

Babel

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Version 3.84.2967
2022/12/30

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	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	48
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	49
1.31	Current and future work	50
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	51
3	The interface between the core of babel and the language definition files	52
3.1	Guidelines for contributed languages	53
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	56
3.6	Support for extending macros	56
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	57
3.9	Executing code based on the selector	60
II	Source code	61
4	Identification and loading of required files	61
5	locale directory	61

6	Tools	62
6.1	Multiple languages	66
6.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	66
6.3	<code>base</code>	68
6.4	<code>key=value</code> options and other general option	68
6.5	Conditional loading of shorthands	70
6.6	Interlude for Plain	71
7	Multiple languages	71
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	96
7.7	Support for saving macro definitions	98
7.8	Short tags	99
7.9	Hyphens	100
7.10	Multiencoding strings	101
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 bahavior	134
8.1	Cross referencing macros	136
8.2	Marks	139
8.3	Preventing clashes with other packages	140
8.3.1	<code>ifthen</code>	140
8.3.2	<code>varioref</code>	141
8.3.3	<code>hhline</code>	141
8.4	Encoding and fonts	142
8.5	Basic bidi support	143
8.6	Local Language Configuration	146
8.7	Language options	147
9	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	150
10	Loading hyphenation patterns	150
11	Font handling with <code>fontspec</code>	154
12	Hooks for XeTeX and LuaTeX	158
12.1	XeTeX	158
12.2	Layout	160
12.3	8-bit TeX	161
12.4	LuaTeX	162
12.5	Southeast Asian scripts	168
12.6	CJK line breaking	169
12.7	Arabic justification	171
12.8	Common stuff	175
12.9	Automatic fonts and ids switching	175
12.10	Bidi	180
12.11	Layout	182

12.12	Lua: transforms	188
12.13	Lua: Auto bidi with basic and basic-r	196
13	Data for CJK	207
14	The ‘nil’ language	207
15	Calendars	208
15.1	Islamic	209
16	Hebrew	210
17	Persian	214
18	Coptic and Ethiopic	215
19	Buddhist	215
20	Support for Plain T_EX (plain.def)	216
20.1	Not renaming hyphen.tex	216
20.2	Emulating some L ^A T _E X features	217
20.3	General tools	217
20.4	Encoding related macros	221
21	Acknowledgements	223

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 xetex and luatex, the following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and `\today` in Danish and Vietnamese. No additional packages are required, 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 `ruussian`, captions and dates are just strings, and do not switch the language. That means you should set it explicitly if you want to use them, or hyphenation (and in some cases the text itself) will be wrong. For example:

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

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

```
\babelensure{polish}
```

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

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

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

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 `*{⟨char⟩}`

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*{⟨char⟩}` is provided, which makes sure shorthands are always activated.

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

\defineshorthand `[⟨language⟩,⟨language⟩,...]{⟨shorthand⟩}{⟨code⟩}`

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

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

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

EXAMPLE Let’s assume you want a unified set of shorthand for discretionaries (languages do not define shorthands consistently, and `"-`, `\-`, `"=` have different meanings). You 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 $\langle option-name \rangle \{ \langle code \rangle \}$

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

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

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

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

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

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

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

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{სამშარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამშარეულო 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 Niqud 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
klb	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 ^{ul}	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
sichuany
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.polutonic]{babel}
```

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

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

Remember there is an alternative syntax for the latter:

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

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

script= \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= `<counter-name>`

Same for `\Alph`.

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

onchar= `ids | fonts | 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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 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= `<base> <shrink> <stretch>`

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= `<penalty>`

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= `<transform-list>`

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 avaiable 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, syllabe, 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. Çîleya Pêşîn 2019, but with variant=izafa it prints 31'ê Çîleya Pêşînê 2019.

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

New 3.76 Although calendars aren't the primary concern of babel, the package should be able to, at least, generate correctly the current date in the way users would expect in their own culture. Currently, \localedate can print dates in a few calendars (provided the ini locale file has been imported), but year, month and day had to be entered by hand, which is very inconvenient. With this macro, the current date is converted and stored in the three last arguments, which must be macros. Allowed calendars are

buddhist	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` `[\langle language \rangle, \langle language \rangle, ...]{\langle exceptions \rangle}`

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 transform does it. To prevent the conversion (an abbreviation, for example), write "s.
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: !?;.
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 ([*íú*]), the replacement could be {1|*íú*|*íú*}, which maps *í* to *í*, and *ú* to *ú*, so that the diaeresis is removed.

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

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

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

See the [babel site](#) for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account).

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

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

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

See the description above for the optional argument.

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

EXAMPLE You can replace a character (or series of them) by another character (or series of them). Thus, to enter *ž* as *zh* and *š* as *sh* in a newly created locale for transliterated Russian:

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

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

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

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

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

1.22 Selection based on BCP 47 tags

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

It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in *babel*. Instead the data is taken from the *ini* files, which means currently about 250 tags are already recognized. *Babel* performs a simple lookup in the following way: `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 {<text>}
```

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

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For

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.

example, if you load LY1 , LGR, then it is set to LY1, but if you load LY1 , T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}
```

```

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

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

```

\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

    Most Arabic speakers consider the two varieties to be two registers
    of one language, although the two registers can be referred to in
    Arabic as 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 `\textthe` must be defined to select the main language):

```

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

```

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

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

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

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

counters required in all engines (except luatex with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection`).`\section`); required in xetex and pdftex for counters in general, as well as in luatex with `bidi=default`; required in luatex for numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it can depend on the counter format.

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

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 T_EX 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 `\LaTeXe` **New 3.19** .

EXAMPLE Typically, in an Arabic document you would need:

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

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

Digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, xetex). This command is provided to set `{\lr-text}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `r1` counterpart.

Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

`\BabelPatchSection` $\langle section-name \rangle$

Mainly for bidi text, but it 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 T_EX parameters (#1, #2, #3), with the meaning given:

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.27 Languages supported by babel with ldf files

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

Afrikaans afrikaans

Azerbaijani azerbaijani

Basque basque

Breton breton

Bulgarian bulgarian

Catalan catalan

Croatian croatian

Czech czech

Danish danish

Dutch dutch

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

Esperanto esperanto

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

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

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

```

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

```

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

1.28 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).

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

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

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

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

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

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

1.29 Tweaking some features

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

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

<code>bidi.mirroring</code>	<code>linebreak.cjk</code>	<code>layout.lists</code>
<code>bidi.text</code>	<code>justify.arabic</code>	<code>autoload.bcp47</code>
<code>linebreak.sea</code>	<code>layout.tabular</code>	<code>bcp47.toname</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 `hhline` 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 `hhline` (`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 T_EX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of T_EX, not of babel. Alternatively, you may use `\useshorthands` 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 T_EX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²¹

But that is the easy part, because they don't require modifying the L^AT_EX internals.

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

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

²⁰This explains why L^AT_EX 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.

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

“(1)-ból”, but “from (3)” is “(3)-ból”, in Spanish an item labelled “3.^o” may be referred to as either “ítem 3.^o” or “3.^{er} ítem”, and so on.

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with language.dat

T_EX and most engines based on it (pdfT_EX, xetex, ϵ -T_EX, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, L^AT_EX, XeL^AT_EX, pdfL^AT_EX). 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
```

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

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

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

```
=british

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.²⁵ For example:

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain $\text{T}_{\text{E}}\text{X}$ users, so the files have to be coded so that they can be read by both \LaTeX and plain $\text{T}_{\text{E}}\text{X}$. The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\<lang>captions`, `\<lang>date`, `\<lang>extras` and `\<lang>noextras` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\<lang>date` but not `\<lang>captions` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.

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

- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is /).

Some recommendations:

- The preferred shorthand is ", which is not used in \LaTeX (quotes are entered as `` and ' '). Other good choices are characters which are not used in a certain context (eg, = in an ancient language). Note however =, <, >, : and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras<lang>` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\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<lang>`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁶
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

3.1 Guidelines for contributed languages

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

As to ldf files, now language files are “outsourced” and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN).

Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

- 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.

²⁶But not removed, for backward compatibility.

- 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 T_EX sense of set of hyphenation patterns.

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

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

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

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

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

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

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

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

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

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

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

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

\LdfInit The macro `\LdfInit` performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the `@`-sign, preventing the `.ldf` file from being processed twice, etc.

\ldf@quit The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes

resetting the category code of the @-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.

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

`\loadlocalcfg` After processing a language definition file, \TeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to `\captions{lang}` to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by `\ldf@finish`.

`\substitutefontfamily` (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands

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

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


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

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

3.4 Support for active characters

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

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

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

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

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

`\bbl@add@special` The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]

`\bbl@remove@special` It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

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

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

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

3.6 Support for extending macros

`\addto` The macro `\addto{<control sequence>}{<TeX code>}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`.

²⁷This mechanism was introduced by Bernd Raichle.

Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

3.7 Macros common to a number of languages

- `\bbl@allowhyphens` In several languages compound words are used. This means that when \TeX has to hyphenate such a compound word, it only does so at the ‘-’ that is used in such words. To allow hyphenation in the rest of such a compound word, the macro `\bbl@allowhyphens` can be used.
- `\allowhyphens` Same as `\bbl@allowhyphens`, but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with `\accent` in OT1.
Note the previous command (`\bbl@allowhyphens`) has different applications (hyphens and discretionaries) than this one (composite chars). Note also prior to version 3.7, `\allowhyphens` had the behavior of `\bbl@allowhyphens`.
- `\set@low@box` For some languages, quotes need to be lowered to the baseline. For this purpose the macro `\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.
- `\save@sf@q` Sometimes it is necessary to preserve the `\spacefactor`. For this purpose the macro `\save@sf@q` is available. It takes one argument, saves the current `\spacefactor`, executes the argument, and restores the `\spacefactor`.
- `\bbl@frenchspacing` The commands `\bbl@frenchspacing` and `\bbl@nonfrenchspacing` can be used to properly switch French spacing on and off.
- `\bbl@nonfrenchspacing`

3.8 Encoding-dependent strings

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

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

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

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

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

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

Encoding info is `charset=` followed by a `charset`, which if given sets how the strings should be translated to the internal representation used by the engine, typically `utf8`, which is the only value supported currently (default is no translations). Note `charset` is applied by

luatex and xetex when reading the file, not when the macro or string is used in the document.

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

²⁸In future releases further categories may be added.

```

\number\year}

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

\EndBabelCommands

```

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

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

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

\EndBabelCommands Marks the end of the series of blocks.

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

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

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

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

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

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

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

```

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

```

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at $\backslash\text{MakeUppercase}$ and $\backslash\text{MakeLowercase}$. The code would typically be things like $\backslash\text{let}\backslash\text{BB}\backslash\text{bb}$ and $\backslash\text{uccode}$ or $\backslash\text{lccode}$ (although for the reasons explained above, changes in lc/uc codes may not work). A $\langle map-list \rangle$ is a series of macros using the internal format of $\backslash\text{@uc1clist}$ (eg, $\backslash\text{bb}\backslash\text{BB}\backslash\text{cc}\backslash\text{CC}$). The mandatory arguments take precedence over the optional one. This command, unlike $\backslash\text{SetString}$, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \LaTeX , we can set for Turkish:

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

```

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

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

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

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

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

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

- `\BabelLower` $\langle uccode \rangle \langle lccode \rangle$ 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` $\langle uccode\text{-}from \rangle \langle uccode\text{-}to \rangle \langle step \rangle \langle lccode\text{-}from \rangle$ loops though the given uppercase codes, using the `step`, and assigns them the `lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO` $\langle uccode\text{-}from \rangle \langle uccode\text{-}to \rangle \langle step \rangle \langle lccode \rangle$ loops though the given uppercase codes, using the `step`, and assigns them the `lccode`, which is fixed (MO stands for *many-to-one*).

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

```

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

```

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

3.9 Executing code based on the selector

\IfBabelSelectorTF $\langle selectors \rangle \langle true \rangle \langle false \rangle$

New 3.67 Sometimes a different setup is desired depending on the selector used. Values allowed in $\langle selectors \rangle$ 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 an uppercase letter. It can be just a letter (eg, `babel.name.A`, `babel.name.B`) or a name (eg, `date.long.Nominative`, `date.long.Formal`, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a

lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.

6 Tools

```
1 <<version=3.84.2967>>
2 <<date=2022/12/30>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1{#2}}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@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}
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` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³⁰. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```
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{%
```

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

```

34 \begingroup
35 \let\l\lnoexpand
36 \let\l\lbblexp@en
37 \let\l\lbblexp@ue
38 \edef\bblexp@aux{\endgroup#1}%
39 \bblexp@aux}
40 \def\bblexp@en#1>{\expandafter\lnoexpand\csname#1\endcsname}%
41 \def\bblexp@ue#1]{%
42 \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

`\bblexp@trim` The following piece of code is stolen (with some changes) from `keyval`, by David Carlisle. It defines two macros: `\bblexp@trim` and `\bblexp@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\bblexp@tempa#1{%
44 \long\def\bblexp@trim##1##2{%
45 \futurelet\bblexp@trima\bblexp@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46 \def\bblexp@trim@c{%
47 \ifx\bblexp@trima\@sptoken
48 \expandafter\bblexp@trim@b
49 \else
50 \expandafter\bblexp@trim@b\expandafter#1%
51 \fi}%
52 \long\def\bblexp@trim@b#1##1 \@nil{\bblexp@trim@i##1}}
53 \bblexp@tempa{ }
54 \long\def\bblexp@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bblexp@trim@def#1{\bblexp@trim{def#1}}

```

`\bblexp@ifunset` To check if a macro is defined, we create a new macro, which does the same as `\ifundefined`. However, in an ϵ -tex engine, it is based on `\ifcsname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

```

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

```

`\bblexp@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\bblexp@ifblank#1{%
77 \bblexp@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bblexp@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
79 \def\bblexp@ifset#1#2#3{%
80 \bblexp@ifunset{#1}{#3}{\bblexp{\l\lbblexp@ifblank{\@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\bb1@forkv#1#2{%
82   \def\bb1@kvcmd##1##2##3{#2}%
83   \bb1@kvnext#1,\@nil,}
84 \def\bb1@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86     \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
87     \expandafter\bb1@kvnext
88   \fi}
89 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
90   \bb1@trim\def\bb1@forkv@a{#1}%
91   \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@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\bb1@vforeach#1#2{%
93   \def\bb1@forcmd##1{#2}%
94   \bb1@fornext#1,\@nil,}
95 \def\bb1@fornext#1,{%
96   \ifx\@nil#1\relax\else
97     \bb1@ifblank{#1}{\bb1@trim\bb1@forcmd{#1}}%
98     \expandafter\bb1@fornext
99   \fi}
100 \def\bb1@foreach#1{\expandafter\bb1@vforeach\expandafter{#1}}

```

\bb1@replace Returns implicitly \toks@ with the modified string.

```

101 \def\bb1@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}%
103   \def\bb1@replace@aux##1#2##2#2{%
104     \ifx\bb1@nil##2%
105       \toks@\expandafter{\the\toks@##1}%
106     \else
107       \toks@\expandafter{\the\toks@##1#3}%
108       \bb1@afterfi
109       \bb1@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      \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132      \catcode64=\the\catcode64\relax}% Restore @
133      \else
134      \let\bbl@tempc\@empty % Not \relax
135      \fi
136      \bbl@exp{% For the 'uplevel' assignments
137      \endgroup
138      \bbl@tempc}} % empty or expand to set #1 with changes
139 \fi

```

Two further tools. `\bbl@ifsamestring` first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). `\bbl@engine` takes the following values: 0 is 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\bbl@ifsamestring#1#2{%
141   \begingroup
142   \protected@edef\bbl@tempb{#1}%
143   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
144   \protected@edef\bbl@tempc{#2}%
145   \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
146   \ifx\bbl@tempb\bbl@tempc
147     \aftergroup\@firstoftwo
148   \else
149     \aftergroup\@secondoftwo
150   \fi
151 \endgroup}
152 \chardef\bbl@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     \fin@
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\bb1@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182   \toks@\expandafter\expandafter\expandafter{%
183     \csname extras\language\endcsname}%
184   \bb1@exp{\in@{#1}{\the\toks@}}%
185   \ifin@ \else
186     \@temptokena{#2}%
187     \edef\bb1@tempc{\the\@temptokena\the\toks@}%
188     \toks@\expandafter{\bb1@tempc#3}%
189     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
190   \fi}
191 \<</Basic macros>>

```

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

```

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 \TeX < 2. Preserved for compatibility.

