item_base.attr
6880         if crep.pre == nil then -- TeXbook p96

```

```

6881         d.penalty = crep.penalty or tex.hyphenpenalty
6882     else
6883         d.penalty = crep.penalty or tex.exhyphenpenalty
6884     end
6885     placeholder = '|'
6886     head, new = node.insert_before(head, item, d)
6887
6888 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6889     -- ERROR
6890
6891 elseif crep and crep.penalty then
6892     d = node.new(14, 0) -- (penalty, userpenalty)
6893     d.attr = item_base.attr
6894     d.penalty = crep.penalty
6895     head, new = node.insert_before(head, item, d)
6896
6897 elseif crep and crep.space then
6898     -- 655360 = 10 pt = 10 * 65536 sp
6899     d = node.new(12, 13) -- (glue, spaceskip)
6900     local quad = font.getfont(item_base.font).size or 655360
6901     node.setglue(d, crep.space[1] * quad,
6902                  crep.space[2] * quad,
6903                  crep.space[3] * quad)
6904     if mode == 0 then
6905         placeholder = ' '
6906     end
6907     head, new = node.insert_before(head, item, d)
6908
6909 elseif crep and crep.spacefactor then
6910     d = node.new(12, 13) -- (glue, spaceskip)
6911     local base_font = font.getfont(item_base.font)
6912     node.setglue(d,
6913                  crep.spacefactor[1] * base_font.parameters['space'],
6914                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6915                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6916     if mode == 0 then
6917         placeholder = ' '
6918     end
6919     head, new = node.insert_before(head, item, d)
6920
6921 elseif mode == 0 and crep and crep.space then
6922     -- ERROR
6923
6924 end -- ie replacement cases
6925
6926 -- Shared by disc, space and penalty.
6927 if sc == 1 then
6928     word_head = head
6929 end
6930 if crep.insert then
6931     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6932     table.insert(w_nodes, sc, new)
6933     last = last + 1
6934 else
6935     w_nodes[sc] = d
6936     node.remove(head, item)
6937     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6938 end
6939
6940 last_match = utf8.offset(w, sc+1+step)
6941
6942 ::next::
6943

```

```

6944         end -- for each replacement
6945
6946         if Babel.debug then
6947             print('.....', '/')
6948             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6949         end
6950
6951     end -- for match
6952
6953 end -- for patterns
6954
6955 ::next::
6956     word_head = nw
6957 end -- for substring
6958 return head
6959 end
6960
6961 -- This table stores capture maps, numbered consecutively
6962 Babel.capture_maps = {}
6963
6964 -- The following functions belong to the next macro
6965 function Babel.capture_func(key, cap)
6966     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6967     local cnt
6968     local u = unicode.utf8
6969     ret, cnt = ret:gsub('{{[0-9]}|^|+|.|.}', Babel.capture_func_map)
6970     if cnt == 0 then
6971         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
6972             function (n)
6973                 return u.char(tonumber(n, 16))
6974             end)
6975     end
6976     ret = ret:gsub("%[%[%]]%.", '')
6977     ret = ret:gsub("%.%[%[%]]%", '')
6978     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6979 end
6980
6981 function Babel.capt_map(from, mapno)
6982     return Babel.capture_maps[mapno][from] or from
6983 end
6984
6985 -- Handle the {n|abc|ABC} syntax in captures
6986 function Babel.capture_func_map(capno, from, to)
6987     local u = unicode.utf8
6988     from = u.gsub(from, '{{(%x%x%x%x+)}}',
6989         function (n)
6990             return u.char(tonumber(n, 16))
6991         end)
6992     to = u.gsub(to, '{{(%x%x%x%x+)}}',
6993         function (n)
6994             return u.char(tonumber(n, 16))
6995         end)
6996     local froms = {}
6997     for s in string.utfcharacters(from) do
6998         table.insert(froms, s)
6999     end
7000     local cnt = 1
7001     table.insert(Babel.capture_maps, {})
7002     local mlen = table.getn(Babel.capture_maps)
7003     for s in string.utfcharacters(to) do
7004         Babel.capture_maps[mlen][froms[cnt]] = s
7005         cnt = cnt + 1
7006     end

```

```

7007 return "]]..Babel.capt_map(m[" .. capno .. "]," ..
7008         (mlen) .. ").." .. "["
7009 end
7010
7011 -- Create/Extend reversed sorted list of kashida weights:
7012 function Babel.capture_kashida(key, wt)
7013   wt = tonumber(wt)
7014   if Babel.kashida_wts then
7015     for p, q in ipairs(Babel.kashida_wts) do
7016       if wt == q then
7017         break
7018       elseif wt > q then
7019         table.insert(Babel.kashida_wts, p, wt)
7020         break
7021       elseif table.getn(Babel.kashida_wts) == p then
7022         table.insert(Babel.kashida_wts, wt)
7023       end
7024     end
7025   else
7026     Babel.kashida_wts = { wt }
7027   end
7028   return 'kashida = ' .. wt
7029 end
7030 </transforms>

```

12.13 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.


```

7031 (*basic-r)
7032 Babel = Babel or {}
7033
7034 Babel.bidi_enabled = true
7035
7036 require('babel-data-bidi.lua')
7037
7038 local characters = Babel.characters
7039 local ranges = Babel.ranges
7040
7041 local DIR = node.id("dir")
7042
7043 local function dir_mark(head, from, to, outer)
7044   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7045   local d = node.new(DIR)
7046   d.dir = '+' .. dir
7047   node.insert_before(head, from, d)
7048   d = node.new(DIR)
7049   d.dir = '-' .. dir
7050   node.insert_after(head, to, d)
7051 end
7052
7053 function Babel.bidi(head, ispar)
7054   local first_n, last_n          -- first and last char with nums
7055   local last_es                  -- an auxiliary 'last' used with nums
7056   local first_d, last_d          -- first and last char in L/R block
7057   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):

```

7058   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7059   local strong_lr = (strong == 'l') and 'l' or 'r'
7060   local outer = strong
7061
7062   local new_dir = false
7063   local first_dir = false
7064   local inmath = false
7065
7066   local last_lr
7067
7068   local type_n = ''
7069
7070   for item in node.traverse(head) do
7071
7072     -- three cases: glyph, dir, otherwise
7073     if item.id == node.id'glyph'
7074       or (item.id == 7 and item.subtype == 2) then
7075
7076       local itemchar
7077       if item.id == 7 and item.subtype == 2 then
7078         itemchar = item.replace.char
7079       else
7080         itemchar = item.char
7081       end
7082       local chardata = characters[itemchar]
7083       dir = chardata and chardata.d or nil
7084       if not dir then
7085         for nn, et in ipairs(ranges) do
7086           if itemchar < et[1] then
7087             break
7088           elseif itemchar <= et[2] then
7089             dir = et[3]

```

```

7090         break
7091     end
7092 end
7093 end
7094 dir = dir or 'l'
7095 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).

```

7096     if new_dir then
7097         attr_dir = 0
7098         for at in node.traverse(item.attr) do
7099             if at.number == Babel.attr_dir then
7100                 attr_dir = at.value & 0x3
7101             end
7102         end
7103         if attr_dir == 1 then
7104             strong = 'r'
7105         elseif attr_dir == 2 then
7106             strong = 'al'
7107         else
7108             strong = 'l'
7109         end
7110         strong_lr = (strong == 'l') and 'l' or 'r'
7111         outer = strong_lr
7112         new_dir = false
7113     end
7114
7115     if dir == 'nsm' then dir = strong end -- W1

```

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

```

7116     dir_real = dir -- We need dir_real to set strong below
7117     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:

```

7118     if strong == 'al' then
7119         if dir == 'en' then dir = 'an' end -- W2
7120         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7121         strong_lr = 'r' -- W3
7122     end

```

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

```

7123     elseif item.id == node.id'dir' and not inmath then
7124         new_dir = true
7125         dir = nil
7126     elseif item.id == node.id'math' then
7127         inmath = (item.subtype == 0)
7128     else
7129         dir = nil -- Not a char
7130     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>.

```

7131     if dir == 'en' or dir == 'an' or dir == 'et' then
7132         if dir ~= 'et' then
7133             type_n = dir
7134         end

```

```

7135     first_n = first_n or item
7136     last_n = last_es or item
7137     last_es = nil
7138   elseif dir == 'es' and last_n then -- W3+W6
7139     last_es = item
7140   elseif dir == 'cs' then           -- it's right - do nothing
7141   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7142     if strong_lr == 'r' and type_n ~= '' then
7143       dir_mark(head, first_n, last_n, 'r')
7144     elseif strong_lr == 'l' and first_d and type_n == 'an' then
7145       dir_mark(head, first_n, last_n, 'r')
7146       dir_mark(head, first_d, last_d, outer)
7147       first_d, last_d = nil, nil
7148     elseif strong_lr == 'l' and type_n ~= '' then
7149       last_d = last_n
7150     end
7151     type_n = ''
7152     first_n, last_n = nil, nil
7153   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:

```

7154   if dir == 'l' or dir == 'r' then
7155     if dir ~= outer then
7156       first_d = first_d or item
7157       last_d = item
7158     elseif first_d and dir ~= strong_lr then
7159       dir_mark(head, first_d, last_d, outer)
7160       first_d, last_d = nil, nil
7161     end
7162   end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resptly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

7163   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7164     item.char = characters[item.char] and
7165       characters[item.char].m or item.char
7166   elseif (dir or new_dir) and last_lr ~= item then
7167     local mir = outer .. strong_lr .. (dir or outer)
7168     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7169       for ch in node.traverse(node.next(last_lr)) do
7170         if ch == item then break end
7171         if ch.id == node.id'glyph' and characters[ch.char] then
7172           ch.char = characters[ch.char].m or ch.char
7173         end
7174       end
7175     end
7176   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).

```

7177   if dir == 'l' or dir == 'r' then
7178     last_lr = item
7179     strong = dir_real           -- Don't search back - best save now
7180     strong_lr = (strong == 'l') and 'l' or 'r'
7181   elseif new_dir then
7182     last_lr = nil
7183   end
7184 end

```

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

```
7185 if last_lr and outer == 'r' then
7186   for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7187     if characters[ch.char] then
7188       ch.char = characters[ch.char].m or ch.char
7189     end
7190   end
7191 end
7192 if first_n then
7193   dir_mark(head, first_n, last_n, outer)
7194 end
7195 if first_d then
7196   dir_mark(head, first_d, last_d, outer)
7197 end
```

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

```
7198 return node.prev(head) or head
7199 end
7200 </basic-r>
```

And here the Lua code for bidi=basic:

```
7201 <*basic>
7202 Babel = Babel or {}
7203
7204 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7205
7206 Babel.fontmap = Babel.fontmap or {}
7207 Babel.fontmap[0] = {}      -- l
7208 Babel.fontmap[1] = {}      -- r
7209 Babel.fontmap[2] = {}      -- al/an
7210
7211 Babel.bidi_enabled = true
7212 Babel.mirroring_enabled = true
7213
7214 require('babel-data-bidi.lua')
7215
7216 local characters = Babel.characters
7217 local ranges = Babel.ranges
7218
7219 local DIR = node.id('dir')
7220 local GLYPH = node.id('glyph')
7221
7222 local function insert_implicit(head, state, outer)
7223   local new_state = state
7224   if state.sim and state.eim and state.sim ~= state.eim then
7225     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7226     local d = node.new(DIR)
7227     d.dir = '+' .. dir
7228     node.insert_before(head, state.sim, d)
7229     local d = node.new(DIR)
7230     d.dir = '-' .. dir
7231     node.insert_after(head, state.eim, d)
7232   end
7233   new_state.sim, new_state.eim = nil, nil
7234   return head, new_state
7235 end
7236
7237 local function insert_numeric(head, state)
7238   local new
7239   local new_state = state
7240   if state.san and state.ean and state.san ~= state.ean then
7241     local d = node.new(DIR)
```

```

7242     d.dir = '+TLT'
7243     _, new = node.insert_before(head, state.san, d)
7244     if state.san == state.sim then state.sim = new end
7245     local d = node.new(DIR)
7246     d.dir = '-TLT'
7247     _, new = node.insert_after(head, state.ean, d)
7248     if state.ean == state.eim then state.eim = new end
7249 end
7250 new_state.san, new_state.ean = nil, nil
7251 return head, new_state
7252 end
7253
7254 -- TODO - \hbox with an explicit dir can lead to wrong results
7255 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7256 -- was made to improve the situation, but the problem is the 3-dir
7257 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7258 -- well.
7259
7260 function Babel.bidi(head, ispar, hdir)
7261     local d -- d is used mainly for computations in a loop
7262     local prev_d = ''
7263     local new_d = false
7264
7265     local nodes = {}
7266     local outer_first = nil
7267     local inmath = false
7268
7269     local glue_d = nil
7270     local glue_i = nil
7271
7272     local has_en = false
7273     local first_et = nil
7274
7275     local has_hyperlink = false
7276
7277     local ATDIR = Babel.attr_dir
7278
7279     local save_outer
7280     local temp = node.get_attribute(head, ATDIR)
7281     if temp then
7282         temp = temp & 0x3
7283         save_outer = (temp == 0 and 'l') or
7284                     (temp == 1 and 'r') or
7285                     (temp == 2 and 'al')
7286     elseif ispar then -- Or error? Shouldn't happen
7287         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7288     else -- Or error? Shouldn't happen
7289         save_outer = ('TRT' == hdir) and 'r' or 'l'
7290     end
7291     -- when the callback is called, we are just _after_ the box,
7292     -- and the textdir is that of the surrounding text
7293     -- if not ispar and hdir ~= tex.textdir then
7294     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7295     -- end
7296     local outer = save_outer
7297     local last = outer
7298     -- 'al' is only taken into account in the first, current loop
7299     if save_outer == 'al' then save_outer = 'r' end
7300
7301     local fontmap = Babel.fontmap
7302
7303     for item in node.traverse(head) do
7304

```