```

204 \<<*Define core switching macros>> \equiv
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 (\LaTeX , `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\bb1@trace[1]{\message{^^J[ #1 ]}}%
213   \let\bb1@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\bbl@trace[1]{}%
220 \let\bbl@debug@gobble
221 \ifx\directlua@undefined\else
222   \directlua{ Babel = Babel or {}
223     Babel.debug = false }%
224   \fi}
225 \def\bbl@error#1#2{%
226   \begingroup
227     \def\{\MessageBreak}%
228     \PackageError{babel}{#1}{#2}%
229   \endgroup}
230 \def\bbl@warning#1{%
231   \begingroup
232     \def\{\MessageBreak}%
233     \PackageWarning{babel}{#1}%
234   \endgroup}
235 \def\bbl@infowarn#1{%
236   \begingroup
237     \def\{\MessageBreak}%
238     \PackageNote{babel}{#1}%
239   \endgroup}
240 \def\bbl@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\bbbl@opt@provide\@nnil
364 \let\bbbl@opt@provide\@empty %%% MOVE above
365 \else
366 \chardef\bbbl@iniflag\@ne
367 \bbbl@exp{\bbbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
368 \in{,provide,}{, #1,}%
369 \ifin@
370 \def\bbbl@opt@provide{#2}%
371 \bbbl@replace\bbbl@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 `\bbbl@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

375 \bbbl@trace{Conditional loading of shorthands}
376 \def\bbbl@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\bbbl@sh@string
383 \fi}
384 \ifx\bbbl@opt@shorthands\@nnil
385 \def\bbbl@ifshorthand#1#2#3{#2}%
386 \else\ifx\bbbl@opt@shorthands\@empty
387 \def\bbbl@ifshorthand#1#2#3{#3}%
388 \else

```

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

```

389 \def\bbbl@ifshorthand#1{%
390 \bbbl@xin@{\string#1}{\bbbl@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\bbbl@opt@shorthands{%
397 \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

398 \bbbl@ifshorthand{'}%
399 {\PassOptionsToPackage{activeacute}{babel}}{}
400 \bbbl@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\bbbl@opt@headfoot\@nnil\else
404 \g@addto@macro\@resetactivechars{%
405 \set@typeset@protect
406 \expandafter\select@language@x\expandafter{\bbbl@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}{\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\@#5{%
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\@#5\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@bcp@case#2\@empty\@empty\@@\bbl@tempb
500 \bbl@bcp@case#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\language % Still necessary. TODO
528 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
529 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
530 \ifbbl@bcp@allowed
531 \expandafter\ifx\csname date\language\endcsname\relax
532 \expandafter
533 \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@@
534 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
535 \edef\language{\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\selectlanguage_`. 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=\ccclv\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 `luatex`, 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\locale\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\locale{??}%
627 \else
628 \scantokens\expandafter{\expandafter
629 \def\expandafter\locale\expandafter{\language}}%
630 \fi
631 \fi
632 \else
633 \def\locale{#1}% This one has the correct catcodes
634 \fi
635 \select@language{\language}%
636 % write to aux
637 \expandafter\ifx\csname date\language\endcsname\relax\else
638 \if@filesw
639 \ifx\babel@aux\@gobbles\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@select@name\empty
655 \def\bbl@select@name{select}%
656 % set hmap
657 \fi
658 \ifnum\bbl@hmapset=\@cclv\chardef\bbl@hmapset4\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@savedextras\@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@savedextras\@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 other language 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@hymapsel=\@cclv\let\bbl@hymapsel\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 other language 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@hymapsel=\@cclv\chardef\bbl@hymapsel4\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   \begingroup
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 `otherlanguage*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bbl@switch`.

```
805 \def\foreign@language#1{%
806   % set name
807   \edef\language#1}%
808   \ifbbl@usedatagroup
809     \bbl@add\bbl@select@opts{,date,}%
```



```

810 \bbl@usedategroupfalse
811 \fi
812 \bbl@fixname\language
813 % TODO. name@map here?
814 \bbl@provide@locale
815 \bbl@iflanguage\language{%
816 \let\bbl@select@type\@ne
817 \expandafter\bbl@switch\expandafter{\language}}

```

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@}{% Can be \relax!
841 \begingroup
842 \bbl@xin@{\number\language,}{,\bbl@hyphlist}%
843 \ifin@
844 \else
845 \expandafter\bbl@usehooks{hyphenation}{#{1}}{\bbl@tempa}}%
846 \hyphenation{%
847 \bbl@hyphenation@
848 \@ifundefined{bbl@hyphenation@#1}%
849 \@empty
850 {\space\csname bbl@hyphenation@#1\endcsname}}%
851 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
852 \fi
853 \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, `\lccode'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 \begingroup
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.ldf\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\language\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 $\text{\LaTeX 2}\epsilon$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.
Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

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\endcsname
959               \csname lang@#1\endcsname
960           \fi}%
961         \def\uselanguage#1{%
962           \input language.def
963         \endgroup
964       \fi
965     \fi
966     \chardef\l@english\z@
967 \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.

```

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

```

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

```

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

```

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

```

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

```

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

```

1001 \bbl@trace{Hooks}
1002 \newcommand\AddBabelHook[3][{}]{%
1003   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1004   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1005   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1006   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1007     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1008     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1009   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1010 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1011 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1012 \def\bbl@usehooks#1#2{%
1013   \ifx\UseHook\undefined\else\UseHook{babel/*/#1}\fi
1014   \def\bbl@elth#1{%
1015     \bbl@cs{hk@#1}{\bbl@cs{ev@#1@#1@#2}}%
1016     \bbl@cs{ev@#1@#1}%
1017     \ifx\language\undefined\else % Test required for Plain (?)
1018       \ifx\UseHook\undefined\else\UseHook{babel/\language/#1}\fi
1019       \def\bbl@elth#1{%
1020         \bbl@cs{hk@#1}{\bbl@cl{ev@#1@#1@#2}}%
1021         \bbl@cl{ev@#1@#1}%
1022       }%

```

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).

```

1023 \def\bbl@evargs{,% <- don't delete this comma
1024   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1025   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1026   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1027   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1028   beforestart=0,language=2}
1029 \ifx\NewHook\undefined\else
1030   \def\bbl@tempa#1=#2@@{\NewHook{babel/#1}}
1031   \bbl@foreach\bbl@evargs{\bbl@tempa#1@@}
1032 \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 `\babelensure{<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.

```

1033 \bbl@trace{Defining babelensure}
1034 \newcommand\babelensure[2][{}]{%
1035   \AddBabelHook{babel-ensure}{afterextras}{%
1036     \ifcase\bbl@select@type
1037       \bbl@cl{e}%
1038     \fi}%
1039   \begingroup
1040     \let\bbl@ens@include\@empty
1041     \let\bbl@ens@exclude\@empty
1042     \def\bbl@ens@fontenc{\relax}%
1043     \def\bbl@tempb##1{%
1044       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1045     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1046     \def\bbl@tempb##1=##2\@{\@namedef\bbl@ens@##1}{##2}}%
1047     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1048     \def\bbl@tempc{\bbl@ensure}%
1049     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1050       \expandafter{\bbl@ens@include}}%
1051     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1052       \expandafter{\bbl@ens@exclude}}%
1053     \toks@\expandafter{\bbl@tempc}%
1054     \bbl@exp{%
1055   \endgroup
1056   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1057 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1058   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1059     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1060       \edef##1{\noexpand\bbl@nocaption
1061         {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1062       \fi
1063       \ifx##1\@empty\else
1064         \in{##1}{#2}%
1065         \ifin@ \else
1066           \bbl@ifunset{\bbl@ensure@\language\name}%
1067             {\bbl@exp{%
1068               \\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1069                 \\foreignlanguage{\language\name}%
1070                 {\ifx\relax#3\else
1071                   \\fontencoding{#3}\\selectfont
1072                   \fi
1073                   #####1}}}%
1074             }%
1075           \toks@\expandafter{##1}%
1076           \edef##1{%
1077             \bbl@csarg\noexpand{ensure@\language\name}%
1078             {\the\toks@}}%
1079           \fi
1080           \expandafter\bbl@tempb
1081         \fi}%
1082   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1083   \def\bbl@tempa##1{% elt for include list
1084     \ifx##1\@empty\else
1085       \bbl@csarg\in{ensure@\language\name\expandafter}\expandafter{##1}%
1086       \ifin@ \else
1087         \bbl@tempb##1\@empty
1088       \fi
1089       \expandafter\bbl@tempa
1090     \fi}%
1091   \bbl@tempa#1\@empty}
1092 \def\bbl@captionslist{%
1093   \prefacename\refname\abstractname\bibname\chaptername\appendixname

```

```

1094 \contentsname\listfigurename\listtablename\indexname\figurename
1095 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1096 \alsiname\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`.

```

1097 \bbl@trace{Macros for setting language files up}
1098 \def\bbl@ldfinit{%
1099   \let\bbl@screset\@empty
1100   \let\BabelStrings\bbl@opt@string
1101   \let\BabelOptions\@empty
1102   \let\BabelLanguages\relax
1103   \ifx\originalTeX\undefined
1104     \let\originalTeX\@empty
1105   \else
1106     \originalTeX
1107   \fi}
1108 \def\LdfInit#1#2{%
1109   \chardef\atcatcode=\catcode`\@
1110   \catcode`\@=11\relax
1111   \chardef\eqcatcode=\catcode`\=
1112   \catcode`\==12\relax
1113   \expandafter\if\expandafter\@backslashchar
1114     \expandafter\@car\string#2\@nil
1115     \ifx#2\undefined\else
1116       \ldf@quit{#1}%
1117     \fi
1118   \else
1119     \expandafter\ifx\csname#2\endcsname\relax\else
1120       \ldf@quit{#1}%
1121     \fi
1122   \fi
1123   \bbl@ldfinit}

```

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

```

1124 \def\ldf@quit#1{%
1125   \expandafter\main@language\expandafter{#1}%
1126   \catcode`\@=\atcatcode \let\atcatcode\relax
1127   \catcode`\==\eqcatcode \let\eqcatcode\relax
1128   \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.

```

1129 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1130   \bbl@afterlang
1131   \let\bbl@afterlang\relax
1132   \let\BabelModifiers\relax
1133   \let\bbl@screset\relax}%
1134 \def\ldf@finish#1{%
1135   \loadlocalcfg{#1}%
1136   \bbl@afterldf{#1}%
1137   \expandafter\main@language\expandafter{#1}%
1138   \catcode`\@=\atcatcode \let\atcatcode\relax
1139   \catcode`\=\eqcatcode \let\eqcatcode\relax}

```

After the preamble of the document the commands \LdfInit, \ldf@quit and \ldf@finish are no longer needed. Therefore they are turned into warning messages in L^AT_EX.

```

1140 \@onlypreamble\LdfInit
1141 \@onlypreamble\ldf@quit
1142 \@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.

```

1143 \def\main@language#1{%
1144   \def\bbl@main@language{#1}%
1145   \let\language\name\bbl@main@language % TODO. Set locale name
1146   \bbl@id@assign
1147   \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.

```

1148 \def\bbl@beforestart{%
1149   \def\@nolanerr##1{%
1150     \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1151   \bbl@usehooks{beforestart}{}%
1152   \global\let\bbl@beforestart\relax}
1153 \AtBeginDocument{%
1154   {\@nameuse\bbl@beforestart}}% Group!
1155   \if@filesw
1156     \providecommand\babel@aux[2]{}%
1157     \immediate\write\@mainaux{%
1158       \string\providecommand\string\babel@aux[2]{}%
1159       \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1160   \fi
1161   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1162   \ifbbl@single % must go after the line above.
1163     \renewcommand\selectlanguage[1]{}%
1164     \renewcommand\foreignlanguage[2]{#2}%
1165     \global\let\babel@aux\@gobbletwo % Also as flag
1166   \fi
1167   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1168 \def\select@language@x#1{%
1169   \ifcase\bbl@select@type
1170     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1171   \else
1172     \select@language{#1}%
1173   \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.

```

1174 \bbl@trace{Shorthands}
1175 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1176   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1177   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1178   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1179     \begingroup
1180       \catcode`#1\active
1181       \nfss@catcodes
1182       \ifnum\catcode`#1=\active
1183         \endgroup
1184         \bbl@add\nfss@catcodes{\@makeother#1}%
1185       \else
1186         \endgroup
1187       \fi
1188   \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.

```

1189 \def\bbl@remove@special#1{%
1190   \begingroup
1191   \def\x##1##2{\ifnum`#1=`##2\noexpand\empty
1192     \else\noexpand##1\noexpand##2\fi}%
1193   \def\do{\x\do}%
1194   \def\@makeother{\x\@makeother}%
1195   \edef\x{\endgroup
1196     \def\noexpand\dospecials{\dospecials}%
1197     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1198       \def\noexpand\@sanitize{\@sanitize}%
1199     \fi}%
1200   \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).

```

1201 \def\bbl@active@def#1#2#3#4{%
1202   \@namedef{#3#1}{%
1203     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1204       \bbl@afterelse\bbl@sh@select#2#1{#3#arg#1}{#4#1}%
1205     \else
1206       \bbl@afterfi\csname#2@sh@#1\endcsname

```

```
1207 \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.

```
1208 \long\@namedef{#3@arg#1}##1{%
1209 \expandafter\ifx\csname#2@sh@#1@\string##1@endcsname\relax
1210 \bbl@afterelse\csname#4#1@endcsname##1%
1211 \else
1212 \bbl@afterfi\csname#2@sh@#1@\string##1@endcsname
1213 \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.

```
1214 \def\initiate@active@char#1{%
1215 \bbl@ifunset{active@char\string#1}%
1216 {\bbl@withactive
1217 {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1218 {}}
```

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).

```
1219 \def\@initiate@active@char#1#2#3{%
1220 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1221 \ifx#1\@undefined
1222 \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1223 \else
1224 \bbl@csarg\let{oridef@#2}#1%
1225 \bbl@csarg\edef{oridef@#2}{%
1226 \let\noexpand#1%
1227 \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
1228 \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*").

```
1229 \ifx#1#3\relax
1230 \expandafter\let\csname normal@char#2@endcsname#3%
1231 \else
1232 \bbl@info{Making #2 an active character}%
1233 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1234 \@namedef{normal@char#2}{%
1235 \textormath{#3}\csname bbl@oridef@@#2@endcsname}}%
1236 \else
1237 \@namedef{normal@char#2}{#3}%
1238 \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).

```
1239 \bbl@restoreactive{#2}%
1240 \AtBeginDocument{%
1241 \catcode`#2\active
1242 \if@files@w
1243 \immediate\write\@mainaux{\catcode`\string#2\active}%
1244 \fi}%
1245 \expandafter\bbl@add@special\csname#2@endcsname
1246 \catcode`#2\active
1247 \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⟩`).

```

1248 \let\bbl@tempa\@firstoftwo
1249 \if\string^#2%
1250   \def\bbl@tempa{\noexpand\textormath}%
1251 \else
1252   \ifx\bbl@mathnormal\@undefined\else
1253     \let\bbl@tempa\bbl@mathnormal
1254   \fi
1255 \fi
1256 \expandafter\edef\csname active@char#2\endcsname{%
1257   \bbl@tempa
1258     {\noexpand\if@safe@actives
1259       \noexpand\expandafter
1260         \expandafter\noexpand\csname normal@char#2\endcsname
1261       \noexpand\else
1262         \noexpand\expandafter
1263         \expandafter\noexpand\csname bbl@doactive#2\endcsname
1264       \noexpand\fi}%
1265   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1266 \bbl@csarg\edef{doactive#2}{%
1267   \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!).

```

1268 \bbl@csarg\edef{active#2}{%
1269   \noexpand\active@prefix\noexpand#1%
1270   \expandafter\noexpand\csname active@char#2\endcsname}%
1271 \bbl@csarg\edef{normal#2}{%
1272   \noexpand\active@prefix\noexpand#1%
1273   \expandafter\noexpand\csname normal@char#2\endcsname}%
1274 \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.

```

1275 \bbl@active@def#2\user@group{user@active}{language@active}%
1276 \bbl@active@def#2\language@group{language@active}{system@active}%
1277 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as ‘ ’ ends up in a heading \TeX would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```

1278 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1279   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1280 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1281   {\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.

```

1282 \if\string'#2%
1283   \let\prim@s\bbl@prim@s

```

```

1284 \let\active@math@prime#1%
1285 \fi
1286 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

1287 <<{*More package options}>> ≡
1288 \DeclareOption{math=active}{}
1289 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1290 <</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.

```

1291 \@ifpackagewith{babel}{KeepShorthandsActive}%
1292 {\let\bbl@restoreactive\@gobble}%
1293 {\def\bbl@restoreactive#1{%
1294   \bbl@exp{%
1295     \\\AfterBabelLanguage\\CurrentOption
1296     {\catcode`#1=\the\catcode`#1\relax}%
1297     \\\AtEndOfPackage
1298     {\catcode`#1=\the\catcode`#1\relax}}}%
1299   \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.

```

1300 \def\bbl@sh@select#1#2{%
1301   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1302     \bbl@afterelse\bbl@scndcs
1303   \else
1304     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1305   \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.

```

1306 \begingroup
1307 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1308 {\gdef\active@prefix#1{%
1309   \ifx\protect\@typeset@protect
1310     \else
1311       \ifx\protect\@unexpandable@protect
1312         \noexpand#1%
1313       \else
1314         \protect#1%
1315       \fi
1316       \expandafter\@gobble
1317     \fi}}
1318 {\gdef\active@prefix#1{%
1319   \ifincsname
1320     \string#1%
1321     \expandafter\@gobble
1322   \else
1323     \ifx\protect\@typeset@protect
1324     \else
1325       \ifx\protect\@unexpandable@protect
1326         \noexpand#1%
1327       \else

```

```

1328         \protect#1%
1329         \fi
1330         \expandafter\expandafter\expandafter\@gobble
1331         \fi
1332     \fi}}
1333 \endgroup

```

`\if@safe@actives` In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char⟨char⟩`.

```

1334 \newif\if@safe@actives
1335 \@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.

```

1336 \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`.

```

1337 \chardef\bbl@activated\z@
1338 \def\bbl@activate#1{%
1339     \chardef\bbl@activated\@ne
1340     \bbl@withactive{\expandafter\let\expandafter}#1%
1341     \csname bbl@active@\string#1\endcsname}
1342 \def\bbl@deactivate#1{%
1343     \chardef\bbl@activated\tw@
1344     \bbl@withactive{\expandafter\let\expandafter}#1%
1345     \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1346 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1347 \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.

```

1348 \def\babel@texpdf#1#2#3#4{%
1349     \ifx\texorpdfstring\undefined
1350         \textormath{#1}{#3}%
1351     \else
1352         \texorpdfstring{\textormath{#1}{#3}}{#2}%
1353         % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1354     \fi}
1355 %
1356 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1357 \def\@decl@short#1#2#3\@nil#4{%
1358     \def\bbl@tempa{#3}%
1359     \ifx\bbl@tempa\empty
1360         \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1361         \bbl@ifunset{#1@sh@\string#2@}{}%
1362         {\def\bbl@tempa{#4}%
1363         \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa

```

```

1364     \else
1365     \bbl@info
1366     {Redefining #1 shorthand \string#2\\%
1367     in language \CurrentOption}%
1368     \fi}%
1369     \@namedef{#1@sh@\string#2@}{#4}%
1370 \else
1371 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1372 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1373 {\def\bbl@tempa{#4}%
1374 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1375 \else
1376 \bbl@info
1377 {Redefining #1 shorthand \string#2\string#3\\%
1378 in language \CurrentOption}%
1379 \fi}%
1380 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1381 \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.

```

1382 \def\textormath{%
1383 \ifmmode
1384 \expandafter\@secondoftwo
1385 \else
1386 \expandafter\@firstoftwo
1387 \fi}

```

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

```

1388 \def\user@group{user}
1389 \def\language@group{english} % TODO. I don't like defaults
1390 \def\system@group{system}

```

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

```

1391 \def\usesshorthands{%
1392 \ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1393 \def\bbl@usesesh@s#1{%
1394 \bbl@usesesh@x
1395 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1396 {#1}}
1397 \def\bbl@usesesh@x#1#2{%
1398 \bbl@ifshorthand{#2}%
1399 {\def\user@group{user}%
1400 \initiate@active@char{#2}%
1401 #1%
1402 \bbl@activate{#2}}%
1403 {\bbl@error
1404 {I can't declare a shorthand turned off (\string#2)}
1405 {Sorry, but you can't use shorthands which have been\\%
1406 turned off in the package options}}}

```

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

```

1407 \def\user@language@group{user\language@group}
1408 \def\bbl@set@user@generic#1#2{%

```

```

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

```

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

```

1426 \def\languageshorthands#1{\def\language@group{#1}}

```

`\aliasshorthand` First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with `\aliasshorthands{"}{/}` is `\active@prefix /\active@char/`, so we still need to let the latest to `\active@char`.

```

1427 \def\aliasshorthand#1#2{%
1428 \bbl@ifshorthand{#2}%
1429 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1430 \ifx\document\@notprerr
1431 \@notshorthand{#2}%
1432 \else
1433 \initiate@active@char{#2}%
1434 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1435 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1436 \bbl@activate{#2}%
1437 \fi
1438 \fi}%
1439 {\bbl@error
1440 {Cannot declare a shorthand turned off (\string#2)}
1441 {Sorry, but you cannot use shorthands which have been\\%
1442 turned off in the package options}}}

```

`\@notshorthand`

```

1443 \def\@notshorthand#1{%
1444 \bbl@error{%
1445 The character '\string #1' should be made a shorthand character;\\%
1446 add the command \string\usesshorthands\string{#1\string} to
1447 the preamble.\\%
1448 I will ignore your instruction}%
1449 {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.

```

1450 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1451 \DeclareRobustCommand*\shorthandoff{%
1452 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1453 \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.

```

1454 \def\bb1@switch@sh#1#2{%
1455   \ifx#2\@nnil\else
1456     \bb1@ifunset{\bb1@active@\string#2}%
1457     {\bb1@error
1458       {I can't switch '\string#2' on or off--not a shorthand}%
1459       {This character is not a shorthand. Maybe you made\\
1460        a typing mistake? I will ignore your instruction.}}%
1461     {\ifcase#1    off, on, off*
1462       \catcode`#212\relax
1463     \or
1464       \catcode`#2\active
1465       \bb1@ifunset{\bb1@shdef@\string#2}%
1466       {}%
1467       {\bb1@withactive{\expandafter\let\expandafter}%#2%
1468         \csname bbl@shdef@\string#2\endcsname
1469         \bb1@csarg\let{\shdef@\string#2}\relax}%
1470     \ifcase\bb1@activated\or
1471       \bb1@activate{#2}%
1472     \else
1473       \bb1@deactivate{#2}%
1474     \fi
1475   \or
1476     \bb1@ifunset{\bb1@shdef@\string#2}%
1477     {\bb1@withactive{\bb1@csarg\let{\shdef@\string#2}}#2}%
1478     {}%
1479     \csname bbl@oricat@\string#2\endcsname
1480     \csname bbl@oridef@\string#2\endcsname
1481   \fi}%
1482 \bb1@afterfi\bb1@switch@sh#1%
1483 \fi}

```

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

```

1484 \def\babelshorthand{\active@prefix\babelshorthand\bb1@putsh}
1485 \def\bb1@putsh#1{%
1486   \bb1@ifunset{\bb1@active@\string#1}%
1487   {\bb1@putsh@i#1\@empty\@nnil}%
1488   {\csname bbl@active@\string#1\endcsname}}
1489 \def\bb1@putsh@i#1#2\@nnil{%
1490   \csname\language@group @sh@\string#1@%
1491     \ifx\@empty#2\else\string#2@\fi\endcsname}
1492 \ifx\bb1@opt@shorthands\@nnil\else
1493   \let\bb1@s@initiate@active@char\initiate@active@char
1494   \def\initiate@active@char#1{%
1495     \bb1@ifshorthand{#1}{\bb1@s@initiate@active@char{#1}}{}}
1496   \let\bb1@s@switch@sh\bb1@switch@sh
1497   \def\bb1@switch@sh#1#2{%
1498     \ifx#2\@nnil\else
1499       \bb1@afterfi
1500       \bb1@ifshorthand{#2}{\bb1@s@switch@sh#1{#2}}{\bb1@switch@sh#1}%
1501     \fi}
1502   \let\bb1@s@activate\bb1@activate
1503   \def\bb1@activate#1{%
1504     \bb1@ifshorthand{#1}{\bb1@s@activate{#1}}{}}
1505   \let\bb1@s@deactivate\bb1@deactivate
1506   \def\bb1@deactivate#1{%
1507     \bb1@ifshorthand{#1}{\bb1@s@deactivate{#1}}{}}
1508 \fi

```

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


```
1509 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@string#1}{#3}{#2}}
```

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

```
1510 \def\bbl@prim@s{%
1511   \prime\futurelet\@let@token\bbl@pr@m@s}
1512 \def\bbl@if@primes#1#2{%
1513   \ifx#1\@let@token
1514     \expandafter\@firstoftwo
1515   \else\ifx#2\@let@token
1516     \bbl@afterelse\expandafter\@firstoftwo
1517   \else
1518     \bbl@afterfi\expandafter\@secondoftwo
1519   \fi\fi}
1520 \begingroup
1521   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1522   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1523   \lowercase{%
1524     \gdef\bbl@pr@m@s{%
1525       \bbl@if@primes""%
1526       \pr@@@s
1527       {\bbl@if@primes*\^pr@@@t\egroup}}
1528 \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).

```
1529 \initiate@active@char{~}
1530 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1531 \bbl@activate{~}
```

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

```
1532 \expandafter\def\csname OT1dqpos\endcsname{127}
1533 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1534 \ifx\f@encoding\undefined
1535   \def\f@encoding{OT1}
1536 \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.

```
1537 \bbl@trace{Language attributes}
1538 \newcommand\languageattribute[2]{%
1539   \def\bbl@tempc{#1}%
1540   \bbl@fixname\bbl@tempc
1541   \bbl@iflanguage\bbl@tempc{%
1542     \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.

```

1543 \ifx\bbl@known@attrs\@undefined
1544 \in@false
1545 \else
1546 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1547 \fi
1548 \ifin@
1549 \bbl@warning{%
1550 You have more than once selected the attribute '##1'\%
1551 for language #1. Reported}%
1552 \else

```

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

```

1553 \bbl@exp{%
1554 \\\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1555 \edef\bbl@tempa{\bbl@tempc-##1}%
1556 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1557 {\csname\bbl@tempc @attr@##1\endcsname}%
1558 {\@attrerr{\bbl@tempc}{##1}}%
1559 \fi}}
1560 \onlypreamble\languageattribute

```

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

```

1561 \newcommand*{\@attrerr}[2]{%
1562 \bbl@error
1563 {The attribute #2 is unknown for language #1.}%
1564 {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}`.

```

1565 \def\bbl@declare@ttribute#1#2#3{%
1566 \bbl@xin@{,#2,}{,\BabelModifiers,}%
1567 \ifin@
1568 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1569 \fi
1570 \bbl@add@list\bbl@attributes{#1-#2}%
1571 \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

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

```

1572 \def\bbl@ifattributeset#1#2#3#4{%
1573 \ifx\bbl@known@attrs\@undefined
1574 \in@false
1575 \else
1576 \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1577 \fi
1578 \ifin@
1579 \bbl@afterelse#3%
1580 \else
1581 \bbl@afterfi#4%
1582 \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 \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise.

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```

1583 \def\bbl@ifknown@ttrib#1#2{%
1584   \let\bbl@tempa\@secondoftwo
1585   \bbl@loopx\bbl@tempb{#2}{%
1586     \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1587     \ifin@
1588       \let\bbl@tempa\@firstoftwo
1589     \else
1590       \fi}%
1591   \bbl@tempa}

```

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

```

1592 \def\bbl@clear@ttribs{%
1593   \ifx\bbl@attributes\@undefined\else
1594     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1595       \expandafter\bbl@clear@ttrib\bbl@tempa.
1596     }%
1597     \let\bbl@attributes\@undefined
1598   \fi}
1599 \def\bbl@clear@ttrib#1-#2.{%
1600   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1601 \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`

```

1602 \bbl@trace{Macros for saving definitions}
1603 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1604 \newcount\babel@savecnt
1605 \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.

```

1606 \def\babel@save#1{%
1607   \def\bbl@tempa{, #1,}% Clumsy, for Plain
1608   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1609     \expandafter{\expandafter,\bbl@savedextras,}}%
1610   \expandafter\in\bbl@tempa
1611   \ifin@\else
1612     \bbl@add\bbl@savedextras{, #1,}%
1613     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1614     \toks@\expandafter{\originalTeX\let#1=}%
1615     \bbl@exp{%
1616       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%

```

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

```

1617 \advance\babel@savecnt\@ne
1618 \fi}
1619 \def\babel@savevariable#1{%
1620 \toks@\expandafter{\originalTeX #1}%
1621 \bbl@exp{\def\\originalTeX{\the\toks@\the#1\relax}}}

```

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

```

1622 \def\bbl@frenchspacing{%
1623 \ifnum\the\scode`\.=\@m
1624 \let\bbl@nonfrenchspacing\relax
1625 \else
1626 \frenchspacing
1627 \let\bbl@nonfrenchspacing\nonfrenchspacing
1628 \fi}
1629 \let\bbl@nonfrenchspacing\nonfrenchspacing
1630 \let\bbl@elt\relax
1631 \edef\bbl@fs@chars{%
1632 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1633 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1634 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1635 \def\bbl@pre@fs{%
1636 \def\bbl@elt##1##2##3{\scode`##1=\the\scode`##1\relax}%
1637 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1638 \def\bbl@post@fs{%
1639 \bbl@save@sfcodes
1640 \edef\bbl@tempa{\bbl@c1{frspc}}%
1641 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1642 \if u\bbl@tempa % do nothing
1643 \else\if n\bbl@tempa % non french
1644 \def\bbl@elt##1##2##3{%
1645 \ifnum\scode`##1=##2\relax
1646 \babel@savevariable{\scode`##1}%
1647 \scode`##1=##3\relax
1648 \fi}%
1649 \bbl@fs@chars
1650 \else\if y\bbl@tempa % french
1651 \def\bbl@elt##1##2##3{%
1652 \ifnum\scode`##1=##3\relax
1653 \babel@savevariable{\scode`##1}%
1654 \scode`##1=##2\relax
1655 \fi}%
1656 \bbl@fs@chars
1657 \fi\fi\fi}

```

7.8 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the macros `\text{<tag>}` and `\<tag>`. Definitions are first expanded so that they don't contain `\csname` but the actual macro.

```

1658 \bbl@trace{Short tags}
1659 \def\babeltags#1{%
1660 \edef\bbl@tempa{\zap@space#1 \@empty}%
1661 \def\bbl@tempb##1=##2\@{ }%
1662 \edef\bbl@tempc{%
1663 \noexpand\newcommand
1664 \expandafter\noexpand\csname ##1\endcsname{%
1665 \noexpand\protect
1666 \expandafter\noexpand\csname other language*\endcsname{##2}}

```

```

1667 \noexpand\newcommand
1668 \expandafter\noexpand\csname text##1\endcsname{%
1669 \noexpand\foreignlanguage{##2}}%
1670 \bbl@tempc}%
1671 \bbl@for\bbl@tempa\bbl@tempa{%
1672 \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.

```

1673 \bbl@trace{Hyphens}
1674 \@onlypreamble\babelhyphenation
1675 \AtEndOfPackage{%
1676 \newcommand\babelhyphenation[2][\@empty]{%
1677 \ifx\bbl@hyphenation@relax
1678 \let\bbl@hyphenation@\@empty
1679 \fi
1680 \ifx\bbl@hyphlist\@empty\else
1681 \bbl@warning{%
1682 You must not intermingle \string\selectlanguage\space and\\%
1683 \string\babelhyphenation\space or some exceptions will not\\%
1684 be taken into account. Reported}%
1685 \fi
1686 \ifx\@empty#1%
1687 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1688 \else
1689 \bbl@vforeach{#1}{%
1690 \def\bbl@tempa{##1}%
1691 \bbl@fixname\bbl@tempa
1692 \bbl@iflanguage\bbl@tempa{%
1693 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1694 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1695 }{%
1696 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1697 #2}}}%
1698 \fi}}

```

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

```

1699 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1700 \def\bbl@t@one{T1}
1701 \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`.

```

1702 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1703 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}%
1704 \def\bbl@hyphen{%
1705 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1706 \def\bbl@hyphen@i#1#2{%
1707 \bbl@ifunset{bbl@hy#1#2\@empty}%
1708 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1709 {\csname bbl@hy#1#2\@empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word – the version with a single `@` is used when further hyphenation is allowed, while that with `@@` if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

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

There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

```
1710 \def\bbl@usehyphen#1{%
1711   \leavevmode
1712   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1713   \nobreak\hskip\z@skip}
1714 \def\bbl@@usehyphen#1{%
1715   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1716 \def\bbl@hyphenchar{%
1717   \ifnum\hyphenchar\font=\m@ne
1718     \babe\lnullhyphen
1719   \else
1720     \char\hyphenchar\font
1721   \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.

```
1722 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1723 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1724 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1725 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1726 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1727 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1728 \def\bbl@hy@repeat{%
1729   \bbl@usehyphen{%
1730     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1731 \def\bbl@hy@@repeat{%
1732   \bbl@usehyphen{%
1733     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1734 \def\bbl@hy@empty{\hskip\z@skip}
1735 \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.

```
1736 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}
```

7.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

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

```
1737 \bbl@trace{Multiencoding strings}
1738 \def\bbl@tglobal#1{\global\let#1#1}
```

The second one. We need to patch \@ucllist, 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 \@ucllist 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).

```
1739 \@ifpackagewith{babel}{nocase}%
1740 {\let\bbl@patchuclc\relax}%
```

```

1741 {\def\bbl@patchuclc{%
1742   \global\let\bbl@patchuclc\relax
1743   \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1744   \gdef\bbl@uclc##1{%
1745     \let\bbl@encoded\bbl@encoded@uclc
1746     \bbl@ifunset{\language @bbl@uclc}% and resumes it
1747     {##1}%
1748     {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1749      \csname\language @bbl@uclc\endcsname}%
1750     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1751   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1752   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1753 % A temporary hack, for testing purposes:
1754 \def\BabelRestoreCase{%
1755   \DeclareRobustCommand{\MakeUppercase}[1]{%
1756     \def\reserved@a####1####2{\let####1####2\reserved@a}%
1757     \def\i{I}\def\j{J}%
1758     \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b@gobble}%
1759     \let\UTF@two@octets@noexpand\@empty
1760     \let\UTF@three@octets@noexpand\@empty
1761     \let\UTF@four@octets@noexpand\@empty
1762     \protected@edef\reserved@a{\uppercase{##1}}%
1763     \reserved@a
1764   }}%
1765   \DeclareRobustCommand{\MakeLowercase}[1]{%
1766     \def\reserved@a####1####2{\let####2####1\reserved@a}%
1767     \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b@gobble}%
1768     \let\UTF@two@octets@noexpand\@empty
1769     \let\UTF@three@octets@noexpand\@empty
1770     \let\UTF@four@octets@noexpand\@empty
1771     \protected@edef\reserved@a{\lowercase{##1}}%
1772     \reserved@a}}
1773 <<(*More package options)>> ≡
1774 \DeclareOption{nocase}{}
1775 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1776 <<(*More package options)>> ≡
1777 \let\bbl@opt@strings\@nnil % accept strings=value
1778 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1779 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1780 \def\BabelStringsDefault{generic}
1781 <</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.

```

1782 \@onlypreamble\StartBabelCommands
1783 \def\StartBabelCommands{%
1784   \begingroup
1785   \@tempcnta="7F
1786   \def\bbl@tempa{%
1787     \ifnum\@tempcnta>"FF\else
1788       \catcode\@tempcnta=11
1789       \advance\@tempcnta\@ne
1790       \expandafter\bbl@tempa
1791     \fi}%
1792   \bbl@tempa
1793   <<Macros local to BabelCommands>>
1794   \def\bbl@provstring##1##2{%
1795     \providecommand##1{##2}%
1796     \bbl@tglobal##1}%

```

```

1797 \global\let\bbl@scafter\@empty
1798 \let\StartBabelCommands\bbl@startcmds
1799 \ifx\BabelLanguages\relax
1800   \let\BabelLanguages\CurrentOption
1801 \fi
1802 \begingroup
1803 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1804 \StartBabelCommands}
1805 \def\bbl@startcmds{%
1806   \ifx\bbl@screset\@nnil\else
1807     \bbl@usehooks{stopcommands}{}%
1808   \fi
1809 \endgroup
1810 \begingroup
1811 \@ifstar
1812   {\ifx\bbl@opt@strings\@nnil
1813     \let\bbl@opt@strings\BabelStringsDefault
1814   \fi
1815   \bbl@startcmds@i}%
1816 \bbl@startcmds@i}
1817 \def\bbl@startcmds@i#1#2{%
1818   \edef\bbl@L{\zap@space#1 \@empty}%
1819   \edef\bbl@G{\zap@space#2 \@empty}%
1820   \bbl@startcmds@ii}
1821 \let\bbl@startcmds\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.

```

1822 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1823   \let\SetString\gobbletwo
1824   \let\bbl@stringdef\gobbletwo
1825   \let\AfterBabelCommands\gobble
1826   \ifx\@empty#1%
1827     \def\bbl@sc@label{generic}%
1828     \def\bbl@encstring##1##2{%
1829       \ProvideTextCommandDefault##1{##2}%
1830       \bbl@tglobal##1%
1831       \expandafter\bbl@tglobal\curname\string?\string##1\endcurname}%
1832     \let\bbl@sctest\in@true
1833   \else
1834     \let\bbl@sc@charset\space % <- zapped below
1835     \let\bbl@sc@fontenc\space % <- " "
1836     \def\bbl@tempa##1=##2\@nil{%
1837       \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%
1838     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1839     \def\bbl@tempa##1 ##2{% space -> comma
1840       ##1%
1841       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1842     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1843     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1844     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1845     \def\bbl@encstring##1##2{%
1846       \bbl@foreach\bbl@sc@fontenc{%
1847         \bbl@ifunset{T@####1}%
1848         {}%
1849         {\ProvideTextCommand##1{####1}{##2}%

```



```

1850         \bbl@toglobal##1%
1851         \expandafter
1852         \bbl@toglobal\csname###1\string##1\endcsname}}}%
1853     \def\bbl@sctest{%
1854         \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1855     \fi
1856     \ifx\bbl@opt@strings\@nnil          % ie, no strings key -> defaults
1857     \else\ifx\bbl@opt@strings\relax      % ie, strings=encoded
1858         \let\AfterBabelCommands\bbl@aftercmds
1859         \let\SetString\bbl@setstring
1860         \let\bbl@stringdef\bbl@encstring
1861     \else          % ie, strings=value
1862         \bbl@sctest
1863     \ifin@
1864         \let\AfterBabelCommands\bbl@aftercmds
1865         \let\SetString\bbl@setstring
1866         \let\bbl@stringdef\bbl@provstring
1867     \fi\fi\fi
1868     \bbl@scswitch
1869     \ifx\bbl@G\@empty
1870         \def\SetString##1##2{%
1871             \bbl@error{Missing group for string \string##1}%
1872             {You must assign strings to some category, typically\\%
1873             captions or extras, but you set none}}}%
1874     \fi
1875     \ifx\@empty#1%
1876         \bbl@usehooks{defaultcommands}}}%
1877     \else
1878         \@expandtwoargs
1879         \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
1880     \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group` *(language)* 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` *(language)* 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).

```

1881 \def\bbl@forlang#1#2{%
1882     \bbl@for#1\bbl@L{%
1883         \bbl@xin@{,#1,}{,\BabelLanguages,}%
1884         \ifin#2\relax\fi}}
1885 \def\bbl@scswitch{%
1886     \bbl@forlang\bbl@tempa{%
1887         \ifx\bbl@G\@empty\else
1888             \if\SetString\gobbletwo\else
1889                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1890                 \bbl@xin@{\bbl@GL,}{,\bbl@screset,}%
1891             \ifin@ \else
1892                 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1893                 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1894             \fi
1895         \fi
1896     \fi}}
1897 \AtEndOfPackage{%
1898     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1899     \let\bbl@scswitch\relax}
1900 \@onlypreamble\EndBabelCommands
1901 \def\EndBabelCommands{%
1902     \bbl@usehooks{stopcommands}}}%
1903 \endgroup

```

```

1904 \endgroup
1905 \bbl@scafter}
1906 \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.

```

1907 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1908 \bbl@forlang\bbl@tempa{%
1909 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1910 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1911 {\bbl@exp{%
1912 \global\bbl@add\<\bbl@G\bbl@tempa>{\bbl@scset\#1\<\bbl@LC>}}}%
1913 }%
1914 \def\BabelString{#2}%
1915 \bbl@usehooks{stringprocess}{}%
1916 \expandafter\bbl@stringdef
1917 \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.

```

1918 \ifx\bbl@opt@strings\relax
1919 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1920 \bbl@patchuclc
1921 \let\bbl@encoded\relax
1922 \def\bbl@encoded@uclc#1{%
1923 \inmathwarn#1%
1924 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1925 \expandafter\ifx\csname ?\string#1\endcsname\relax
1926 \TextSymbolUnavailable#1%
1927 \else
1928 \csname ?\string#1\endcsname
1929 \fi
1930 \else
1931 \csname\cf@encoding\string#1\endcsname
1932 \fi}
1933 \else
1934 \def\bbl@scset#1#2{\def#1{#2}}
1935 \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.