```

7305 -- In what follows, #node is the last (previous) node, because the
7306 -- current one is not added until we start processing the neutrals.
7307
7308 -- three cases: glyph, dir, otherwise
7309 if item.id == GLYPH
7310     or (item.id == 7 and item.subtype == 2) then
7311
7312     local d_font = nil
7313     local item_r
7314     if item.id == 7 and item.subtype == 2 then
7315         item_r = item.replace -- automatic discs have just 1 glyph
7316     else
7317         item_r = item
7318     end
7319     local chardata = characters[item_r.char]
7320     d = chardata and chardata.d or nil
7321     if not d or d == 'nsm' then
7322         for nn, et in ipairs(ranges) do
7323             if item_r.char < et[1] then
7324                 break
7325             elseif item_r.char <= et[2] then
7326                 if not d then d = et[3]
7327                 elseif d == 'nsm' then d_font = et[3]
7328                 end
7329                 break
7330             end
7331         end
7332     end
7333     d = d or 'l'
7334
7335     -- A short 'pause' in bidi for mapfont
7336     d_font = d_font or d
7337     d_font = (d_font == 'l' and 0) or
7338             (d_font == 'nsm' and 0) or
7339             (d_font == 'r' and 1) or
7340             (d_font == 'al' and 2) or
7341             (d_font == 'an' and 2) or nil
7342     if d_font and fontmap and fontmap[d_font][item_r.font] then
7343         item_r.font = fontmap[d_font][item_r.font]
7344     end
7345
7346     if new_d then
7347         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7348         if inmath then
7349             attr_d = 0
7350         else
7351             attr_d = node.get_attribute(item, ATDIR)
7352             attr_d = attr_d & 0x3
7353         end
7354         if attr_d == 1 then
7355             outer_first = 'r'
7356             last = 'r'
7357         elseif attr_d == 2 then
7358             outer_first = 'r'
7359             last = 'al'
7360         else
7361             outer_first = 'l'
7362             last = 'l'
7363         end
7364         outer = last
7365         has_en = false
7366         first_et = nil
7367         new_d = false

```

```

7368     end
7369
7370     if glue_d then
7371         if (d == 'l' and 'l' or 'r') ~= glue_d then
7372             table.insert(nodes, {glue_i, 'on', nil})
7373         end
7374         glue_d = nil
7375         glue_i = nil
7376     end
7377
7378     elseif item.id == DIR then
7379         d = nil
7380
7381         if head ~= item then new_d = true end
7382
7383     elseif item.id == node.id'glue' and item.subtype == 13 then
7384         glue_d = d
7385         glue_i = item
7386         d = nil
7387
7388     elseif item.id == node.id'math' then
7389         inmath = (item.subtype == 0)
7390
7391     elseif item.id == 8 and item.subtype == 19 then
7392         has_hyperlink = true
7393
7394     else
7395         d = nil
7396     end
7397
7398     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7399     if last == 'al' and d == 'en' then
7400         d = 'an'          -- W3
7401     elseif last == 'al' and (d == 'et' or d == 'es') then
7402         d = 'on'          -- W6
7403     end
7404
7405     -- EN + CS/ES + EN      -- W4
7406     if d == 'en' and #nodes >= 2 then
7407         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7408             and nodes[#nodes-1][2] == 'en' then
7409             nodes[#nodes][2] = 'en'
7410         end
7411     end
7412
7413     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
7414     if d == 'an' and #nodes >= 2 then
7415         if (nodes[#nodes][2] == 'cs')
7416             and nodes[#nodes-1][2] == 'an' then
7417             nodes[#nodes][2] = 'an'
7418         end
7419     end
7420
7421     -- ET/EN              -- W5 + W7->1 / W6->on
7422     if d == 'et' then
7423         first_et = first_et or (#nodes + 1)
7424     elseif d == 'en' then
7425         has_en = true
7426         first_et = first_et or (#nodes + 1)
7427     elseif first_et then -- d may be nil here !
7428         if has_en then
7429             if last == 'l' then
7430                 temp = 'l'    -- W7

```

```

7431         else
7432             temp = 'en'    -- W5
7433         end
7434     else
7435         temp = 'on'        -- W6
7436     end
7437     for e = first_et, #nodes do
7438         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7439     end
7440     first_et = nil
7441     has_en = false
7442 end
7443
7444 -- Force mathdir in math if ON (currently works as expected only
7445 -- with 'l')
7446 if inmath and d == 'on' then
7447     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7448 end
7449
7450 if d then
7451     if d == 'al' then
7452         d = 'r'
7453         last = 'al'
7454     elseif d == 'l' or d == 'r' then
7455         last = d
7456     end
7457     prev_d = d
7458     table.insert(nodes, {item, d, outer_first})
7459 end
7460
7461 outer_first = nil
7462
7463 end
7464
7465 -- TODO -- repeated here in case EN/ET is the last node. Find a
7466 -- better way of doing things:
7467 if first_et then    -- dir may be nil here !
7468     if has_en then
7469         if last == 'l' then
7470             temp = 'l'    -- W7
7471         else
7472             temp = 'en'    -- W5
7473         end
7474     else
7475         temp = 'on'        -- W6
7476     end
7477     for e = first_et, #nodes do
7478         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7479     end
7480 end
7481
7482 -- dummy node, to close things
7483 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7484
7485 ----- NEUTRAL -----
7486
7487 outer = save_outer
7488 last = outer
7489
7490 local first_on = nil
7491
7492 for q = 1, #nodes do
7493     local item

```



```

7494
7495     local outer_first = nodes[q][3]
7496     outer = outer_first or outer
7497     last = outer_first or last
7498
7499     local d = nodes[q][2]
7500     if d == 'an' or d == 'en' then d = 'r' end
7501     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7502
7503     if d == 'on' then
7504         first_on = first_on or q
7505     elseif first_on then
7506         if last == d then
7507             temp = d
7508         else
7509             temp = outer
7510         end
7511         for r = first_on, q - 1 do
7512             nodes[r][2] = temp
7513             item = nodes[r][1] -- MIRRORING
7514             if Babel.mirroring_enabled and item.id == GLYPH
7515                 and temp == 'r' and characters[item.char] then
7516                 local font_mode = ''
7517                 if item.font > 0 and font.fonts[item.font].properties then
7518                     font_mode = font.fonts[item.font].properties.mode
7519                 end
7520                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7521                     item.char = characters[item.char].m or item.char
7522                 end
7523             end
7524         end
7525         first_on = nil
7526     end
7527
7528     if d == 'r' or d == 'l' then last = d end
7529 end
7530
7531 ----- IMPLICIT, REORDER -----
7532
7533 outer = save_outer
7534 last = outer
7535
7536 local state = {}
7537 state.has_r = false
7538
7539 for q = 1, #nodes do
7540
7541     local item = nodes[q][1]
7542
7543     outer = nodes[q][3] or outer
7544
7545     local d = nodes[q][2]
7546
7547     if d == 'nsm' then d = last end -- W1
7548     if d == 'en' then d = 'an' end
7549     local isdir = (d == 'r' or d == 'l')
7550
7551     if outer == 'l' and d == 'an' then
7552         state.san = state.san or item
7553         state.ean = item
7554     elseif state.san then
7555         head, state = insert_numeric(head, state)
7556     end

```

```

7557
7558   if outer == 'l' then
7559       if d == 'an' or d == 'r' then      -- im -> implicit
7560           if d == 'r' then state.has_r = true end
7561           state.sim = state.sim or item
7562           state.eim = item
7563       elseif d == 'l' and state.sim and state.has_r then
7564           head, state = insert_implicit(head, state, outer)
7565       elseif d == 'l' then
7566           state.sim, state.eim, state.has_r = nil, nil, false
7567       end
7568   else
7569       if d == 'an' or d == 'l' then
7570           if nodes[q][3] then -- nil except after an explicit dir
7571               state.sim = item -- so we move sim 'inside' the group
7572           else
7573               state.sim = state.sim or item
7574           end
7575           state.eim = item
7576       elseif d == 'r' and state.sim then
7577           head, state = insert_implicit(head, state, outer)
7578       elseif d == 'r' then
7579           state.sim, state.eim = nil, nil
7580       end
7581   end
7582
7583   if isdir then
7584       last = d      -- Don't search back - best save now
7585   elseif d == 'on' and state.san then
7586       state.san = state.san or item
7587       state.ean = item
7588   end
7589
7590 end
7591
7592 head = node.prev(head) or head
7593
7594 ----- FIX HYPERLINKS -----
7595
7596 if has_hyperlink then
7597     local flag, linking = 0, 0
7598     for item in node.traverse(head) do
7599         if item.id == DIR then
7600             if item.dir == '+TRT' or item.dir == '+TLT' then
7601                 flag = flag + 1
7602             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7603                 flag = flag - 1
7604             end
7605             elseif item.id == 8 and item.subtype == 19 then
7606                 linking = flag
7607             elseif item.id == 8 and item.subtype == 20 then
7608                 if linking > 0 then
7609                     if item.prev.id == DIR and
7610                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7611                         d = node.new(DIR)
7612                         d.dir = item.prev.dir
7613                         node.remove(head, item.prev)
7614                         node.insert_after(head, item, d)
7615                     end
7616                 end
7617                 linking = 0
7618             end
7619         end

```

```

7620 end
7621
7622 return head
7623 end
7624 </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.

```

7625 <*nil>
7626 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language
7627 \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.

```

7628 \ifx\l@nil\@undefined
7629 \newlanguage\l@nil
7630 \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
7631 \let\bbl@elt\relax
7632 \edef\bbl@languages{% Add it to the list of languages
7633 \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
7634 \fi

```

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

```

7635 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
7636 \let\captionnil\@empty
7637 \let\datenil\@empty

```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```

7638 \def\bbl@inidata@nil{%
7639 \bbl@elt{identification}{tag.ini}{und}%
7640 \bbl@elt{identification}{load.level}{0}%
7641 \bbl@elt{identification}{charset}{utf8}%
7642 \bbl@elt{identification}{version}{1.0}%
7643 \bbl@elt{identification}{date}{2022-05-16}%
7644 \bbl@elt{identification}{name.local}{nil}%
7645 \bbl@elt{identification}{name.english}{nil}%
7646 \bbl@elt{identification}{name.babel}{nil}%
7647 \bbl@elt{identification}{tag.bcp47}{und}%
7648 \bbl@elt{identification}{language.tag.bcp47}{und}%

```

```

7649 \bbl@elt{identification}{tag.opentype}{dflt}%
7650 \bbl@elt{identification}{script.name}{Latin}%
7651 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7652 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7653 \bbl@elt{identification}{level}{1}%
7654 \bbl@elt{identification}{encodings}{}%
7655 \bbl@elt{identification}{derivate}{no}}
7656 \@namedef{bbl@tbcp@nil}{und}
7657 \@namedef{bbl@lbcp@nil}{und}
7658 \@namedef{bbl@lotf@nil}{dflt}
7659 \@namedef{bbl@elname@nil}{nil}
7660 \@namedef{bbl@lname@nil}{nil}
7661 \@namedef{bbl@esname@nil}{Latin}
7662 \@namedef{bbl@sname@nil}{Latin}
7663 \@namedef{bbl@sbc@nil}{Latn}
7664 \@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.

```

7665 \ldf@finish{nil}
7666 \</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.

```

7667 <<Compute Julian day>> ≡
7668 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7669 \def\bbl@cs@gregleap#1{%
7670   (\bbl@fpmo{#1}{4} == 0) &&
7671   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7672 \def\bbl@cs@jd#1#2#3{% year, month, day
7673   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7674     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7675     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7676     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }%
7677 <</Compute Julian day>>

```

15.1 Islamic

The code for the Civil calendar is based on it, too.

```

7678 <ca-islamic>
7679 \ExplSyntaxOn
7680 <<Compute Julian day>>
7681 % == islamic (default)
7682 % Not yet implemented
7683 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar.

```

7684 \def\bbl@cs@isltojd#1#2#3{% year, month, day
7685   ((#3 + ceil(29.5 * (#2 - 1)) +
7686     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7687     1948439.5) - 1) }
7688 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7689 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7690 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7691 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7692 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7693 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7694   \edef\bbl@tempa{%

```

```

7695 \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7696 \edef#5{%
7697 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7698 \edef#6{\fp_eval:n{
7699 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7700 \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).

```

7701 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7702 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7703 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7704 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7705 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7706 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7707 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7708 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7709 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7710 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7711 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7712 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7713 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7714 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7715 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7716 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7717 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7718 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7719 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7720 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7721 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7722 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7723 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7724 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7725 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7726 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7727 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7728 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7729 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7730 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7731 65401,65431,65460,65490,65520}
7732 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr{x}{+1}}
7733 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr{x}}
7734 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr{x}{-1}}
7735 \def\bbl@ca@islamcuqr{x#1#2-#3-#4\@#5#6#7}%
7736 \ifnum#2>2014 \ifnum#2<2038
7737 \bbl@afterfi\expandafter\@gobble
7738 \fi\fi
7739 {\bbl@error{Year-out-of-range}{The~allowed~range-is~2014-2038}}%
7740 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7741 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7742 \count@\@ne
7743 \bbl@foreach\bbl@cs@umalqura@data{%
7744 \advance\count@\@ne
7745 \ifnum##1>\bbl@tempd\else
7746 \edef\bbl@tempe{\the\count@}%
7747 \edef\bbl@tempb{##1}%
7748 \fi}%
7749 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
7750 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7751 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7752 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%