```

1936 <{*Macros local to BabelCommands}> \equiv
1937 \def\SetStringLoop##1##2{%
1938 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1939 \count@\z@
1940 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1941 \advance\count@\@ne
1942 \toks@\expandafter{\bbl@tempa}%
1943 \bbl@exp{%
1944 \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1945 \count@=\the\count@\relax}}}%
1946 <{/Macros local to BabelCommands}>

```

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

```

1947 \def\bbl@aftercmds#1{%
1948 \toks@\expandafter{\bbl@scafter#1}%
1949 \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.

```

1950 <<{*Macros local to BabelCommands}>> ≡
1951   \newcommand\SetCase[3][{}]{%
1952     \bbl@patchuclc
1953     \bbl@forlang\bbl@tempa{%
1954       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1955       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1956       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1957 <</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.

```

1958 <<{*Macros local to BabelCommands}>> ≡
1959   \newcommand\SetHyphenMap[1]{%
1960     \bbl@forlang\bbl@tempa{%
1961       \expandafter\bbl@stringdef
1962       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1963 <</Macros local to BabelCommands>>

```

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

```

1964 \newcommand\BabelLower[2]{% one to one.
1965   \ifnum\lccode#1=#2\else
1966     \babel@savevariable{\lccode#1}%
1967     \lccode#1=#2\relax
1968   \fi}
1969 \newcommand\BabelLowerMM[4]{% many-to-many
1970   \@tempcnta=#1\relax
1971   \@tempcntb=#4\relax
1972   \def\bbl@tempa{%
1973     \ifnum\@tempcnta>#2\else
1974       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1975       \advance\@tempcnta#3\relax
1976       \advance\@tempcntb#3\relax
1977       \expandafter\bbl@tempa
1978     \fi}%
1979   \bbl@tempa}
1980 \newcommand\BabelLowerMO[4]{% many-to-one
1981   \@tempcnta=#1\relax
1982   \def\bbl@tempa{%
1983     \ifnum\@tempcnta>#2\else
1984       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1985       \advance\@tempcnta#3
1986       \expandafter\bbl@tempa
1987     \fi}%
1988   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1989 <<{*More package options}>> ≡
1990 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1991 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1992 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1993 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1994 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1995 <</More package options>>

```

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

```

1996 \AtEndOfPackage{%
1997   \ifx\bbl@opt@hyphenmap\undefined
1998     \bbl@xin@{,}{\bbl@language@opts}%
1999     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2000   \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.

```

2001 \newcommand\setlocalecaption{% TODO. Catch typos.
2002   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
2003 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
2004   \bbbl@trim@def\bbbl@tempa{#2}%
2005   \bbbl@xin@{.template}\bbbl@tempa}%
2006   \ifin@
2007     \bbbl@ini@captions@template{#3}{#1}%
2008   \else
2009     \edef\bbbl@tempd{%
2010       \expandafter\expandafter\expandafter
2011       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2012     \bbbl@xin@
2013       {\expandafter\string\csname #2name\endcsname}%
2014       {\bbbl@tempd}%
2015     \ifin@ % Renew caption
2016       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
2017     \ifin@
2018       \bbbl@exp{%
2019         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2020         {\bbbl@scset\<#2name>\<#1#2name>}%
2021         {}}%
2022       \else % Old way converts to new way
2023         \bbbl@ifunset{#1#2name}%
2024         {\bbbl@exp{%
2025           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2026           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2027           {\def\<#2name>\<#1#2name>}}%
2028         {}}%
2029       {}%
2030     \fi
2031   \else
2032     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
2033     \ifin@ % New way
2034       \bbbl@exp{%
2035         \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
2036         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2037         {\bbbl@scset\<#2name>\<#1#2name>}%
2038         {}}%
2039       \else % Old way, but defined in the new way
2040         \bbbl@exp{%
2041           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2042           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2043           {\def\<#2name>\<#1#2name>}}%
2044         {}}%
2045       \fi%
2046     \fi
2047     \@namedef{#1#2name}{#3}%
2048     \toks@\expandafter{\bbbl@captionslist}%
2049     \bbbl@exp{\\in@\<#2name>\the\toks@}%
2050     \ifin@ \else
2051       \bbbl@exp{\\bbbl@add\\bbbl@captionslist{\<#2name>}}%
2052       \bbbl@tglobal\bbbl@captionslist
2053     \fi
2054   \fi}
2055 % \def\bbbl@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.

```
2056 \bbl@trace{Macros related to glyphs}
2057 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z@\hbox{#1}%
2058   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2059   \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.

```
2060 \def\save@sf@q#1{\leavevmode
2061   \begingroup
2062   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2063   \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.

```
2064 \ProvideTextCommand{\quotedblbase}{OT1}{%
2065   \save@sf@q{\set@low@box{\textquotedblright\}%
2066   \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.

```
2067 \ProvideTextCommandDefault{\quotedblbase}{%
2068   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2069 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2070   \save@sf@q{\set@low@box{\textquoteright\}%
2071   \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.

```
2072 \ProvideTextCommandDefault{\quotesinglbase}{%
2073   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2074 \ProvideTextCommand{\guillemetleft}{OT1}{%
2075   \ifmmode
2076     \ll
2077   \else
2078     \save@sf@q{\nobreak
2079       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2080     \fi}
2081 \ProvideTextCommand{\guillemetright}{OT1}{%
2082   \ifmmode
2083     \gg
2084   \else
2085     \save@sf@q{\nobreak
2086       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2087     \fi}
2088 \ProvideTextCommand{\guillemotleft}{OT1}{%
2089   \ifmmode
2090     \ll
2091   \else
```

```

2092 \save@sf@q{\nobreak
2093 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2094 \fi}
2095 \ProvideTextCommand{\guillemotright}{OT1}{%
2096 \ifmmode
2097 \gg
2098 \else
2099 \save@sf@q{\nobreak
2100 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2101 \fi}

```

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

```

2102 \ProvideTextCommandDefault{\guillemetleft}{%
2103 \UseTextSymbol{OT1}{\guillemetleft}}
2104 \ProvideTextCommandDefault{\guillemetright}{%
2105 \UseTextSymbol{OT1}{\guillemetright}}
2106 \ProvideTextCommandDefault{\guillemotleft}{%
2107 \UseTextSymbol{OT1}{\guillemotleft}}
2108 \ProvideTextCommandDefault{\guillemotright}{%
2109 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2110 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2111 \ifmmode
2112 <%
2113 \else
2114 \save@sf@q{\nobreak
2115 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2116 \fi}
2117 \ProvideTextCommand{\guilsinglright}{OT1}{%
2118 \ifmmode
2119 >%
2120 \else
2121 \save@sf@q{\nobreak
2122 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2123 \fi}

```

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

```

2124 \ProvideTextCommandDefault{\guilsinglleft}{%
2125 \UseTextSymbol{OT1}{\guilsinglleft}}
2126 \ProvideTextCommandDefault{\guilsinglright}{%
2127 \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.

```

2128 \DeclareTextCommand{\ij}{OT1}{%
2129 i\kern-0.02em\bbl@allowhyphens j}
2130 \DeclareTextCommand{\IJ}{OT1}{%
2131 I\kern-0.02em\bbl@allowhyphens J}
2132 \DeclareTextCommand{\ij}{T1}{\char188}
2133 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2134 \ProvideTextCommandDefault{\ij}{%
2135 \UseTextSymbol{OT1}{\ij}}
2136 \ProvideTextCommandDefault{\IJ}{%
2137 \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

2138 \def\crrtic@{\hrule height0.1ex width0.3em}
2139 \def\crttic@{\hrule height0.1ex width0.33em}
2140 \def\ddj@{%
2141   \setbox0\hbox{d}\dimen@=\ht0
2142   \advance\dimen@1ex
2143   \dimen@.45\dimen@
2144   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2145   \advance\dimen@ii.5ex
2146   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2147 \def\DDJ@{%
2148   \setbox0\hbox{D}\dimen@=.55\ht0
2149   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2150   \advance\dimen@ii.15ex % correction for the dash position
2151   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2152   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2153   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2154 %
2155 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2156 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2157 \ProvideTextCommandDefault{\dj}{%
2158   \UseTextSymbol{OT1}{\dj}}
2159 \ProvideTextCommandDefault{\DJ}{%
2160   \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.

```

2161 \DeclareTextCommand{\SS}{OT1}{SS}
2162 \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 2163 \ProvideTextCommandDefault{\glq}{%
2164   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2165 \ProvideTextCommand{\grq}{T1}{%
2166   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2167 \ProvideTextCommand{\grq}{TU}{%
2168   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2169 \ProvideTextCommand{\grq}{OT1}{%
2170   \save@sf@q{\kern-.0125em
2171     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2172     \kern.07em\relax}}
2173 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 2174 \ProvideTextCommandDefault{\glqq}{%
2175   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2176 \ProvideTextCommand{\grqq}{T1}{%
2177   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2178 \ProvideTextCommand{\grqq}{TU}{%
2179   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}

```

```

2180 \ProvideTextCommand{\grqq}{OT1}{%
2181   \save@sf@q{\kern-.07em
2182     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2183     \kern.07em\relax}}
2184 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

\flq The ‘french’ single guillemets.

```

\frq
2185 \ProvideTextCommandDefault{\flq}{%
2186   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2187 \ProvideTextCommandDefault{\frq}{%
2188   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq The ‘french’ double guillemets.

```

\frqq
2189 \ProvideTextCommandDefault{\flqq}{%
2190   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2191 \ProvideTextCommandDefault{\frqq}{%
2192   \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).

```

2193 \def\umlauthigh{%
2194   \def\bbl@umlauta##1{\leavevmode\bgroup%
2195     \accent\csname\fontencoding dqpos\endcsname
2196     ##1\bbl@allowhyphens\egroup}%
2197   \let\bbl@umlaute\bbl@umlauta}
2198 \def\umlautlow{%
2199   \def\bbl@umlauta{\protect\lower@umlaut}}
2200 \def\umlautelow{%
2201   \def\bbl@umlaute{\protect\lower@umlaut}}
2202 \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.

```

2203 \expandafter\ifx\csname U@D\endcsname\relax
2204   \csname newdimen\endcsname\U@D
2205 \fi

```

The following code fools \TeX ’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.

```

2206 \def\lower@umlaut#1{%
2207   \leavevmode\bgroup
2208   \U@D 1ex%
2209   {\setbox\z@\hbox{%
2210     \char\csname\fontencoding dqpos\endcsname}%
2211     \dimen@ -.45ex\advance\dimen@\ht\z@
2212     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2213   \accent\csname\fontencoding dqpos\endcsname
2214   \fontdimen5\font\U@D #1%
2215   \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).

```

2216 \AtBeginDocument{%
2217   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2218   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2219   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2220   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{\i}}%
2221   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2222   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2223   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2224   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2225   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2226   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2227   \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.

```

2228 \ifx\l@english\@undefined
2229   \chardef\l@english\z@
2230 \fi
2231 % The following is used to cancel rules in ini files (see Amharic).
2232 \ifx\l@unhyphenated\@undefined
2233   \newlanguage\l@unhyphenated
2234 \fi

```

7.13 Layout

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

```

2235 \bbl@trace{Bidi layout}
2236 \providecommand\IfBabelLayout[3]{#3}%
2237 \newcommand\BabelPatchSection[1]{%
2238   \@ifundefined{#1}{}{%
2239     \bbl@exp{\let<bbl@ss@#1>\<#1>}%
2240     \@namedef{#1}{%
2241       \ifstar{\bbl@presec@#1}%
2242       {\@dblarg{\bbl@presec@x{#1}}}}%
2243 \def\bbl@presec@x#1[#2]#3{%
2244   \bbl@exp{%
2245     \\\select@language@x{\bbl@main@language}%
2246     \\\bbl@cs{sspre@#1}%
2247     \\\bbl@cs{ss@#1}%
2248     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2249     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2250     \\\select@language@x{\language}}%
2251 \def\bbl@presec@s#1#2{%
2252   \bbl@exp{%
2253     \\\select@language@x{\bbl@main@language}%
2254     \\\bbl@cs{sspre@#1}%
2255     \\\bbl@cs{ss@#1}*%
2256     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2257     \\\select@language@x{\language}}%
2258 \IfBabelLayout{sectioning}%
2259   {\BabelPatchSection{part}%
2260    \BabelPatchSection{chapter}%
2261    \BabelPatchSection{section}%
2262    \BabelPatchSection{subsection}%
2263    \BabelPatchSection{subsubsection}%
2264    \BabelPatchSection{paragraph}%

```

```

2265 \BabelPatchSection{subparagraph}%
2266 \def\babel@toc#1{%
2267 \select@language{x{\bbl@main@language}}{}
2268 \IfBabelLayout{captions}%
2269 {\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.

```

2270 \bbl@trace{Input engine specific macros}
2271 \ifcase\bbl@engine
2272 \input txtbabel.def
2273 \or
2274 \input luababel.def
2275 \or
2276 \input xebabel.def
2277 \fi
2278 \providecommand\babelfont{%
2279 \bbl@error
2280 {This macro is available only in LuaLaTeX and XeLaTeX.}%
2281 {Consider switching to these engines.}}
2282 \providecommand\babelprehyphenation{%
2283 \bbl@error
2284 {This macro is available only in LuaLaTeX.}%
2285 {Consider switching to that engine.}}
2286 \ifx\babelposthyphenation\@undefined
2287 \let\babelposthyphenation\babelprehyphenation
2288 \let\babelpatterns\babelprehyphenation
2289 \let\babelcharproperty\babelprehyphenation
2290 \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.

```

2291 \bbl@trace{Creating languages and reading ini files}
2292 \let\bbl@extend@ini\@gobble
2293 \newcommand\babelprovide[2][]{%
2294 \let\bbl@savelangname\language
2295 \edef\bbl@savlocaleid{\the\localeid}%
2296 % Set name and locale id
2297 \edef\language{#2}%
2298 \bbl@id@assign
2299 % Initialize keys
2300 \bbl@foreach{captions,date,import,main,script,language,%
2301 hyphenrules,linebreaking,justification,mapfont,maparabic,%
2302 mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2303 Alph,labels,labels*,calendar,date}%
2304 {\bbl@csarg\let{KVP@##1}\@nnil}%
2305 \global\let\bbl@release@transforms\@empty
2306 \let\bbl@calendars\@empty
2307 \global\let\bbl@inidata\@empty
2308 \global\let\bbl@extend@ini\@gobble
2309 \gdef\bbl@key@list{;}%
2310 \bbl@forkv{#1}{%
2311 \in@{/}{##1}%
2312 \ifin@
2313 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2314 \bbl@renewinikey##1\@{##2}%
2315 \else

```

```

2316 \bbl@csarg\ifx{KVP@##1}\@nnil\else
2317 \bbl@error
2318 {Unknown key '##1' in \string\babelprovide}%
2319 {See the manual for valid keys}%
2320 \fi
2321 \bbl@csarg\def{KVP@##1}{##2}%
2322 \fi}%
2323 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2324 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2325 % == init ==
2326 \ifx\bbl@screset\@undefined
2327 \bbl@ldfinit
2328 \fi
2329 % == date (as option) ==
2330 % \ifx\bbl@KVP@date\@nnil\else
2331 % \fi
2332 % ==
2333 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2334 \ifcase\bbl@howloaded
2335 \let\bbl@lbkflag\@empty % new
2336 \else
2337 \ifx\bbl@KVP@hyphenrules\@nnil\else
2338 \let\bbl@lbkflag\@empty
2339 \fi
2340 \ifx\bbl@KVP@import\@nnil\else
2341 \let\bbl@lbkflag\@empty
2342 \fi
2343 \fi
2344 % == import, captions ==
2345 \ifx\bbl@KVP@import\@nnil\else
2346 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2347 {\ifx\bbl@initoload\relax
2348 \begingroup
2349 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2350 \bbl@input@texini{#2}%
2351 \endgroup
2352 \else
2353 \xdef\bbl@KVP@import{\bbl@initoload}%
2354 \fi}%
2355 {}%
2356 \let\bbl@KVP@date\@empty
2357 \fi
2358 \ifx\bbl@KVP@captions\@nnil
2359 \let\bbl@KVP@captions\bbl@KVP@import
2360 \fi
2361 % ==
2362 \ifx\bbl@KVP@transforms\@nnil\else
2363 \bbl@replace\bbl@KVP@transforms{ },}%
2364 \fi
2365 % == Load ini ==
2366 \ifcase\bbl@howloaded
2367 \bbl@provide@new{#2}%
2368 \else
2369 \bbl@ifblank{#1}%
2370 {}% With \bbl@load@basic below
2371 {\bbl@provide@renew{#2}}%
2372 \fi
2373 % Post tasks
2374 % -----
2375 % == subsequent calls after the first provide for a locale ==
2376 \ifx\bbl@inidata\@empty\else
2377 \bbl@extend@ini{#2}%
2378 \fi

```

```

2379 % == ensure captions ==
2380 \ifx\bbbl@KVP@captions\@nnil\else
2381   \bbbl@ifunset{bbbl@extracaps@#2}%
2382     {\bbbl@exp{\bbabelensure[exclude=\\today]{#2}}}%
2383     {\bbbl@exp{\bbabelensure[exclude=\\today,
2384       include=\[bbbl@extracaps@#2]]{#2}}}%
2385   \bbbl@ifunset{bbbl@ensure@language}%
2386     {\bbbl@exp{%
2387       \\DeclareRobustCommand\<bbbl@ensure@language>[1]{%
2388         \\foreignlanguage{language}%
2389         {####1}}}%
2390     }%
2391   \bbbl@exp{%
2392     \\bbbl@tglobal\<bbbl@ensure@language>%
2393     \\bbbl@tglobal\<bbbl@ensure@language\space>%
2394   \fi
2395   % ==
2396   % At this point all parameters are defined if 'import'. Now we
2397   % execute some code depending on them. But what about if nothing was
2398   % imported? We just set the basic parameters, but still loading the
2399   % whole ini file.
2400   \bbbl@load@basic{#2}%
2401   % == script, language ==
2402   % Override the values from ini or defines them
2403   \ifx\bbbl@KVP@script\@nnil\else
2404     \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2405   \fi
2406   \ifx\bbbl@KVP@language\@nnil\else
2407     \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2408   \fi
2409   \ifcase\bbbl@engine\or
2410     \bbbl@ifunset{bbbl@chrng@language}{}%
2411     {\directlua{
2412       Babel.set_chranges_b('\bbbl@cl{sbc}', '\bbbl@cl{chrng}') }}%
2413   \fi
2414   % == onchar ==
2415   \ifx\bbbl@KVP@onchar\@nnil\else
2416     \bbbl@luahyphenate
2417     \bbbl@exp{%
2418       \\AddToHook{env/document/before}{\select@language{#2}}}%
2419     \directlua{
2420       if Babel.locale_mapped == nil then
2421         Babel.locale_mapped = true
2422         Babel.linebreaking.add_before(Babel.locale_map)
2423         Babel.loc_to_scr = {}
2424         Babel.chr_to_loc = Babel.chr_to_loc or {}
2425       end
2426       Babel.locale_props[\the\localeid].letters = false
2427     }%
2428     \bbbl@xin@{ letters }{ \bbbl@KVP@onchar\space}%
2429     \ifin@
2430       \directlua{
2431         Babel.locale_props[\the\localeid].letters = true
2432       }%
2433     \fi
2434     \bbbl@xin@{ ids }{ \bbbl@KVP@onchar\space}%
2435     \ifin@
2436       \ifx\bbbl@starthyphens\undefined % Needed if no explicit selection
2437         \AddBabelHook{babel-onchar}{beforestart}{\bbbl@starthyphens}%
2438       \fi
2439       \bbbl@exp{\bbbl@add\bbbl@starthyphens
2440         {\bbbl@patterns@lua{language}}}%
2441       % TODO - error/warning if no script

```

```

2442 \directlua{
2443   if Babel.script_blocks['\bbl@cl{sbc}'] then
2444     Babel.loc_to_scr[\the\localeid] =
2445       Babel.script_blocks['\bbl@cl{sbc}']
2446     Babel.locale_props[\the\localeid].lc = \the\localeid\space
2447     Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@\language}\space
2448   end
2449 }%
2450 \fi
2451 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2452 \ifin@
2453   \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys@\language}}{}%
2454   \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}{}%
2455   \directlua{
2456     if Babel.script_blocks['\bbl@cl{sbc}'] then
2457       Babel.loc_to_scr[\the\localeid] =
2458         Babel.script_blocks['\bbl@cl{sbc}']
2459     end}%
2460   \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2461     \AtBeginDocument{%
2462       \bbl@patchfont{\bbl@mapselect}%
2463       {\selectfont}}%
2464     \def\bbl@mapselect{%
2465       \let\bbl@mapselect\relax
2466       \edef\bbl@prefontid{\fontid\font}}%
2467     \def\bbl@mapdir##1{%
2468       {\def\language{##1}%
2469       \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2470       \bbl@switchfont
2471       \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2472         \directlua{
2473           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2474             [\bbl@prefontid] = \fontid\font\space}%
2475         \fi}}%
2476     \fi
2477     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2478   \fi
2479   % TODO - catch non-valid values
2480 \fi
2481 % == mapfont ==
2482 % For bidi texts, to switch the font based on direction
2483 \ifx\bbl@KVP@mapfont\@nnil\else
2484   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2485   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\the
2486     mapfont. Use 'direction'.%
2487     {See the manual for details.}}}%
2488   \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys@\language}}{}%
2489   \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}{}%
2490   \ifx\bbl@mapselect\undefined % TODO. See onchar.
2491     \AtBeginDocument{%
2492       \bbl@patchfont{\bbl@mapselect}%
2493       {\selectfont}}%
2494     \def\bbl@mapselect{%
2495       \let\bbl@mapselect\relax
2496       \edef\bbl@prefontid{\fontid\font}}%
2497     \def\bbl@mapdir##1{%
2498       {\def\language{##1}%
2499       \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2500       \bbl@switchfont
2501       \directlua{Babel.fontmap
2502         [\the\csname bbl@wdir@##1\endcsname]%
2503         [\bbl@prefontid]=\fontid\font}}}%
2504   \fi

```

```

2505 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
2506 \fi
2507 % == Line breaking: intraspace, intrapenalty ==
2508 % For CJK, East Asian, Southeast Asian, if interspace in ini
2509 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2510 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2511 \fi
2512 \bbl@provide@intraspace
2513 % == Line breaking: CJK quotes ==
2514 \ifcase\bbl@engine\or
2515 \bbl@xin@{/c}{\bbl@cl{lnbrk}}%
2516 \ifin@
2517 \bbl@ifunset{bbl@quote@\languagename}{}%
2518 {directlua{
2519 Babel.locale_props[\the\localeid].cjk_quotes = {}
2520 local cs = 'op'
2521 for c in string.utfvalues(
2522 [[\csname bbl@quote@\languagename\endcsname]]) do
2523 if Babel.cjk_characters[c].c == 'qu' then
2524 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2525 end
2526 cs = ( cs == 'op') and 'cl' or 'op'
2527 end
2528 }}%
2529 \fi
2530 \fi
2531 % == Line breaking: justification ==
2532 \ifx\bbl@KVP@justification\@nnil\else
2533 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2534 \fi
2535 \ifx\bbl@KVP@linebreaking\@nnil\else
2536 \bbl@xin@{\bbl@KVP@linebreaking,}%
2537 {,elongated,kashida,cjk,padding,unhyphenated,}%
2538 \ifin@
2539 \bbl@csarg\xdef
2540 {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2541 \fi
2542 \fi
2543 \bbl@xin@{/e}{\bbl@cl{lnbrk}}%
2544 \ifin@\else\bbl@xin@{/k}{\bbl@cl{lnbrk}}\fi
2545 \ifin@\bbl@arabicjust\fi
2546 \bbl@xin@{/p}{\bbl@cl{lnbrk}}%
2547 \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2548 % == Line breaking: hyphenate.other.(locale|script) ==
2549 \ifx\bbl@lbkflag\@empty
2550 \bbl@ifunset{bbl@hyotl@\languagename}{}%
2551 {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}}%
2552 \bbl@startcommands*\languagename}%
2553 \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2554 \ifcase\bbl@engine
2555 \ifnum##1<257
2556 \SetHyphenMap{\BabelLower{##1}{##1}}%
2557 \fi
2558 \else
2559 \SetHyphenMap{\BabelLower{##1}{##1}}%
2560 \fi}%
2561 \bbl@endcommands}%
2562 \bbl@ifunset{bbl@hyots@\languagename}{}%
2563 {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}}%
2564 \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2565 \ifcase\bbl@engine
2566 \ifnum##1<257
2567 \global\lccode##1=##1\relax

```

```

2568         \fi
2569     \else
2570         \global\lccode##1=##1\relax
2571     \fi}}%
2572 \fi
2573 % == Counters: maparabic ==
2574 % Native digits, if provided in ini (TeX level, xe and lua)
2575 \ifcase\bb@engine\else
2576     \bb@ifunset{\bb@dgnat@\language\name}{}%
2577     {\expandafter\ifx\csname \bb@dgnat@\language\name\endcsname\@empty\else
2578         \expandafter\expandafter\expandafter
2579         \bb@setdigits\csname \bb@dgnat@\language\name\endcsname
2580         \ifx\bb@KVP@maparabic\@nnil\else
2581             \ifx\bb@latinarabic\@undefined
2582                 \expandafter\let\expandafter\@arabic
2583                 \csname \bb@counter@\language\name\endcsname
2584             \else % ie, if layout=counters, which redefines \@arabic
2585                 \expandafter\let\expandafter\bb@latinarabic
2586                 \csname \bb@counter@\language\name\endcsname
2587             \fi
2588         \fi
2589     \fi}%
2590 \fi
2591 % == Counters: mapdigits ==
2592 % > luababel.def
2593 % == Counters: alph, Alph ==
2594 \ifx\bb@KVP@alph\@nnil\else
2595     \bb@exp{%
2596         \\bb@add\<\bb@preextras@\language\name>{%
2597             \\babel@save\\@alph
2598             \let\\@alph\<\bb@cntr@\bb@KVP@alph @\language\name>}}%
2599 \fi
2600 \ifx\bb@KVP@Alph\@nnil\else
2601     \bb@exp{%
2602         \\bb@add\<\bb@preextras@\language\name>{%
2603             \\babel@save\\@Alph
2604             \let\\@Alph\<\bb@cntr@\bb@KVP@Alph @\language\name>}}%
2605 \fi
2606 % == Calendars ==
2607 \ifx\bb@KVP@calendar\@nnil
2608     \edef\bb@KVP@calendar{\bb@cl{calpr}}%
2609 \fi
2610 \def\bb@tempe##1 ##2\@{% % Get first calendar
2611     \def\bb@tempa{##1}}%
2612     \bb@exp{\\bb@tempe\bb@KVP@calendar\space\\@}%
2613 \def\bb@tempe##1.##2.##3\@{%
2614     \def\bb@tempc{##1}%
2615     \def\bb@tempb{##2}}%
2616 \expandafter\bb@tempe\bb@tempa..\@
2617 \bb@csarg\edef{calpr@\language\name}{%
2618     \ifx\bb@tempc\@empty\else
2619         calendar=\bb@tempc
2620     \fi
2621     \ifx\bb@tempb\@empty\else
2622         ,variant=\bb@tempb
2623     \fi}%
2624 % == engine specific extensions ==
2625 % Defined in XXXbabel.def
2626 \bb@provide@extra{#2}%
2627 % == require.babel in ini ==
2628 % To load or reload the babel-*.tex, if require.babel in ini
2629 \ifx\bb@beforestart\relax\else % But not in doc aux or body
2630     \bb@ifunset{\bb@rqtex@\language\name}{}%

```

```

2631     {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
2632       \let\BabelBeforeIni\@gobbletwo
2633       \chardef\atcatcode=\catcode`\@
2634       \catcode`\@=11\relax
2635       \bbl@input@texini{\bbl@cs{rqtex@\language}}}%
2636       \catcode`\@=\atcatcode
2637       \let\atcatcode\relax
2638       \global\bbl@csarg\let{rqtex@\language}\relax
2639     \fi}%
2640 \bbl@foreach\bbl@calendars{%
2641   \bbl@ifunset{\bbl@ca##1}{%
2642     \chardef\atcatcode=\catcode`\@
2643     \catcode`\@=11\relax
2644     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2645     \catcode`\@=\atcatcode
2646     \let\atcatcode\relax}%
2647   }{}%
2648 \fi
2649 % == frenchspacing ==
2650 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2651 \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2652 \ifin@
2653   \bbl@extras@wrap{\bbl@pre@fs}%
2654   {\bbl@pre@fs}%
2655   {\bbl@post@fs}%
2656 \fi
2657 % == transforms ==
2658 % > luababel.def
2659 % == main ==
2660 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2661   \let\language\bbl@savelangname
2662   \chardef\localeid\bbl@savelocaleid\relax
2663 \fi}

```

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

```

2664 \def\bbl@provide@new#1{%
2665   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2666   \@namedef{extras#1}{}%
2667   \@namedef{noextras#1}{}%
2668   \bbl@startcommands*{#1}{captions}%
2669   \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2670     \def\bbl@tempb##1{% elt for \bbl@captionslist
2671       \ifx##1\@empty\else
2672         \bbl@exp{%
2673           \\SetString\\##1{%
2674             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2675         \expandafter\bbl@tempb
2676       \fi}%
2677   \expandafter\bbl@tempb\bbl@captionslist\@empty
2678 \else
2679   \ifx\bbl@initoload\relax
2680     \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2681   \else
2682     \bbl@read@ini{\bbl@initoload}2% % Same
2683   \fi
2684 \fi
2685 \StartBabelCommands*{#1}{date}%
2686 \ifx\bbl@KVP@date\@nnil
2687   \bbl@exp{%
2688     \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2689 \else
2690   \bbl@savetoday

```



```

2691 \bbl@savestate
2692 \fi
2693 \bbl@endcommands
2694 \bbl@load@basic{#1}%
2695 % == hyphenmins == (only if new)
2696 \bbl@exp{%
2697 \gdef\<#1hyphenmins>{%
2698 {\bbl@ifunset\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2699 {\bbl@ifunset\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
2700 % == hyphenrules (also in renew) ==
2701 \bbl@provide@hyphens{#1}%
2702 \ifx\bbl@KVP@main\@nnil\else
2703 \expandafter\main@language\expandafter{#1}%
2704 \fi}
2705 %
2706 \def\bbl@provide@renew#1{%
2707 \ifx\bbl@KVP@captions\@nnil\else
2708 \StartBabelCommands*{#1}{captions}%
2709 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2710 \EndBabelCommands
2711 \fi
2712 \ifx\bbl@KVP@date\@nnil\else
2713 \StartBabelCommands*{#1}{date}%
2714 \bbl@savetoday
2715 \bbl@savestate
2716 \EndBabelCommands
2717 \fi
2718 % == hyphenrules (also in new) ==
2719 \ifx\bbl@lbkflag\@empty
2720 \bbl@provide@hyphens{#1}%
2721 \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.)

```

2722 \def\bbl@load@basic#1{%
2723 \ifcase\bbl@howloaded\or\or
2724 \ifcase\csname bbl@llevel@\language\endcsname
2725 \bbl@csarg\let{lname@\language}\relax
2726 \fi
2727 \fi
2728 \bbl@ifunset\bbl@lname@#1}%
2729 {\def\BabelBeforeIni##1##2{%
2730 \begingroup
2731 \let\bbl@ini@captions@aux\@gobbletwo
2732 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2733 \bbl@read@ini{##1}1%
2734 \ifx\bbl@initoload\relax\endinput\fi
2735 \endgroup}%
2736 \begingroup % boxed, to avoid extra spaces:
2737 \ifx\bbl@initoload\relax
2738 \bbl@input@texini{#1}%
2739 \else
2740 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2741 \fi
2742 \endgroup}%
2743 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

2744 \def\bbl@provide@hyphens#1{%
2745 \let\bbl@tempa\relax
2746 \ifx\bbl@KVP@hyphenrules\@nnil\else
2747 \bbl@replace\bbl@KVP@hyphenrules{ },}%
2748 \bbl@foreach\bbl@KVP@hyphenrules{%

```

```

2749 \ifx\bb1@tempa\relax % if not yet found
2750 \bb1@ifsamestring{##1}{+}%
2751 {\bb1@exp{\addlanguage\<l@##1>}}}%
2752 {}%
2753 \bb1@ifunset{l@##1}%
2754 {}%
2755 {\bb1@exp{\let\bb1@tempa\<l@##1>}}}%
2756 \fi}%
2757 \ifx\bb1@tempa\relax
2758 \bb1@warning{%
2759 Requested 'hyphenrules=' for '\language' not found.\\%
2760 Using the default value. Reported}%
2761 \fi
2762 \fi
2763 \ifx\bb1@tempa\relax % if no opt or no language in opt found
2764 \ifx\bb1@KVP@import\@nnil
2765 \ifx\bb1@initoload\relax\else
2766 \bb1@exp{% and hyphenrules is not empty
2767 \bb1@ifblank{\bb1@cs{hyphr@#1}}}%
2768 {}%
2769 {\let\bb1@tempa\<l@bb1@cl{hyphr}>}}}%
2770 \fi
2771 \else % if importing
2772 \bb1@exp{% and hyphenrules is not empty
2773 \bb1@ifblank{\bb1@cs{hyphr@#1}}}%
2774 {}%
2775 {\let\bb1@tempa\<l@bb1@cl{hyphr}>}}}%
2776 \fi
2777 \fi
2778 \bb1@ifunset{bb1@tempa}% ie, relax or undefined
2779 {\bb1@ifunset{l@#1}% no hyphenrules found - fallback
2780 {\bb1@exp{\adddialect\<l@#1>\language}}}%
2781 {}}% so, l@<lang> is ok - nothing to do
2782 {\bb1@exp{\adddialect\<l@#1>\bb1@tempa}}}% found in opt list or ini

```

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

```

2783 \def\bb1@input@texini#1{%
2784 \bb1@bsphack
2785 \bb1@exp{%
2786 \catcode`\%%=14 \catcode`\==0
2787 \catcode`\={1 \catcode`\}=2
2788 \lowercase{\InputIfFileExists{babel-#1.tex}}}%
2789 \catcode`\==\the\catcode`\% \relax
2790 \catcode`\==\the\catcode`\= \relax
2791 \catcode`\={\the\catcode`\{ \relax
2792 \catcode`\}= \the\catcode`\} \relax}%
2793 \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.

```

2794 \def\bb1@inline#1\bb1@inline{%
2795 \ifnextchar[\bb1@inisect{\ifnextchar;\bb1@iniskip\bb1@inistore}#1\@@}% ]
2796 \def\bb1@inisect[#1]#2\@@{\def\bb1@section{#1}}
2797 \def\bb1@iniskip#1\@@{}% if starts with ;
2798 \def\bb1@inistore#1=#2\@@{}% full (default)
2799 \bb1@trim@def\bb1@tempa{#1}%
2800 \bb1@trim\toks@{#2}%
2801 \bb1@xin@{;\bb1@section/\bb1@tempa;}{\bb1@key@list}%
2802 \ifin@ \else
2803 \bb1@xin@{,identification/include.}%
2804 {,\bb1@section/\bb1@tempa}%
2805 \ifin@ \edef\bb1@required@inis{\the\toks@}\fi
2806 \bb1@exp{%

```

```

2807     \\g@addto@macro\\bbl@inidata{%
2808     \\bbl@elt{\\bbl@section}{\\bbl@tempa}{\\the\\toks@}}}%
2809   \\fi}
2810 \\def\\bbl@inistore@min#1=#2@@{%   minimal (maybe set in \\bbl@read@ini)
2811   \\bbl@trim\\def\\bbl@tempa{#1}%
2812   \\bbl@trim\\toks@{#2}%
2813   \\bbl@xin@{.identification.}{.\\bbl@section.}%
2814   \\ifin@
2815     \\bbl@exp{\\g@addto@macro\\bbl@inidata{%
2816       \\bbl@elt{identification}{\\bbl@tempa}{\\the\\toks@}}}%
2817   \\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.

```

2818 \\def\\bbl@loop@ini{%
2819   \\loop
2820   \\if T\\ifeof\\bbl@readstream F\\fi T\\relax % Trick, because inside \\loop
2821     \\endlinechar\\m@ne
2822     \\read\\bbl@readstream to \\bbl@line
2823     \\endlinechar`^^M
2824     \\ifx\\bbl@line\\empty\\else
2825       \\expandafter\\bbl@iniline\\bbl@line\\bbl@iniline
2826     \\fi
2827   \\repeat}
2828 \\ifx\\bbl@readstream\\undefined
2829   \\csname newread\\endcsname\\bbl@readstream
2830 \\fi
2831 \\def\\bbl@read@ini#1#2{%
2832   \\global\\let\\bbl@extend@ini\\gobble
2833   \\openin\\bbl@readstream=babel-#1.ini
2834   \\ifeof\\bbl@readstream
2835     \\bbl@error
2836     {There is no ini file for the requested language\\%
2837     (#1: \\language). Perhaps you misspelled it or your\\%
2838     installation is not complete.}%
2839     {Fix the name or reinstall babel.}%
2840   \\else
2841     % == Store ini data in \\bbl@inidata ==
2842     \\catcode`\\[=12 \\catcode`\\]=12 \\catcode`\\==12 \\catcode`\\&=12
2843     \\catcode`\\;=12 \\catcode`\\|=12 \\catcode`\\%=14 \\catcode`\\-=12
2844     \\bbl@info{Importing
2845       \\ifcase#2font and identification \\or basic \\fi
2846       data for \\language\\%
2847       from babel-#1.ini. Reported}%
2848     \\ifnum#2=\\z@
2849       \\global\\let\\bbl@inidata\\empty
2850       \\let\\bbl@inistore\\bbl@inistore@min   % Remember it's local
2851     \\fi
2852     \\def\\bbl@section{identification}%
2853     \\let\\bbl@required@inis\\empty
2854     \\bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2855     \\bbl@inistore load.level=#2\\@@
2856     \\bbl@loop@ini
2857     \\ifx\\bbl@required@inis\\empty\\else
2858       \\bbl@replace\\bbl@required@inis{ }{,}%
2859       \\bbl@foreach\\bbl@required@inis{%
2860         \\openin\\bbl@readstream=##1.ini
2861         \\bbl@loop@ini}%
2862     \\fi

```

```

2863 % == Process stored data ==
2864 \bbl@csarg\xdef{lini@language}{#1}%
2865 \bbl@read@ini@aux
2866 % == 'Export' data ==
2867 \bbl@ini@exports{#2}%
2868 \global\bbl@csarg\let{inidata@language}\bbl@inidata
2869 \global\let\bbl@inidata@empty
2870 \bbl@exp{\bbl@add@list\bbl@ini@loaded{language}}%
2871 \bbl@tglobal\bbl@ini@loaded
2872 \fi}
2873 \def\bbl@read@ini@aux{%
2874 \let\bbl@savestrings@empty
2875 \let\bbl@savetoday@empty
2876 \let\bbl@savestate@empty
2877 \def\bbl@elt##1##2##3{%
2878 \def\bbl@section{##1}%
2879 \in@{=date.}{=##1}% Find a better place
2880 \ifin@
2881 \bbl@ifunset{bbl@inikv@##1}%
2882 {\bbl@ini@calendar{##1}}%
2883 {}%
2884 \fi
2885 \in@{=identification/extension.}{=##1/##2}%
2886 \ifin@
2887 \bbl@ini@extension{##2}%
2888 \fi
2889 \bbl@ifunset{bbl@inikv@##1}{}%
2890 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2891 \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.