```

```

7753 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}
7754 \ExplSyntaxOff
7755 \bbl@add\bbl@precalendar{%
7756 \bbl@replace\bbl@ld@calendar{-civil}}}%
7757 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
7758 \bbl@replace\bbl@ld@calendar{+}}}%
7759 \bbl@replace\bbl@ld@calendar{-}}}%
7760 </ca-islamic>

```

16 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaption 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 hebcal.sty

```

7761 <*ca-hebrew>
7762 \newcount\bbl@cntcommon
7763 \def\bbl@remainder#1#2#3{%
7764 #3=#1\relax
7765 \divide #3 by #2\relax
7766 \multiply #3 by -#2\relax
7767 \advance #3 by #1\relax}%
7768 \newif\ifbbl@divisible
7769 \def\bbl@checkifdivisible#1#2{%
7770 {\countdef\tmp=0
7771 \bbl@remainder{#1}{#2}{\tmp}%
7772 \ifnum \tmp=0
7773 \global\bbl@divisibletrue
7774 \else
7775 \global\bbl@divisiblefalse
7776 \fi}}
7777 \newif\ifbbl@gregleap
7778 \def\bbl@ifgregleap#1{%
7779 \bbl@checkifdivisible{#1}{4}%
7780 \ifbbl@divisible
7781 \bbl@checkifdivisible{#1}{100}%
7782 \ifbbl@divisible
7783 \bbl@checkifdivisible{#1}{400}%
7784 \ifbbl@divisible
7785 \bbl@gregleaptrue
7786 \else
7787 \bbl@gregleapfalse
7788 \fi
7789 \else
7790 \bbl@gregleaptrue
7791 \fi
7792 \else
7793 \bbl@gregleapfalse
7794 \fi
7795 \ifbbl@gregleap}
7796 \def\bbl@gregdayspriormonths#1#2#3{%
7797 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7798 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7799 \bbl@ifgregleap{#2}%
7800 \ifnum #1 > 2
7801 \advance #3 by 1
7802 \fi
7803 \fi
7804 \global\bbl@cntcommon=#3}%
7805 #3=\bbl@cntcommon}
7806 \def\bbl@gregdaysprioryears#1#2{%
7807 {\countdef\tmpc=4
7808 \countdef\tmpb=2

```

```

7809 \tmpb=#1\relax
7810 \advance \tmpb by -1
7811 \tmpc=\tmpb
7812 \multiply \tmpc by 365
7813 #2=\tmpc
7814 \tmpc=\tmpb
7815 \divide \tmpc by 4
7816 \advance #2 by \tmpc
7817 \tmpc=\tmpb
7818 \divide \tmpc by 100
7819 \advance #2 by -\tmpc
7820 \tmpc=\tmpb
7821 \divide \tmpc by 400
7822 \advance #2 by \tmpc
7823 \global\bbl@cntcommon=#2\relax}%
7824 #2=\bbl@cntcommon}
7825 \def\bbl@absfromgreg#1#2#3#4{%
7826 {\countdef\tmpd=0
7827 #4=#1\relax
7828 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7829 \advance #4 by \tmpd
7830 \bbl@gregdaysprioryears{#3}{\tmpd}%
7831 \advance #4 by \tmpd
7832 \global\bbl@cntcommon=#4\relax}%
7833 #4=\bbl@cntcommon}
7834 \newif\ifbbl@hebrleap
7835 \def\bbl@checkleaphebyear#1{%
7836 {\countdef\tmpa=0
7837 \countdef\tmpb=1
7838 \tmpa=#1\relax
7839 \multiply \tmpa by 7
7840 \advance \tmpa by 1
7841 \bbl@remainder{\tmpa}{19}{\tmpb}%
7842 \ifnum \tmpb < 7
7843 \global\bbl@hebrleaptrue
7844 \else
7845 \global\bbl@hebrleapfalse
7846 \fi}}
7847 \def\bbl@hebrlapsedmonths#1#2{%
7848 {\countdef\tmpa=0
7849 \countdef\tmpb=1
7850 \countdef\tmpc=2
7851 \tmpa=#1\relax
7852 \advance \tmpa by -1
7853 #2=\tmpa
7854 \divide #2 by 19
7855 \multiply #2 by 235
7856 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7857 \tmpc=\tmpb
7858 \multiply \tmpb by 12
7859 \advance #2 by \tmpb
7860 \multiply \tmpc by 7
7861 \advance \tmpc by 1
7862 \divide \tmpc by 19
7863 \advance #2 by \tmpc
7864 \global\bbl@cntcommon=#2}%
7865 #2=\bbl@cntcommon}
7866 \def\bbl@hebrlapseddays#1#2{%
7867 {\countdef\tmpa=0
7868 \countdef\tmpb=1
7869 \countdef\tmpc=2
7870 \bbl@hebrlapsedmonths{#1}{#2}%
7871 \tmpa=#2\relax

```

```

7872 \multiply \tmpa by 13753
7873 \advance \tmpa by 5604
7874 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7875 \divide \tmpa by 25920
7876 \multiply #2 by 29
7877 \advance #2 by 1
7878 \advance #2 by \tmpa
7879 \bbl@remainder{#2}{7}{\tmpa}%
7880 \ifnum \tmpc < 19440
7881   \ifnum \tmpc < 9924
7882   \else
7883     \ifnum \tmpa=2
7884       \bbl@checkleaphebrewyear{#1}% of a common year
7885       \ifbbl@hebrleap
7886       \else
7887         \advance #2 by 1
7888       \fi
7889     \fi
7890   \fi
7891   \ifnum \tmpc < 16789
7892   \else
7893     \ifnum \tmpa=1
7894       \advance #1 by -1
7895       \bbl@checkleaphebrewyear{#1}% at the end of leap year
7896       \ifbbl@hebrleap
7897         \advance #2 by 1
7898       \fi
7899     \fi
7900   \fi
7901   \else
7902     \advance #2 by 1
7903   \fi
7904   \bbl@remainder{#2}{7}{\tmpa}%
7905   \ifnum \tmpa=0
7906     \advance #2 by 1
7907   \else
7908     \ifnum \tmpa=3
7909       \advance #2 by 1
7910     \else
7911       \ifnum \tmpa=5
7912         \advance #2 by 1
7913       \fi
7914     \fi
7915   \fi
7916   \global\bbl@cntcommon=#2\relax}%
7917 #2=\bbl@cntcommon}
7918 \def\bbl@daysinhebrewyear#1#2{%
7919   {\countdef\tmpe=12
7920    \bbl@hebreleapseddays{#1}{\tmpe}%
7921    \advance #1 by 1
7922    \bbl@hebreleapseddays{#1}{#2}%
7923    \advance #2 by -\tmpe
7924    \global\bbl@cntcommon=#2}%
7925 #2=\bbl@cntcommon}
7926 \def\bbl@hebrdayspriormonths#1#2#3{%
7927   {\countdef\tmpf= 14
7928    #3=\ifcase #1\relax
7929      0 \or
7930      0 \or
7931      30 \or
7932      59 \or
7933      89 \or
7934      118 \or