```

2892 \def\bbl@extend@ini@aux#1{%
2893 \bbl@startcommands*{#1}{captions}%
2894 % Activate captions/... and modify exports
2895 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2896 \setlocalecaption{#1}{##1}{##2}}%
2897 \def\bbl@inikv@captions##1##2{%
2898 \bbl@ini@captions@aux{##1}{##2}}%
2899 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2900 \def\bbl@exportkey##1##2##3{%
2901 \bbl@ifunset{bbl@kv@##2}{}%
2902 {\xdefafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2903 \bbl@exp{\global\let\<bbl@##1@language>\<bbl@kv@##2>}}%
2904 \fi}}%
2905 % As with \bbl@read@ini, but with some changes
2906 \bbl@read@ini@aux
2907 \bbl@ini@exports\tw@
2908 % Update inidata@lang by pretending the ini is read.
2909 \def\bbl@elt##1##2##3{%
2910 \def\bbl@section{##1}%
2911 \bbl@iniline##2=##3\bbl@iniline}%
2912 \csname bbl@inidata@#1\endcsname
2913 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2914 \StartBabelCommands*{#1}{date}% And from the import stuff
2915 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2916 \bbl@savetoday
2917 \bbl@savestate
2918 \bbl@endcommands}

```

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

```

2919 \def\bbl@ini@calendar#1{%
2920 \lowercase{\def\bbl@tempa{=##1=}}%
2921 \bbl@replace\bbl@tempa{=date.gregorian}{}}%

```

```

2922 \bbl@replace\bbl@tempa{=date.}{}%
2923 \in@{.licr=}{#1=}%
2924 \ifin@
2925   \ifcase\bbl@engine
2926     \bbl@replace\bbl@tempa{.licr=}{}%
2927   \else
2928     \let\bbl@tempa\relax
2929   \fi
2930 \fi
2931 \ifx\bbl@tempa\relax\else
2932   \bbl@replace\bbl@tempa{=}{}%
2933   \ifx\bbl@tempa\@empty\else
2934     \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2935   \fi
2936   \bbl@exp{%
2937     \def<\bbl@inikv@#1>####1####2{%
2938       \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2939 \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).

```

2940 \def\bbl@renewinikey#1/#2\@#3{%
2941   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2942   \edef\bbl@tempb{\zap@space #2 \@empty}%    key
2943   \bbl@trim\toks@{#3}%                      value
2944   \bbl@exp{%
2945     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2946     \\g@addto@macro\\bbl@inidata{%
2947       \\\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.

```

2948 \def\bbl@exportkey#1#2#3{%
2949   \bbl@ifunset{\bbl@kv@#2}%
2950   {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2951   {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2952     \bbl@csarg\gdef{#1@\languagename}{#3}%
2953   \else
2954     \bbl@exp{\global\let<\bbl@#1@\languagename>\<\bbl@kv@#2>}%
2955   \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.

```

2956 \def\bbl@iniwarning#1{%
2957   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2958   {\bbl@warning{%
2959     From babel-\bbl@cs{lini@\languagename}.ini:\%
2960     \bbl@cs{@kv@identification.warning#1}\%
2961     Reported }}}
2962 %
2963 \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.

```

2964 \def\bbl@ini@extension#1{%
2965   \def\bbl@tempa{#1}%
2966   \bbl@replace\bbl@tempa{extension.}{}%
2967   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2968   \bbl@ifunset{\bbl@info@#1}%
2969   {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
2970   \bbl@exp{%

```

```

2971      \\g@addto@macro\\bbl@moreinfo{%
2972      \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
2973      {}}
2974 \let\bbl@moreinfo\@empty
2975 %
2976 \def\bbl@ini@exports#1{%
2977   % Identification always exported
2978   \bbl@iniwarning{}%
2979   \ifcase\bbl@engine
2980     \bbl@iniwarning{.pdflatex}%
2981   \or
2982     \bbl@iniwarning{.lua\latex}%
2983   \or
2984     \bbl@iniwarning{.xel\latex}%
2985   \fi%
2986   \bbl@exportkey{llevel}{identification.load.level}{}%
2987   \bbl@exportkey{elname}{identification.name.english}{}%
2988   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2989     {\csname bbl@elname@languagename\endcsname}}%
2990   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2991   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2992   \bbl@exportkey{lotf}{identification.tag.opentype}{DFLT}%
2993   \bbl@exportkey{esname}{identification.script.name}{}%
2994   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2995     {\csname bbl@esname@languagename\endcsname}}%
2996   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2997   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2998   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2999   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
3000   \bbl@moreinfo
3001   % Also maps bcp47 -> languagename
3002   \ifbbl@bcptoname
3003     \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\languagename}%
3004   \fi
3005   % Conditional
3006   \ifnum#1>\z@      % 0 = only info, 1, 2 = basic, (re)new
3007     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3008     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3009     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3010     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3011     \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3012     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3013     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3014     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3015     \bbl@exportkey{intsp}{typography.intraspaces}{}%
3016     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3017     \bbl@exportkey{chrng}{characters.ranges}{}%
3018     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3019     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3020     \ifnum#1=\tw@      % only (re)new
3021       \bbl@exportkey{rqtex}{identification.require.babel}{}%
3022       \bbl@tglobal\bbl@savetoday
3023       \bbl@tglobal\bbl@savestate
3024       \bbl@savestrings
3025     \fi
3026   \fi}

```

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

```

3027 \def\bbl@inikv#1#2{%      key=value
3028   \toks@{#2}%             This hides #'s from ini values
3029   \bbl@csarg\edef{@kv@bbl@section.#1}{\the\toks@}}

```

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

```

3030 \let\bbl@inikv@identification\bbl@inikv

```

```

3031 \let\bbl@inikv@date\bbl@inikv
3032 \let\bbl@inikv@typography\bbl@inikv
3033 \let\bbl@inikv@characters\bbl@inikv
3034 \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 \localenumeral, and another one preserving the trailing .1 for the ‘units’.

```

3035 \def\bbl@inikv@counters#1#2{%
3036   \bbl@ifsamestring{#1}{digits}%
3037   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3038             decimal digits}%
3039    {Use another name.}}%
3040   }%
3041   \def\bbl@tempc{#1}%
3042   \bbl@trim@def{\bbl@tempb*}{#2}%
3043   \in@{.1$}{#1$}%
3044   \ifin@
3045     \bbl@replace\bbl@tempc{.1}{}%
3046     \bbl@csarg\protected@xdef{cnt@#1@\language@}{%
3047       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3048   \fi
3049   \in@{.F.}{#1}%
3050   \ifin@ \else \in@{.S.}{#1} \fi
3051   \ifin@
3052     \bbl@csarg\protected@xdef{cnt@#1@\language@}{\bbl@tempb*}%
3053   \else
3054     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3055     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \ \
3056     \bbl@csarg{\global\expandafter\let}{cnt@#1@\language@}\bbl@tempa
3057   \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.

```

3058 \ifcase\bbl@engine
3059   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3060     \bbl@ini@captions@aux{#1}{#2}}
3061 \else
3062   \def\bbl@inikv@captions#1#2{%
3063     \bbl@ini@captions@aux{#1}{#2}}
3064 \fi

```

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

```

3065 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3066   \bbl@replace\bbl@tempa{.template}{}%
3067   \def\bbl@toreplace{#1}{}%
3068   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3069   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3070   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3071   \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3072   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3073   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3074   \ifin@
3075     \@nameuse{\bbl@patch\bbl@tempa}%
3076     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3077   \fi
3078   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3079   \ifin@
3080     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3081     \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3082       \\\bbl@ifunset{\bbl@tempa fmt@\\\language@}%
3083       {[fnum@\bbl@tempa]}%
3084       {\\\@nameuse{\bbl@tempa fmt@\\\language@}}}}%

```

```

3085 \fi}
3086 \def\bbl@ini@captions@aux#1#2{%
3087 \bbl@trim@def\bbl@tempa{#1}%
3088 \bbl@xin@{.template}{\bbl@tempa}%
3089 \ifin@
3090 \bbl@ini@captions@template{#2}\language\name
3091 \else
3092 \bbl@ifblank{#2}%
3093 {\bbl@exp{%
3094 \toks@{\bbl@nocaption{\bbl@tempa}\language\name\bbl@tempa name}}}%
3095 {\bbl@trim\toks@{#2}}%
3096 \bbl@exp{%
3097 \bbl@add\bbl@savestrings{%
3098 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3099 \toks@\expandafter{\bbl@captionslist}%
3100 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3101 \ifin@else
3102 \bbl@exp{%
3103 \bbl@add\<\bbl@extracaps@\language\name>{\<\bbl@tempa name>}}%
3104 \bbl@tglobal\<\bbl@extracaps@\language\name>}}%
3105 \fi
3106 \fi}

```

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

```

3107 \def\bbl@list@the{%
3108 part,chapter,section,subsection,subsubsection,paragraph,%
3109 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3110 table,page,footnote,mpfootnote,mpfn}
3111 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3112 \bbl@ifunset{\bbl@map@#1\language\name}%
3113 {\@nameuse{#1}}%
3114 {\@nameuse{\bbl@map@#1\language\name}}}
3115 \def\bbl@inikv@labels#1#2{%
3116 \in@{.map}{#1}%
3117 \ifin@
3118 \ifx\bbl@KVP@labels\@nnil\else
3119 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3120 \ifin@
3121 \def\bbl@tempc{#1}%
3122 \bbl@replace\bbl@tempc{.map}{}%
3123 \in@{#2}{arabic,roman,Roman,alph,Alph,fnsymbol,}%
3124 \bbl@exp{%
3125 \gdef\<\bbl@map@\bbl@tempc @\language\name>%
3126 {\ifin@<#2>\else\localecounter{#2}\fi}}%
3127 \bbl@foreach\bbl@list@the{%
3128 \bbl@ifunset{the##1}{%
3129 {\bbl@exp{\let\bbl@tempd\<the##1>}}%
3130 \bbl@exp{%
3131 \bbl@sreplace\<the##1>%
3132 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3133 \bbl@sreplace\<the##1>%
3134 {\<\@empty @\bbl@tempc>\<c@##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3135 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3136 \toks@\expandafter\expandafter\expandafter{%
3137 \csname the##1\endcsname}%
3138 \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%
3139 \fi}}%
3140 \fi
3141 \fi
3142 %
3143 \else
3144 %
3145 % The following code is still under study. You can test it and make

```



```

3146 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3147 % language dependent.
3148 \in@{enumerate.}{#1}%
3149 \ifin@
3150 \def\bbl@tempa{#1}%
3151 \bbl@replace\bbl@tempa{enumerate.}{}%
3152 \def\bbl@toreplace{#2}%
3153 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3154 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3155 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3156 \toks@\expandafter{\bbl@toreplace}%
3157 % TODO. Execute only once:
3158 \bbl@exp{%
3159 \\\bbl@add\<extras\language>{%
3160 \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3161 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3162 \\\bbl@tglobal\<extras\language>}%
3163 \fi
3164 \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.

```

3165 \def\bbl@chapttype{chapter}
3166 \ifx\@makechapterhead\@undefined
3167 \let\bbl@patchchapter\relax
3168 \else\ifx\thechapter\@undefined
3169 \let\bbl@patchchapter\relax
3170 \else\ifx\ps@headings\@undefined
3171 \let\bbl@patchchapter\relax
3172 \else
3173 \def\bbl@patchchapter{%
3174 \global\let\bbl@patchchapter\relax
3175 \gdef\bbl@chfmt{%
3176 \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3177 {\@chapapp\space\thechapter}
3178 {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}%
3179 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3180 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3181 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3182 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3183 \bbl@tglobal\appendix
3184 \bbl@tglobal\ps@headings
3185 \bbl@tglobal\chaptermark
3186 \bbl@tglobal\@makechapterhead}
3187 \let\bbl@patchappendix\bbl@patchchapter
3188 \fi\fi\fi
3189 \ifx\@part\@undefined
3190 \let\bbl@patchpart\relax
3191 \else
3192 \def\bbl@patchpart{%
3193 \global\let\bbl@patchpart\relax
3194 \gdef\bbl@partformat{%
3195 \bbl@ifunset{\bbl@partfmt@\language}%
3196 {\partname\nobreakspace\thepart}
3197 {\@nameuse{\bbl@partfmt@\language}}}%
3198 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3199 \bbl@tglobal\@part}
3200 \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

```

3201 \let\bbl@calendar\@empty

```

```

3202 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3203 \def\bbl@localedate#1#2#3#4{%
3204   \begingroup
3205     \edef\bbl@they{#2}%
3206     \edef\bbl@them{#3}%
3207     \edef\bbl@thed{#4}%
3208     \edef\bbl@tempe{%
3209       \bbl@ifunset{bbl@calpr@\languagename}{\bbl@cl{calpr}},%
3210       #1}%
3211     \bbl@replace\bbl@tempe{ }{}%
3212     \bbl@replace\bbl@tempe{CONVERT}{convert=% Hackish
3213     \bbl@replace\bbl@tempe{convert}{convert=%}
3214     \let\bbl@ld@calendar\@empty
3215     \let\bbl@ld@variant\@empty
3216     \let\bbl@ld@convert\relax
3217     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld@##1}{##2}}%
3218     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3219     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3220     \ifx\bbl@ld@calendar\@empty\else
3221       \ifx\bbl@ld@convert\relax\else
3222         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3223         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3224       \fi
3225     \fi
3226     \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3227     \edef\bbl@calendar{% Used in \month..., too
3228       \bbl@ld@calendar
3229       \ifx\bbl@ld@variant\@empty\else
3230         .\bbl@ld@variant
3231       \fi}%
3232     \bbl@cased
3233     {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3234     \bbl@they\bbl@them\bbl@thed}%
3235   \endgroup}
3236 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3237 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3238   \bbl@trim@def\bbl@tempa{#1.#2}%
3239   \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3240   {\bbl@trim@def\bbl@tempa{#3}%
3241   \bbl@trim\toks@{#5}%
3242   \@temptokena\expandafter{\bbl@savedate}%
3243   \bbl@exp{% Reverse order - in ini last wins
3244     \def\\bbl@savedate{%
3245       \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3246       \the\@temptokena}}%
3247   {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3248   {\lowercase{\def\bbl@tempb{#6}}%
3249   \bbl@trim@def\bbl@toreplace{#5}%
3250   \bbl@TG@@date
3251   \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3252   \ifx\bbl@savetoday\@empty
3253     \bbl@exp{% TODO. Move to a better place.
3254       \\AfterBabelCommands{%
3255         \def\<\languagename date>{\protect\<\languagename date >}%
3256         \\newcommand\<\languagename date >[4][\%
3257         \\bbl@usedategroupttrue
3258         \<bbl@ensure@\languagename>{%
3259           \\localedate[####1]{####2}{####3}{####4}}}%
3260       \def\\bbl@savetoday{%
3261         \\SetString\\today{%
3262           \<\languagename date>[convert]%
3263           {\the\year}{\the\month}{\the\day}}}%
3264     \fi}%

```

```
3265 {}}}
```

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).

```
3266 \let\bbl@calendar\@empty
3267 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3268   \@nameuse{bbl@ca#2}#1\@@}
3269 \newcommand\BabelDateSpace{\nobreakspace}
3270 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3271 \newcommand\BabelDated[1]{\number#1}
3272 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3273 \newcommand\BabelDateM[1]{\number#1}
3274 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3275 \newcommand\BabelDateMMMM[1]{%
3276   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3277 \newcommand\BabelDatey[1]{\number#1}%
3278 \newcommand\BabelDateyy[1]{%
3279   \ifnum#1<10 0\number#1 %
3280   \else\ifnum#1<100 \number#1 %
3281   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3282   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3283   \else
3284     \bbl@error
3285     {Currently two-digit years are restricted to the\
3286      range 0-9999.}%
3287     {There is little you can do. Sorry.}%
3288   \fi\fi\fi\fi}
3289 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3290 \def\bbl@replace@finish@iii#1{%
3291   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3292 \def\bbl@TG@date{%
3293   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3294   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3295   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3296   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3297   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3298   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3299   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3300   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3301   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3302   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3303   \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[####1|}%
3304   \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[####2|}%
3305   \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[####3|}%
3306   \bbl@replace@finish@iii\bbl@toreplace}
3307 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3308 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3309 \let\bbl@release@transforms\@empty
3310 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3311 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3312 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3313   #1[#2]{#3}{#4}{#5}}
3314 \begingroup % A hack. TODO. Don't require an specific order
3315   \catcode`\%=12
3316   \catcode`\&=14
3317   \gdef\bbl@transforms#1#2#3{&%
3318     \directlua{
3319       local str = [==[#2]==]
3320       str = str:gsub('%.%d+%.%d+$', '')
```

```

3321 tex.print([[def\string\babeltempa{]] .. str .. [ ]])
3322 }&%
3323 \bbl@xin@{, \babeltempa,}{, \bbl@KVP@transforms,}&%
3324 \ifin@
3325 \in@{.0$}{#2$}&%
3326 \ifin@
3327 \directlua{&% (\attribute) syntax
3328 local str = string.match([[ \bbl@KVP@transforms]],
3329 '%([^( ]-)%([ ^])-\babeltempa')
3330 if str == nil then
3331 tex.print([[def\string\babeltempb{ ]])
3332 else
3333 tex.print([[def\string\babeltempb{,attribute=]] .. str .. [ ]])
3334 end
3335 }
3336 \toks@{#3}&%
3337 \bbl@exp{&%
3338 \\g@addto@macro\\bbl@release@transforms{&%
3339 \relax &% Closes previous \bbl@transforms@aux
3340 \\bbl@transforms@aux
3341 \\#1{label=\babeltempa\babeltempb}{\language\the\toks@}}&%
3342 \else
3343 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3344 \fi
3345 \fi}
3346 \endgroup

```

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

```

3347 \def\bbl@provide@lsys#1{%
3348 \bbl@ifunset{bbl@lname@#1}%
3349 {\bbl@load@info{#1}}%
3350 {}%
3351 \bbl@csarg\let{lsys@#1}\@empty
3352 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3353 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3354 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3355 \bbl@ifunset{bbl@lname@#1}{%
3356 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3357 \ifcase\bbl@engine\or\or
3358 \bbl@ifunset{bbl@prehc@#1}{%
3359 {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3360 {}%
3361 {\ifx\bbl@xenoxyph\undefined
3362 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3363 \ifx\AtBeginDocument\notprerr
3364 \expandafter\@secondoftwo % to execute right now
3365 \fi
3366 \AtBeginDocument{%
3367 \bbl@patchfont{\bbl@xenoxyph}%
3368 \expandafter\selectlanguage\expandafter{\language}}%
3369 \fi}}%
3370 \fi
3371 \bbl@csarg\bbl@to@global{lsys@#1}}
3372 \def\bbl@xenoxyph@d{%
3373 \bbl@ifset{bbl@prehc@language}%
3374 {\ifnum\hyphenchar\font=\defaultshyphenchar
3375 \iffontchar\font\bbl@cl{prehc}\relax
3376 \hyphenchar\font\bbl@cl{prehc}\relax
3377 \else\iffontchar\font"200B
3378 \hyphenchar\font"200B
3379 \else
3380 \bbl@warning

```

```

3381         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3382         in the current font, and therefore the hyphen\\%
3383         will be printed. Try changing the fontspec's\\%
3384         'HyphenChar' to another value, but be aware\\%
3385         this setting is not safe (see the manual).\\%
3386         Reported}%
3387         \hyphenchar\font\defaultthyphenchar
3388         \fi\fi
3389         \fi}%
3390         {\hyphenchar\font\defaultthyphenchar}}
3391     % \fi}

```

```

3392 \def\bbl@load@info#1{%
3393   \def\BabelBeforeIni##1##2{%
3394     \begingroup
3395       \bbl@read@ini{##1}0%
3396       \endinput           % babel- .tex may contain onlypreamble's
3397       \endgroup}%         boxed, to avoid extra spaces:
3398   {\bbl@input@texini{#1}}}
```

[illegible]

```
3430 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}  
3431 \ifx\\#1% % \\ before, in case #1 is multiletter  
3432 \bbl@exp{%  
3433 \def\\ \bbl@tempa####1{%
```

```

3434 \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3435 \else
3436 \toks@\expandafter{\the\toks@\or #1}%
3437 \expandafter\bb1@buildifcase
3438 \fi}

```

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

```

3439 \newcommand\localecnumeral[2]{\bb1@cs{cntr@#1@\language}\{#2}}
3440 \def\bb1@localecntr#1#2{\localecnumeral{#2}{#1}}
3441 \newcommand\localecounter[2]{%
3442 \expandafter\bb1@localecntr
3443 \expandafter{\number\csname c@#2\endcsname}\{#1}}
3444 \def\bb1@alphnumeral#1#2{%
3445 \expandafter\bb1@alphnumeral@i\number#2 76543210\@@{#1}}
3446 \def\bb1@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3447 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3448 \bb1@alphnumeral@ii{#9}000000#1\or
3449 \bb1@alphnumeral@ii{#9}00000#1#2\or
3450 \bb1@alphnumeral@ii{#9}0000#1#2#3\or
3451 \bb1@alphnumeral@ii{#9}000#1#2#3#4\else
3452 \bb1@alphnum@invalid{>9999}%
3453 \fi}
3454 \def\bb1@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3455 \bb1@ifunset{bb1@cntr@#1.F.\number#5#6#7#8@\language}%
3456 {\bb1@cs{cntr@#1.4@\language}\{#5}
3457 \bb1@cs{cntr@#1.3@\language}\{#6}
3458 \bb1@cs{cntr@#1.2@\language}\{#7}
3459 \bb1@cs{cntr@#1.1@\language}\{#8}
3460 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3461 \bb1@ifunset{bb1@cntr@#1.S.321@\language}\{#}
3462 {\bb1@cs{cntr@#1.S.321@\language}\{#}
3463 \fi}%
3464 {\bb1@cs{cntr@#1.F.\number#5#6#7#8@\language}\{#}
3465 \def\bb1@alphnum@invalid#1{%
3466 \bb1@error{Alphabetic numeral too large (#1)}%
3467 {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.

```

3468 \def\bb1@localeinfo#1#2{%
3469 \bb1@ifunset{bb1@info@#2}\{#1}%
3470 {\bb1@ifunset{bb1@csname bb1@info@#2\endcsname @\language}\{#1}}
3471 {\bb1@cs{csname bb1@info@#2\endcsname @\language}\{#1}}
3472 \newcommand\localeinfo[1]{%
3473 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3474 \bb1@afterelse\bb1@localeinfo}%
3475 \else
3476 \bb1@localeinfo
3477 {\bb1@error{I've found no info for the current locale.\\%
3478 The corresponding ini file has not been loaded\\%
3479 Perhaps it doesn't exist}%
3480 {See the manual for details.}}%
3481 {#1}%
3482 \fi}
3483 % \@namedef{bb1@info@name.locale}\{lname}
3484 \@namedef{bb1@info@tag.ini}\{lini}
3485 \@namedef{bb1@info@name.english}\{elname}
3486 \@namedef{bb1@info@name.opentype}\{lname}
3487 \@namedef{bb1@info@tag.bcp47}\{tbc47}
3488 \@namedef{bb1@info@language.tag.bcp47}\{lbc47}

```

```

3489 \@namedef{bbl@info@tag.opentype}{lotf}
3490 \@namedef{bbl@info@script.name}{esname}
3491 \@namedef{bbl@info@script.name.opentype}{sname}
3492 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3493 \@namedef{bbl@info@script.tag.opentype}{sotf}
3494 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3495 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3496 % Extensions are dealt with in a special way
3497 % Now, an internal \LaTeX{} macro:
3498 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}

```

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

```

3499 <{*More package options}> \equiv
3500 \DeclareOption{ensureinfo=off}{}
3501 <{/More package options}>
3502 %
3503 \let\bbl@ensureinfo\@gobble
3504 \newcommand\BabelEnsureInfo{%
3505   \ifx\InputIfFileExists\undefined\else
3506     \def\bbl@ensureinfo##1{%
3507       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3508   \fi
3509   \bbl@foreach\bbl@loaded{%
3510     \def\language{##1}%
3511     \bbl@ensureinfo{##1}}}%
3512 \@ifpackagewith{babel}{ensureinfo=off}{}%
3513 {\AtEndOfPackage{% Test for plain.
3514   \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.

```

3515 \newcommand\getlocaleproperty{%
3516   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3517 \def\bbl@getproperty@s#1#2#3{%
3518   \let#1\relax
3519   \def\bbl@elt##1##2##3{%
3520     \bbl@ifsamestring{##1/##2}{#3}%
3521     {\providecommand#1{##3}%
3522       \def\bbl@elt####1####2####3{}}}%
3523   {}}%
3524   \bbl@cs{inidata@#2}}%
3525 \def\bbl@getproperty@x#1#2#3{%
3526   \bbl@getproperty@s{#1}{#2}{#3}%
3527   \ifx#1\relax
3528     \bbl@error
3529     {Unknown key for locale '#2':\%
3530      #3\}%
3531     \string#1 will be set to \relax}%
3532     {Perhaps you misspelled it.}%
3533   \fi}
3534 \let\bbl@ini@loaded\@empty
3535 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

8 Adjusting the Babel bahavior

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

```

3536 \newcommand\babeladjust[1]{% TODO. Error handling.
3537   \bbl@forkv{#1}{%
3538     \bbl@ifunset{bbl@ADJ@##1@##2}%
3539     {\bbl@cs{ADJ@##1}{##2}}%
3540     {\bbl@cs{ADJ@##1@##2}}}

```

```

3541 %
3542 \def\bbl@adjust@lua#1#2{%
3543   \ifvmode
3544     \ifnum\currentgrouplevel=\z@
3545       \directlua{ Babel.#2 }%
3546       \expandafter\expandafter\expandafter\@gobble
3547       \fi
3548   \fi
3549   {\bbl@error   % The error is gobbled if everything went ok.
3550     {Currently, #1 related features can be adjusted only\\%
3551       in the main vertical list.}%
3552     {Maybe things change in the future, but this is what it is.}}}
3553 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3554   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3555 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3556   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3557 \@namedef{bbl@ADJ@bidi.text@on}{%
3558   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3559 \@namedef{bbl@ADJ@bidi.text@off}{%
3560   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3561 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3562   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3563 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3564   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3565 %
3566 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3567   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3568 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3569   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3570 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3571   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3572 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3573   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3574 \@namedef{bbl@ADJ@justify.arabic@on}{%
3575   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3576 \@namedef{bbl@ADJ@justify.arabic@off}{%
3577   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3578 %
3579 \def\bbl@adjust@layout#1{%
3580   \ifvmode
3581     #1%
3582     \expandafter\@gobble
3583   \fi
3584   {\bbl@error   % The error is gobbled if everything went ok.
3585     {Currently, layout related features can be adjusted only\\%
3586       in vertical mode.}%
3587     {Maybe things change in the future, but this is what it is.}}}
3588 \@namedef{bbl@ADJ@layout.tabular@on}{%
3589   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3590 \@namedef{bbl@ADJ@layout.tabular@off}{%
3591   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3592 \@namedef{bbl@ADJ@layout.lists@on}{%
3593   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3594 \@namedef{bbl@ADJ@layout.lists@off}{%
3595   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3596 %
3597 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3598   \bbl@bcpallowedtrue}
3599 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3600   \bbl@bcpallowedfalse}
3601 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3602   \def\bbl@bcp@prefix{#1}}
3603 \def\bbl@bcp@prefix{bcp47-}

```



```

3604 \@namedef{bbl@ADJ@autoload.options}#1{%
3605   \def\bbl@autoload@options{#1}}
3606 \let\bbl@autoload@bcptions\@empty
3607 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3608   \def\bbl@autoload@bcptions{#1}}
3609 \newif\ifbbl@bcptoname
3610 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3611   \bbl@bcptonametrue
3612   \BabelEnsureInfo}
3613 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3614   \bbl@bcptonamefalse}
3615 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3616   \directlua{ Babel.ignore_pre_char = function(node)
3617     return (node.lang == \the\csname l@nohyphenation\endcsname)
3618   end }}
3619 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3620   \directlua{ Babel.ignore_pre_char = function(node)
3621     return false
3622   end }}
3623 \@namedef{bbl@ADJ@select.write@shift}{%
3624   \let\bbl@restorelastskip\relax
3625   \def\bbl@savelastskip{%
3626     \let\bbl@restorelastskip\relax
3627     \ifvmode
3628       \ifdim\lastskip=\z@
3629         \let\bbl@restorelastskip\nobreak
3630       \else
3631         \bbl@exp{%
3632           \def\\bbl@restorelastskip{%
3633             \skip@=\the\lastskip
3634             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3635         \fi
3636       \fi}}
3637 \@namedef{bbl@ADJ@select.write@keep}{%
3638   \let\bbl@restorelastskip\relax
3639   \let\bbl@savelastskip\relax}
3640 \@namedef{bbl@ADJ@select.write@omit}{%
3641   \AddBabelHook{babel-select}{beforestart}{%
3642     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3643 \let\bbl@restorelastskip\relax
3644 \def\bbl@savelastskip##1\bbl@restorelastskip{}
3645 \@namedef{bbl@ADJ@select.encoding@off}{%
3646   \let\bbl@encoding@select@off\@empty}

```

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

```

3647 \ifx\directlua\@undefined\else
3648   \ifx\bbl@luapatterns\@undefined
3649     \input luabel.def
3650   \fi
3651 \fi

```

Continue with \LaTeX .

```

3652 </package | core>
3653 <*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.

```

3654 <<{*More package options}>> ≡
3655 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3656 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3657 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3658 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3659 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3660 <</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.

```

3661 \bbl@trace{Cross referencing macros}
3662 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3663   \def\@newl@bel#1#2#3{%
3664     {\@safe@activestrue
3665       \bbl@ifunset{#1@#2}%
3666         \relax
3667         {\gdef\@multiplelabels{%
3668           \@latex@warning@no@line{There were multiply-defined labels}}%
3669           \@latex@warning@no@line{Label `#2' multiply defined}}%
3670         \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.

```

3671 \CheckCommand*\@testdef[3]{%
3672   \def\reserved@a{#3}%
3673   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3674   \else
3675     \@tempswatrue
3676   \fi}

```

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

```

3677 \def\@testdef#1#2#3{% TODO. With @samestring?
3678   \@safe@activestrue
3679   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3680   \def\bbl@tempb{#3}%
3681   \@safe@activesfalse
3682   \ifx\bbl@tempa\relax
3683   \else
3684     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3685     \fi
3686     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3687     \ifx\bbl@tempa\bbl@tempb
3688     \else
3689       \@tempswatrue
3690     \fi}
3691 \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.

```

3692 \bbl@xin@{R}\bbl@opt@safe
3693 \ifin@
3694   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3695   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%

```

```

3696     {\expandafter\strip@prefix\meaning\ref}%
3697 \ifin@
3698 \bbl@redefine\@kernel@ref#1{%
3699     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activfalse}
3700 \bbl@redefine\@kernel@pageref#1{%
3701     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3702 \bbl@redefine\@kernel@sref#1{%
3703     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3704 \bbl@redefine\@kernel@spageref#1{%
3705     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3706 \else
3707 \bbl@redefineroobust\ref#1{%
3708     \@safe@activetrue\org@ref{#1}\@safe@activfalse}
3709 \bbl@redefineroobust\pageref#1{%
3710     \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
3711 \fi
3712 \else
3713 \let\org@ref\ref
3714 \let\org@pageref\pageref
3715 \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.

```

3716 \bbl@xin@{B}\bbl@opt@safe
3717 \ifin@
3718 \bbl@redefine\@citex[#1]#2{%
3719     \@safe@activetrue\edef\@tempa{#2}\@safe@activfalse
3720     \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.

```

3721 \AtBeginDocument{%
3722     \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.)

```

3723     \def\@citex[#1][#2]#3{%
3724         \@safe@activetrue\edef\@tempa{#3}\@safe@activfalse
3725         \org@@citex[#1][#2]{\@tempa}}%
3726     }{}

```

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

```

3727 \AtBeginDocument{%
3728     \ifpackageloaded{cite}{%
3729         \def\@citex[#1]#2{%
3730             \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3731         }{}

```

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

```

3732 \bbl@redefine\nocite#1{%
3733     \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

`\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

`\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.)

```

3774 \ifx\@mkboth\markboth
3775 \def\bbl@tempc{\let\@mkboth\markboth}%
3776 \else
3777 \def\bbl@tempc{}%
3778 \fi
3779 \bbl@ifunset{markboth}{\bbl@redefine\bbl@redefineroobust
3780 \markboth#1#2{%
3781 \protected@edef\bbl@tempb##1{%
3782 \protect\foreignlanguage
3783 {\language}\protect\bbl@restore@actives##1}}%
3784 \bbl@ifblank{#1}%
3785 {\toks@{}}%
3786 {\toks@\expandafter{\bbl@tempb{#1}}}%
3787 \bbl@ifblank{#2}%
3788 {\@temptokena{}}%
3789 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3790 \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}}%
3791 \bbl@tempc
3792 \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.

```

3793 \bbl@trace{Preventing clashes with other packages}
3794 \ifx\org@ref\@undefined\else
3795 \bbl@xin@{R}\bbl@opt@safe
3796 \ifin@
3797 \AtBeginDocument{%
3798 \@ifpackageloaded{ifthen}{%
3799 \bbl@redefine@long\ifthenelse#1#2#3{%
3800 \let\bbl@temp@pref\pageref
3801 \let\pageref\org@pageref
3802 \let\bbl@temp@ref\ref
3803 \let\ref\org@ref
3804 \@safe@activestrue
3805 \org@ifthenelse{#1}%
3806 {\let\pageref\bbl@temp@pref
3807 \let\ref\bbl@temp@ref
3808 \@safe@activesfalse
3809 #2}%

```

```

3810      {\let\pageref\bb1@temp@pref
3811      \let\ref\bb1@temp@ref
3812      \@safe@activesfalse
3813      #3}%
3814      }%
3815      }{}%
3816    }
3817 \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 `\vrefpagemum`.

```

3818 \AtBeginDocument{%
3819   \@ifpackageloaded{varioref}{%
3820     \bbl@redefine\@vpageref#1[#2]#3{%
3821       \@safe@activestrue
3822       \org@@vpageref{#1}[#2]{#3}%
3823       \@safe@activesfalse}%
3824     \bbl@redefine\vrefpagemum#1#2{%
3825       \@safe@activestrue
3826       \org@vrefpagemum{#1}{#2}%
3827       \@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.

```

3828   \expandafter\def\csname Ref\endcsname#1{%
3829     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3830   }{}%
3831 }
3832 \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.

```

3833 \AtEndOfPackage{%
3834   \AtBeginDocument{%
3835     \@ifpackageloaded{hhline}%
3836     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3837       \else
3838         \makeatletter
3839         \def\@currname{hhline}\input{hhline.sty}\makeatother
3840       \fi}%
3841     {}}

```

`\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.

```

3842 \def\substitutefontfamily#1#2#3{%
3843   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3844   \immediate\write15{%
3845     \string\ProvidesFile{#1#2.fd}%
3846     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}]
3847     \space generated font description file]^^J

```

```

3848 \string\DeclareFontFamily{#1}{#2}{^^J
3849 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3850 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3851 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3852 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3853 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3854 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3855 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3856 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3857 }%
3858 \closeout15
3859 }
3860 \@onlypreamble\substitutefontfamily

```

8.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ and $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 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`

```

3861 \bbl@trace{Encoding and fonts}
3862 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3863 \newcommand\BabelNonText{TS1,T3,TS3}
3864 \let\org@TeX\TeX
3865 \let\org@LaTeX\LaTeX
3866 \let\ensureascii\@firstofone
3867 \AtBeginDocument{%
3868   \def\@elt#1{,#1,}%
3869   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3870   \let\@elt\relax
3871   \let\bbl@tempb\@empty
3872   \def\bbl@tempc{OT1}%
3873   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3874     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3875   \bbl@foreach\bbl@tempa{%
3876     \bbl@xin@{#1}{\BabelNonASCII}%
3877     \ifin@
3878       \def\bbl@tempb{#1}% Store last non-ascii
3879     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3880     \ifin@\else
3881       \def\bbl@tempc{#1}% Store last ascii
3882     \fi
3883   \fi}%
3884   \ifx\bbl@tempb\@empty\else
3885     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3886     \ifin@\else
3887       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3888     \fi
3889   \edef\ensureascii#1{%
3890     {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3891   \DeclareTextCommandDefault{\TeX}{\ensureascii\org@TeX}%
3892   \DeclareTextCommandDefault{\LaTeX}{\ensureascii\org@LaTeX}%
3893   \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.