```



```

7935         148 \or
7936         148 \or
7937         177 \or
7938         207 \or
7939         236 \or
7940         266 \or
7941         295 \or
7942         325 \or
7943         400
7944 \fi
7945 \bbl@checkleaphebryear{#2}%
7946 \ifbbl@hebrleap
7947     \ifnum #1 > 6
7948         \advance #3 by 30
7949     \fi
7950 \fi
7951 \bbl@daysinhebryear{#2}{\tmpf}%
7952 \ifnum #1 > 3
7953     \ifnum \tmpf=353
7954         \advance #3 by -1
7955     \fi
7956     \ifnum \tmpf=383
7957         \advance #3 by -1
7958     \fi
7959 \fi
7960 \ifnum #1 > 2
7961     \ifnum \tmpf=355
7962         \advance #3 by 1
7963     \fi
7964     \ifnum \tmpf=385
7965         \advance #3 by 1
7966     \fi
7967 \fi
7968 \global\bbl@cntcommon=#3\relax}%
7969 #3=\bbl@cntcommon}
7970 \def\bbl@absfromhebr#1#2#3#4{%
7971     {#4=#1\relax
7972     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7973     \advance #4 by #1\relax
7974     \bbl@hebrrelapseddays{#3}{#1}%
7975     \advance #4 by #1\relax
7976     \advance #4 by -1373429
7977     \global\bbl@cntcommon=#4\relax}%
7978 #4=\bbl@cntcommon}
7979 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7980     {\countdef\tmpx= 17
7981     \countdef\tmpy= 18
7982     \countdef\tmpz= 19
7983     #6=#3\relax
7984     \global\advance #6 by 3761
7985     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7986     \tmpz=1 \tmpy=1
7987     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7988     \ifnum \tmpx > #4\relax
7989         \global\advance #6 by -1
7990         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7991     \fi
7992     \advance #4 by -\tmpx
7993     \advance #4 by 1
7994     #5=#4\relax
7995     \divide #5 by 30
7996     \loop
7997         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%

```

```

7998      \ifnum \tmpx < #4\relax
7999      \advance #5 by 1
8000      \tmpy=\tmpx
8001      \repeat
8002      \global\advance #5 by -1
8003      \global\advance #4 by -\tmpy}}
8004 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8005 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8006 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8007   \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8008   \bbl@hebrfromgreg
8009   {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8010   {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8011   \edef#4{\the\bbl@hebryear}%
8012   \edef#5{\the\bbl@hebrmonth}%
8013   \edef#6{\the\bbl@hebrday}}
8014 \</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).

```

8015 \<ca-persian>
8016 \ExplSyntaxOn
8017 \<\Compute Julian day>
8018 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8019 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8020 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8021   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8022   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8023     \bbl@afterfi\expandafter\@gobble
8024   \fi\fi
8025   {\bbl@error{Year-out-of-range}{The~allowed~range-is~2013-2050}}%
8026   \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8027   \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8028   \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8029   \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8030   \ifnum\bbl@tempc<\bbl@tempb
8031     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8032     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8033     \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8034     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8035   \fi
8036   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8037   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8038   \edef#5{\fp_eval:n{% set Jalali month
8039     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8040   \edef#6{\fp_eval:n{% set Jalali day
8041     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}
8042 \ExplSyntaxOff
8043 \</ca-persian>

```

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.

```

8044 \<ca-coptic>
8045 \ExplSyntaxOn

```

```

8046 <<Compute Julian day>>
8047 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8048   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8049   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8050   \edef#4{\fp_eval:n{%
8051     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8052   \edef\bbl@tempc{\fp_eval:n{%
8053     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8054   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8055   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8056 \ExplSyntaxOff
8057 </ca-coptic>
8058 <*ca-ethiopic>
8059 \ExplSyntaxOn
8060 <<Compute Julian day>>
8061 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8062   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8063   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8064   \edef#4{\fp_eval:n{%
8065     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8066   \edef\bbl@tempc{\fp_eval:n{%
8067     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8068   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8069   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8070 \ExplSyntaxOff
8071 </ca-ethiopic>

```

19 Buddhist

That's very simple.

```

8072 <*ca-buddhist>
8073 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8074   \edef#4{\number\numexpr#1+543\relax}%
8075   \edef#5{#2}%
8076   \edef#6{#3}%
8077 </ca-buddhist>

```

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 `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```

8078 <*bplain | blplain>
8079 \catcode`\{=1 % left brace is begin-group character
8080 \catcode`\}=2 % right brace is end-group character
8081 \catcode`\#=6 % hash mark is macro parameter character

```

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

```
8082 \openin 0 hyphen.cfg
8083 \ifeof0
8084 \else
8085   \let\input
```

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

```
8086   \def\input #1 {%
8087     \let\input\input
8088     \a hyphen.cfg
8089     \let\input\undefined
8090   }
8091 \fi
8092 </bplain | blplain>
```

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

```
8093 <bplain>\a plain.tex
8094 <blplain>\a lplain.tex
```

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

```
8095 <bplain>\def\fmtname{babel-plain}
8096 <blplain>\def\fmtname{babel-lplain}
```

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

20.2 Emulating some \LaTeX features

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

```
8097 <<*Emulate LaTeX>> ≡
8098 \def\@empty{}
8099 \def\loadlocalcfg#1{%
8100   \openin0#1.cfg
8101   \ifeof0
8102     \closein0
8103   \else
8104     \closein0
8105     {\immediate\write16{*****}%
8106      \immediate\write16{* Local config file #1.cfg used}%
8107      \immediate\write16{**}%
8108     }
8109     \input #1.cfg\relax
8110   \fi
8111   \@endofldf}
```

20.3 General tools

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

```
8112 \long\def\@firstofone#1{#1}
8113 \long\def\@firstoftwo#1#2{#1}
8114 \long\def\@secondoftwo#1#2{#2}
8115 \def\@nnil{\nil}
8116 \def\@gobbletwo#1#2{}
8117 \def\@ifstar#1{\ifnextchar *{\@firstoftwo{#1}}}
```

```

8118 \def\@star@or@long#1{%
8119   \@ifstar
8120   {\let\l@ngrel@x\relax#1}%
8121   {\let\l@ngrel@x\long#1}}
8122 \let\l@ngrel@x\relax
8123 \def\@car#1#2\@nil{#1}
8124 \def\@cdr#1#2\@nil{#2}
8125 \let\@typeset@protect\relax
8126 \let\protected@edef\edef
8127 \long\def\@gobble#1{}
8128 \edef\@backslashchar{\expandafter\@gobble\string\}
8129 \def\strip@prefix#1>{}
8130 \def\g@addto@macro#1#2{%
8131   \toks@\expandafter{#1#2}%
8132   \xdef#1{\the\toks@}}
8133 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8134 \def\@nameuse#1{\csname #1\endcsname}
8135 \def\@ifundefined#1{%
8136   \expandafter\ifx\csname#1\endcsname\relax
8137     \expandafter\@firstoftwo
8138   \else
8139     \expandafter\@secondoftwo
8140   \fi}
8141 \def\@expandtwoargs#1#2#3{%
8142   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8143 \def\zap@space#1 #2{%
8144   #1%
8145   \ifx#2\@empty\else\expandafter\zap@space\fi
8146   #2}
8147 \let\bbl@trace\@gobble
8148 \def\bbl@error#1#2{%
8149   \begingroup
8150     \newlinechar=`^^J
8151     \def\{^^J(babel) }%
8152     \errhelp{#2}\errmessage{\#1}%
8153   \endgroup}
8154 \def\bbl@warning#1{%
8155   \begingroup
8156     \newlinechar=`^^J
8157     \def\{^^J(babel) }%
8158     \message{\#1}%
8159   \endgroup}
8160 \let\bbl@infowarn\bbl@warning
8161 \def\bbl@info#1{%
8162   \begingroup
8163     \newlinechar=`^^J
8164     \def\{^^J}%
8165     \wlog{#1}%
8166   \endgroup}

```

$\LaTeX 2\epsilon$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

8167 \ifx\@preamblecmds\@undefined
8168   \def\@preamblecmds{}
8169 \fi
8170 \def\@onlypreamble#1{%
8171   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8172     \@preamblecmds\do#1}}
8173 \@onlypreamble\@onlypreamble

```

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

```

8174 \def\begin{document}{%
8175   \@begin{document}hook
8176   \global\let\@begin{document}hook\@undefined

```

```

8177 \def\do##1{\global\let##1\@undefined}%
8178 \@preamblecmds
8179 \global\let\do\noexpand}

8180 \ifx\@begindocumenthook\@undefined
8181 \def\@begindocumenthook{}
8182 \fi
8183 \@onlypreamble\@begindocumenthook
8184 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8185 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8186 \@onlypreamble\AtEndOfPackage
8187 \def\@endofldf{}
8188 \@onlypreamble\@endofldf
8189 \let\bb1@afterlang\@empty
8190 \chardef\bb1@opt@hyphenmap\z@

```

\TeX 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.

```

8191 \catcode`\&=\z@
8192 \ifx&\if@files\@undefined
8193 \expandafter\let\csname if@files\expandafter\endcsname
8194 \csname iffalse\endcsname
8195 \fi
8196 \catcode`\&=4

```

Mimick \TeX 's commands to define control sequences.

```

8197 \def\newcommand{\@star@or@long\new@command}
8198 \def\new@command#1{%
8199 \@testopt{\@newcommand#1}0}
8200 \def\@newcommand#1[#2]{%
8201 \@ifnextchar [{\@xargdef#1[#2]}%
8202 {\@argdef#1[#2]}}
8203 \long\def\@argdef#1[#2]#3{%
8204 \@yargdef#1\@ne{#2}{#3}}
8205 \long\def\@xargdef#1[#2][#3]#4{%
8206 \expandafter\def\expandafter#1\expandafter{%
8207 \expandafter\@protected@testopt\expandafter #1%
8208 \csname\string#1\expandafter\endcsname{#3}}}%
8209 \expandafter\@yargdef\csname\string#1\endcsname
8210 \tw@{#2}{#4}}
8211 \long\def\@yargdef#1#2#3{%
8212 \@tempcnta#3\relax
8213 \advance \@tempcnta \@ne
8214 \let\@hash@\relax
8215 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8216 \@tempcntb #2%
8217 \@whilenum\@tempcntb <\@tempcnta
8218 \do{%
8219 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8220 \advance\@tempcntb \@ne}%
8221 \let\@hash@##%
8222 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8223 \def\providecommand{\@star@or@long\provide@command}
8224 \def\provide@command#1{%
8225 \begingroup
8226 \escapechar\m@ne\edef\@gtempa{\string#1}%
8227 \endgroup
8228 \expandafter\@ifundefined\@gtempa
8229 {\def\reserved@a{\new@command#1}}%
8230 {\let\reserved@a\relax

```

```

8231 \def\reserved@a{\new@command\reserved@a}}%
8232 \reserved@a}%

8233 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8234 \def\declare@robustcommand#1{%
8235 \edef\reserved@a{\string#1}%
8236 \def\reserved@b{#1}%
8237 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8238 \edef#1{%
8239 \ifx\reserved@a\reserved@b
8240 \noexpand\x@protect
8241 \noexpand#1%
8242 \fi
8243 \noexpand\protect
8244 \expandafter\noexpand\csname
8245 \expandafter\@gobble\string#1 \endcsname
8246 }%
8247 \expandafter\new@command\csname
8248 \expandafter\@gobble\string#1 \endcsname
8249 }
8250 \def\x@protect#1{%
8251 \ifx\protect\@typeset@protect\else
8252 \@x@protect#1%
8253 \fi
8254 }
8255 \catcode\&=\z@ % Trick to hide conditionals
8256 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

8257 \def\bbl@tempa{\csname newif\endcsname&fin@}
8258 \catcode\&=4
8259 \ifx\in@\@undefined
8260 \def\in@#1#2{%
8261 \def\in@@##1##2##3\in@@{%
8262 \ifx\in@@#2\in@false\else\in@true\fi}%
8263 \in@@#2#1\in@\in@@}
8264 \else
8265 \let\bbl@tempa\@empty
8266 \fi
8267 \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).

```

8268 \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.

```

8269 \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.

```

8270 \ifx\@tempcnta\@undefined
8271 \csname newcount\endcsname\@tempcnta\relax
8272 \fi
8273 \ifx\@tempcntb\@undefined
8274 \csname newcount\endcsname\@tempcntb\relax
8275 \fi

```

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

```

8276 \ifx\bye\@undefined
8277   \advance\count10 by -2\relax
8278 \fi
8279 \ifx\@ifnextchar\@undefined
8280   \def\@ifnextchar#1#2#3{%
8281     \let\reserved@d=#1%
8282     \def\reserved@a{#2}\def\reserved@b{#3}%
8283     \futurelet\@let@token\@ifnch}
8284   \def\@ifnch{%
8285     \ifx\@let@token\@sptoken
8286       \let\reserved@c\@xifnch
8287     \else
8288       \ifx\@let@token\reserved@d
8289         \let\reserved@c\reserved@a
8290       \else
8291         \let\reserved@c\reserved@b
8292       \fi
8293     \fi
8294     \reserved@c}
8295   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
8296   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8297 \fi
8298 \def\@testopt#1#2{%
8299   \@ifnextchar[#{#1}{#1[#2]}}
8300 \def\@protected@testopt#1{%
8301   \ifx\protect\@typeset@protect
8302     \expandafter\@testopt
8303   \else
8304     \@x@protect#1%
8305   \fi}
8306 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8307   #2\relax}\fi}
8308 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8309   \else\expandafter\@gobble\fi{#1}}

```

20.4 Encoding related macros

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

```

8310 \def\DeclareTextCommand{%
8311   \@dec@text@cmd\providecommand
8312 }
8313 \def\ProvideTextCommand{%
8314   \@dec@text@cmd\providecommand
8315 }
8316 \def\DeclareTextSymbol#1#2#3{%
8317   \@dec@text@cmd\chardef#1{#2}#3\relax
8318 }
8319 \def\@dec@text@cmd#1#2#3{%
8320   \expandafter\def\expandafter#2%
8321     \expandafter{%
8322       \csname#3-cmd\expandafter\endcsname
8323       \expandafter#2%
8324       \csname#3\string#2\endcsname
8325     }%
8326 %   \let\@ifdefinable\@rc@ifdefinable
8327   \expandafter#1\csname#3\string#2\endcsname
8328 }
8329 \def\@current@cmd#1{%
8330   \ifx\protect\@typeset@protect\else
8331     \noexpand#1\expandafter\@gobble

```



```

8332 \fi
8333 }
8334 \def\@changed@cmd#1#2{%
8335 \ifx\protect\@typeset@protect
8336 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8337 \expandafter\ifx\csname ?\string#1\endcsname\relax
8338 \expandafter\def\csname ?\string#1\endcsname{%
8339 \@changed@x@err{#1}%
8340 }%
8341 \fi
8342 \global\expandafter\let
8343 \csname\cf@encoding\string#1\expandafter\endcsname
8344 \csname ?\string#1\endcsname
8345 \fi
8346 \csname\cf@encoding\string#1%
8347 \expandafter\endcsname
8348 \else
8349 \noexpand#1%
8350 \fi
8351 }
8352 \def\@changed@x@err#1{%
8353 \errhelp{Your command will be ignored, type <return> to proceed}%
8354 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8355 \def\DeclareTextCommandDefault#1{%
8356 \DeclareTextCommand#1?%
8357 }
8358 \def\ProvideTextCommandDefault#1{%
8359 \ProvideTextCommand#1?%
8360 }
8361 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8362 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8363 \def\DeclareTextAccent#1#2#3{%
8364 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8365 }
8366 \def\DeclareTextCompositeCommand#1#2#3#4{%
8367 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8368 \edef\reserved@b{\string##1}%
8369 \edef\reserved@c{%
8370 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8371 \ifx\reserved@b\reserved@c
8372 \expandafter\expandafter\expandafter\ifx
8373 \expandafter\@car\reserved@a\relax\relax\@nil
8374 \@text@composite
8375 \else
8376 \edef\reserved@b##1{%
8377 \def\expandafter\noexpand
8378 \csname#2\string#1\endcsname####1{%
8379 \noexpand\@text@composite
8380 \expandafter\noexpand\csname#2\string#1\endcsname
8381 ####1\noexpand\empty\noexpand\@text@composite
8382 {##1}%
8383 }%
8384 }%
8385 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8386 \fi
8387 \expandafter\def\csname\expandafter\string\csname
8388 #2\endcsname\string#1-\string#3\endcsname{#4}
8389 \else
8390 \errhelp{Your command will be ignored, type <return> to proceed}%
8391 \errmessage{\string\DeclareTextCompositeCommand\space used on
8392 inappropriate command \protect#1}
8393 \fi
8394 }

```

```

8395 \def\@text@composite#1#2#3\@text@composite{%
8396   \expandafter\@text@composite@x
8397     \csname\string#1-\string#2\endcsname
8398 }
8399 \def\@text@composite@x#1#2{%
8400   \ifx#1\relax
8401     #2%
8402   \else
8403     #1%
8404   \fi
8405 }
8406 %
8407 \def\@strip@args#1:#2-#3\@strip@args{#2}
8408 \def\DeclareTextComposite#1#2#3#4{%
8409   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8410   \bgroup
8411     \lccode\@=#4%
8412     \lowercase{%
8413   \egroup
8414     \reserved@a @%
8415   }%
8416 }
8417 %
8418 \def\UseTextSymbol#1#2{#2}
8419 \def\UseTextAccent#1#2#3{}
8420 \def\@use@text@encoding#1{}
8421 \def\DeclareTextSymbolDefault#1#2{%
8422   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8423 }
8424 \def\DeclareTextAccentDefault#1#2{%
8425   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8426 }
8427 \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.

```

8428 \DeclareTextAccent{"}{OT1}{127}
8429 \DeclareTextAccent{'}{OT1}{19}
8430 \DeclareTextAccent{^}{OT1}{94}
8431 \DeclareTextAccent{\`}{OT1}{18}
8432 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8433 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8434 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8435 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8436 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8437 \DeclareTextSymbol{\i}{OT1}{16}
8438 \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`.

```

8439 \ifx\scriptsize\undefined
8440   \let\scriptsize\sevenrm
8441 \fi

```

And a few more “dummy” definitions.

```

8442 \def\language{english}%
8443 \let\bbl@opt@shorthands\@nnil
8444 \def\bbl@ifshorthand#1#2#3#2{%
8445   \let\bbl@language@opts\@empty
8446   \ifx\babeloptionstrings\undefined
8447     \let\bbl@opt@strings\@nnil
8448   \else

```

```

8449 \let\bbl@opt@strings\babeloptionstrings
8450 \fi
8451 \def\BabelStringsDefault{generic}
8452 \def\bbl@tempa{normal}
8453 \ifx\babeloptionmath\bbl@tempa
8454 \def\bbl@mathnormal{\noexpand\textormath}
8455 \fi
8456 \def\AfterBabelLanguage#1#2{}
8457 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
8458 \let\bbl@afterlang\relax
8459 \def\bbl@opt@safe{BR}
8460 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
8461 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
8462 \expandafter\newif\csname ifbbl@single\endcsname
8463 \chardef\bbl@bidimode\z@
8464 <</Emulate LaTeX>>

A proxy file:

8465 <*plain>
8466 \input babel.def
8467 </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 \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] 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 \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomititis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).