```

3894 \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.

```

3895 \AtBeginDocument{%
3896   \@ifpackageloaded{fontspec}%
3897   {\xdef\latinencoding{%
3898     \ifx\UTFencname\undefined
3899       EU\ifcase\bbl@engine\or2\or1\fi
3900     \else
3901       \UTFencname
3902     \fi}}%
3903   {\gdef\latinencoding{OT1}%
3904     \ifx\cf@encoding\bbl@t@one
3905       \xdef\latinencoding{\bbl@t@one}%
3906     \else
3907       \def\@elt#1{,#1,}%
3908       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3909       \let\@elt\relax
3910       \bbl@xin@{,T1,}\bbl@tempa
3911       \ifin@
3912         \xdef\latinencoding{\bbl@t@one}%
3913       \fi
3914     \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.

```

3915 \DeclareRobustCommand{\latintext}{%
3916   \fontencoding{\latinencoding}\selectfont
3917   \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.

```

3918 \ifx\@undefined\DeclareTextFontCommand
3919   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3920 \else
3921   \DeclareTextFontCommand{\textlatin}{\latintext}
3922 \fi

```

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

```

3923 \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 LuaTeX-jā shows, vertical typesetting is possible, too.

```

3924 \bbl@trace{Loading basic (internal) bidi support}
3925 \ifodd\bbl@engine
3926 \else % TODO. Move to txtbabel
3927   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3928     \bbl@error
3929     {The bidi method 'basic' is available only in\\%
3930      luatex. I'll continue with 'bidi=default', so\\%
3931      expect wrong results}%
3932     {See the manual for further details.}%
3933   \let\bbl@beforeforeign\leavevmode
3934   \AtEndOfPackage{%
3935     \EnableBabelHook{babel-bidi}%
3936     \bbl@xebidipar}
3937   \fi\fi
3938   \def\bbl@loadxebidi#1{%
3939     \ifx\RTLfootnotetext\@undefined
3940       \AtEndOfPackage{%
3941         \EnableBabelHook{babel-bidi}%
3942         \bbl@loadfontspec % bidi needs fontspec
3943         \usepackage#1{bidi}}%
3944     \fi}
3945   \ifnum\bbl@bidimode>200
3946     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3947       \bbl@tentative{bidi=bidi}
3948       \bbl@loadxebidi{}
3949     \or
3950       \bbl@loadxebidi{[rldocument]}
3951     \or
3952       \bbl@loadxebidi{}
3953     \fi
3954   \fi
3955 \fi
3956 % TODO? Separate:
3957 \ifnum\bbl@bidimode=\@ne
3958   \let\bbl@beforeforeign\leavevmode
3959   \ifodd\bbl@engine
3960     \newattribute\bbl@attr@dir
3961     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3962     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3963   \fi
3964   \AtEndOfPackage{%
3965     \EnableBabelHook{babel-bidi}%
3966     \ifodd\bbl@engine\else
3967       \bbl@xebidipar
3968     \fi}
3969 \fi

```

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

```

3970 \bbl@trace{Macros to switch the text direction}
3971 \def\bbl@alscripts{Arabic,Syriac,Thaana,}
3972 \def\bbl@rscripts{% TODO. Base on codes ??
3973   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3974   Old Hungarian,Lydian,Mandaean,Manichaean,%
3975   Meroitic Cursive,Meroitic,Old North Arabian,%
3976   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3977   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3978   Old South Arabian,}%
3979 \def\bbl@provide@dirs#1{%

```

```

3980 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3981 \ifin@
3982 \global\bbl@csarg\chardef{wdir@#1}\@ne
3983 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3984 \ifin@
3985 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3986 \fi
3987 \else
3988 \global\bbl@csarg\chardef{wdir@#1}\z@
3989 \fi
3990 \ifodd\bbl@engine
3991 \bbl@csarg\ifcase{wdir@#1}%
3992 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3993 \or
3994 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3995 \or
3996 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3997 \fi
3998 \fi}
3999 \def\bbl@switchdir{%
4000 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4001 \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4002 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4003 \def\bbl@setdirs#1{% TODO - math
4004 \ifcase\bbl@select@type % TODO - strictly, not the right test
4005 \bbl@bodydir{#1}%
4006 \bbl@pardir{#1}%
4007 \fi
4008 \bbl@textdir{#1}}
4009 % TODO. Only if \bbl@bidimode > 0?:
4010 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4011 \DisableBabelHook{babel-bidi}

```

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

```

4012 \ifodd\bbl@engine % luatex=1
4013 \else % pdftex=0, xetex=2
4014 \newcount\bbl@dirlevel
4015 \chardef\bbl@thetextdir\z@
4016 \chardef\bbl@thepardir\z@
4017 \def\bbl@textdir#1{%
4018 \ifcase#1\relax
4019 \chardef\bbl@thetextdir\z@
4020 \bbl@textdir@i\beginL\endL
4021 \else
4022 \chardef\bbl@thetextdir\@ne
4023 \bbl@textdir@i\beginR\endR
4024 \fi}
4025 \def\bbl@textdir@i#1#2{%
4026 \ifhmode
4027 \ifnum\currentgrouplevel>\z@
4028 \ifnum\currentgrouplevel=\bbl@dirlevel
4029 \bbl@error{Multiple bidi settings inside a group}%
4030 {I'll insert a new group, but expect wrong results.}%
4031 \bgroup\aftergroup#2\aftergroup\egroup
4032 \else
4033 \ifcase\currentgrouptype\or % 0 bottom
4034 \aftergroup#2% 1 simple {}
4035 \or
4036 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4037 \or
4038 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4039 \or\or\or % vbox vtop align
4040 \or

```

```

4041      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4042      \or\or\or\or\or\or % output math disc insert vcent mathchoice
4043      \or
4044      \aftergroup#2% 14 \begingroup
4045      \else
4046      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4047      \fi
4048      \fi
4049      \bbl@dirlevel\currentgrouplevel
4050      \fi
4051      #1%
4052      \fi}
4053      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4054      \let\bbl@bodydir\@gobble
4055      \let\bbl@pagedir\@gobble
4056      \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).

```

4057      \def\bbl@xebidipar{%
4058      \let\bbl@xebidipar\relax
4059      \TeXeTstate\@ne
4060      \def\bbl@xeeverypar{%
4061      \ifcase\bbl@thepardir
4062      \ifcase\bbl@thetextdir\else\beginR\fi
4063      \else
4064      {\setbox\z@\lastbox\beginR\box\z@}%
4065      \fi}%
4066      \let\bbl@severypar\everypar
4067      \newtoks\everypar
4068      \everypar=\bbl@severypar
4069      \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4070      \ifnum\bbl@bidimode>200
4071      \let\bbl@textdir@i\@gobbletwo
4072      \let\bbl@xebidipar\@empty
4073      \AddBabelHook{bidi}{foreign}{%
4074      \def\bbl@tempa{\def\BabelText####1}%
4075      \ifcase\bbl@thetextdir
4076      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4077      \else
4078      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4079      \fi}
4080      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4081      \fi
4082      \fi

```

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

```

4083      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4084      \AtBeginDocument{%
4085      \ifx\pdfstringdefDisableCommands\@undefined\else
4086      \ifx\pdfstringdefDisableCommands\relax\else
4087      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4088      \fi
4089      \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`.

```

4090 \bbl@trace{Local Language Configuration}
4091 \ifx\loadlocalcfg\@undefined
4092   \@ifpackagewith{babel}{noconfigs}%
4093   {\let\loadlocalcfg\@gobble}%
4094   {\def\loadlocalcfg#1{%
4095     \InputIfFileExists{#1.cfg}%
4096     {\typeout{*****^J%
4097               * Local config file #1.cfg used^^J%
4098               *}}}%
4099   \@empty}}
4100 \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).

```

4101 \bbl@trace{Language options}
4102 \let\bbl@afterlang\relax
4103 \let\BabelModifiers\relax
4104 \let\bbl@loaded\@empty
4105 \def\bbl@load@language#1{%
4106   \InputIfFileExists{#1.ldf}%
4107   {\edef\bbl@loaded{\CurrentOption
4108     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4109     \expandafter\let\expandafter\bbl@afterlang
4110       \csname\CurrentOption.ldf-h@@k\endcsname
4111     \expandafter\let\expandafter\BabelModifiers
4112       \csname bbl@mod@\CurrentOption\endcsname}%
4113   {\bbl@error{%
4114     Unknown option '\CurrentOption'. Either you misspelled it\\%
4115     or the language definition file \CurrentOption.ldf was not found}}%
4116     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4117     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4118     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.

```

4119 \def\bbl@try@load@lang#1#2#3{%
4120   \IfFileExists{\CurrentOption.ldf}%
4121   {\bbl@load@language{\CurrentOption}}%
4122   {#1\bbl@load@language{#2}#3}}
4123 %
4124 \DeclareOption{hebrew}{%
4125   \input{rlbabel.def}%
4126   \bbl@load@language{hebrew}}
4127 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4128 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4129 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4130 \DeclareOption{polutonikogreek}{%
4131   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4132 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4133 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4134 \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.

```

4135 \ifx\bbl@opt@config\@nnil
4136   \@ifpackagewith{babel}{noconfigs}{}%
4137   {\InputIfFileExists{bblopts.cfg}%

```

```

4138      {\typeout{*****^J%
4139              * Local config file bblopts.cfg used^^J%
4140              *}}%
4141      {}}%
4142 \else
4143   \InputIfFileExists{\bbl@opt@config.cfg}%
4144   {\typeout{*****^J%
4145           * Local config file \bbl@opt@config.cfg used^^J%
4146           *}}%
4147   {\bbl@error{%
4148     Local config file '\bbl@opt@config.cfg' not found}{%
4149     Perhaps you misspelled it.}}%
4150 \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.

```

4151 \ifx\bbl@opt@main\@nnil
4152   \ifnum\bbl@iniflag>\z@ % if all \ldf's: set implicitly, no main pass
4153     \let\bbl@tempb\@empty
4154     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4155     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4156     \bbl@foreach\bbl@tempb{%      \bbl@tempb is a reversed list
4157       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4158         \ifodd\bbl@iniflag % = *=
4159           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4160         \else % n +=
4161           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4162         \fi
4163       \fi}%
4164   \fi
4165 \else
4166   \bbl@info{Main language set with 'main='. Except if you have\\%
4167     problems, prefer the default mechanism for setting\\%
4168     the main language. Reported}%
4169 \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`).

```

4170 \ifx\bbl@opt@main\@nnil\else
4171   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4172   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4173 \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.

```

4174 \bbl@foreach\bbl@language@opts{%
4175   \def\bbl@tempa{#1}%
4176   \ifx\bbl@tempa\bbl@opt@main\else
4177     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = \ldf)
4178       \bbl@ifunset{ds@#1}%
4179       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4180       {}%
4181     \else % + * (other = ini)
4182       \DeclareOption{#1}{%
4183         \bbl@ldfinit
4184         \babelprovide[import]{#1}%
4185         \bbl@afterldf{}}%
4186     \fi
4187 \fi}

```

```

4188 \bbl@foreach\@classoptionslist{%
4189   \def\bbl@tempa{#1}%
4190   \ifx\bbl@tempa\bbl@opt@main\else
4191     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4192       \bbl@ifunset{ds@#1}%
4193       {\IfFileExists{#1.ldf}%
4194        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4195        {}}%
4196       {}%
4197     \else % + * (other = ini)
4198       \IfFileExists{babel-#1.tex}%
4199       {\DeclareOption{#1}{%
4200        \bbl@ldfinit
4201        \babelprovide[import]{#1}%
4202        \bbl@afterldf{}}}%
4203       {}}%
4204   \fi
4205 \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):

```

4206 \def\AfterBabelLanguage#1{%
4207   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}%
4208   \DeclareOption*{}
4209 \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.

```

4210 \bbl@trace{Option 'main'}
4211 \ifx\bbl@opt@main\@nnil
4212   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4213   \let\bbl@tempc\@empty
4214   \edef\bbl@templ{\bbl@loaded,}
4215   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4216   \bbl@for\bbl@tempb\bbl@tempa{%
4217     \edef\bbl@tempd{\bbl@tempb,}%
4218     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4219     \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4220     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4221   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4222   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4223   \ifx\bbl@tempb\bbl@tempc\else
4224     \bbl@warning{%
4225       Last declared language option is '\bbl@tempc',\%
4226       but the last processed one was '\bbl@tempb'.\%
4227       The main language can't be set as both a global\%
4228       and a package option. Use 'main=\bbl@tempc' as\%
4229       option. Reported}
4230   \fi
4231 \else
4232   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4233     \bbl@ldfinit
4234     \let\CurrentOption\bbl@opt@main
4235     \bbl@exp{% \bbl@opt@provide = empty if *
4236       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4237     \bbl@afterldf{}
4238     \DeclareOption{\bbl@opt@main}{}
4239   \else % case 0,2 (main is ldf)

```

```

4240 \ifx\bbl@loadmain\relax
4241 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4242 \else
4243 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4244 \fi
4245 \ExecuteOptions{\bbl@opt@main}
4246 \@namedef{ds@\bbl@opt@main}{}%
4247 \fi
4248 \DeclareOption*{}
4249 \ProcessOptions*
4250 \fi
4251 \def\AfterBabelLanguage{%
4252 \bbl@error
4253 {Too late for \string\AfterBabelLanguage}%
4254 {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.

```

4255 \ifx\bbl@main@language\undefined
4256 \bbl@info{%
4257 You haven't specified a language as a class or package\\%
4258 option. I'll load 'nil'. Reported}
4259 \bbl@load@language{nil}
4260 \fi
4261 \</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 \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

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

A proxy file for `switch.def`

```

4262 \<*kernel>
4263 \let\bbl@onlyswitch\@empty
4264 \input babel.def
4265 \let\bbl@onlyswitch\@undefined
4266 \</kernel>
4267 \<*patterns>

```

10 Loading hyphenation patterns

The following code is meant to be read by `ini \TeX` because it should instruct \TeX to read hyphenation patterns. To this end the `docstrip` option `patterns` is used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

```

4268 \<Make sure ProvidesFile is defined>
4269 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4270 \xdef\bbl@format{\jobname}
4271 \def\bbl@version{\<<version>>}
4272 \def\bbl@date{\<<date>>}
4273 \ifx\AtBeginDocument\@undefined
4274 \def\@empty{}
4275 \fi
4276 \<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.

```
4277 \def\process@line#1#2 #3 #4 {%
4278   \ifx=#1%
4279     \process@synonym{#2}%
4280   \else
4281     \process@language{#1#2}{#3}{#4}%
4282   \fi
4283   \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.

```
4284 \toks@{}
4285 \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.

```
4286 \def\process@synonym#1{%
4287   \ifnum\last@language=\m@ne
4288     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4289   \else
4290     \expandafter\chardef\csname l@#1\endcsname\last@language
4291     \wlog{\string\l@#1=\string\language\the\last@language}%
4292     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4293       \csname\language\hyphenmins\endcsname
4294     \let\bbl@elt\relax
4295     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4296   \fi}
```

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

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

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

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

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

Some pattern files contain changes to the `\lccode` `\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.

```
4297 \def\process@language#1#2#3{%
4298   \expandafter\addlanguage\csname l@#1\endcsname
```



```

4299 \expandafter\language\csname l@#1\endcsname
4300 \edef\language{#1}%
4301 \bbl@hook@everylanguage{#1}%
4302 % > luatex
4303 \bbl@get@enc#1::\@@@
4304 \begingroup
4305 \lefthyphenmin\m@ne
4306 \bbl@hook@loadpatterns{#2}%
4307 % > luatex
4308 \ifnum\lefthyphenmin=\m@ne
4309 \else
4310 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4311 \the\lefthyphenmin\the\righthyphenmin}%
4312 \fi
4313 \endgroup
4314 \def\bbl@tempa{#3}%
4315 \ifx\bbl@tempa\@empty\else
4316 \bbl@hook@loadexceptions{#3}%
4317 % > luatex
4318 \fi
4319 \let\bbl@elt\relax
4320 \edef\bbl@languages{%
4321 \bbl@languages\bbl@elt{#1}\the\language}{#2}{\bbl@tempa}}%
4322 \ifnum\the\language=\z@
4323 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4324 \set@hyphenmins\tw@\thr@@\relax
4325 \else
4326 \expandafter\expandafter\expandafter\set@hyphenmins
4327 \csname #1hyphenmins\endcsname
4328 \fi
4329 \the\toks@
4330 \toks@{}%
4331 \fi}

```

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

```

4332 \def\bbl@get@enc#1:#2:#3\@@@\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. loadkernel currently loads nothing, but define some basic macros instead.

```

4333 \def\bbl@hook@everylanguage#1{}
4334 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4335 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4336 \def\bbl@hook@loadkernel#1{%
4337 \def\addlanguage{\csname newlanguage\endcsname}%
4338 \def\adddialect##1##2{%
4339 \global\chardef##1##2\relax
4340 \wlog{\string##1 = a dialect from \string\language##2}}%
4341 \def\iflanguage##1{%
4342 \expandafter\ifx\csname l@##1\endcsname\relax
4343 \nolannerr{##1}%
4344 \else
4345 \ifnum\csname l@##1\endcsname=\language
4346 \expandafter\expandafter\expandafter\@firstoftwo
4347 \else
4348 \expandafter\expandafter\expandafter\@secondoftwo
4349 \fi
4350 \fi}%
4351 \def\providehyphenmins##1##2{%
4352 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4353 \@namedef{##1hyphenmins}{##2}%
4354 \fi}%

```

```

4355 \def\set@hyphenmins##1##2{%
4356 \lefthyphenmin##1\relax
4357 \righthyphenmin##2\relax}%
4358 \def\selectlanguage{%
4359 \errhelp{Selecting a language requires a package supporting it}%
4360 \errmessage{Not loaded}}%
4361 \let\foreignlanguage\selectlanguage
4362 \let\otherlanguage\selectlanguage
4363 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4364 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4365 \def\setlocale{%
4366 \errhelp{Find an armchair, sit down and wait}%
4367 \errmessage{Not yet available}}%
4368 \let\uselocale\setlocale
4369 \let\locale\setlocale
4370 \let\selectlocale\setlocale
4371 \let\localename\setlocale
4372 \let\textlocale\setlocale
4373 \let\textlanguage\setlocale
4374 \let\languagetext\setlocale}
4375 \begingroup
4376 \def\AddBabelHook#1#2{%
4377 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4378 \def\next{\toks1}%
4379 \else
4380 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4381 \fi
4382 \next}
4383 \ifx\directlua\undefined
4384 \ifx\XeTeXinputencoding\undefined\else
4385 \input xebabel.def
4386 \fi
4387 \else
4388 \input luababel.def
4389 \fi
4390 \openin1 = babel-\bbl@format.cfg
4391 \ifeof1
4392 \else
4393 \input babel-\bbl@format.cfg\relax
4394 \fi
4395 \closein1
4396 \endgroup
4397 \bbl@hook@loadkernel{switch.def}

```

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

```

4398 \openin1 = language.dat

```

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

```

4399 \def\languagename{english}%
4400 \ifeof1
4401 \message{I couldn't find the file language.dat,\space
4402 I will try the file hyphen.tex}
4403 \input hyphen.tex\relax
4404 \chardef\l@english\z@
4405 \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.

```

4406 \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.

```
4407 \loop
4408 \endlinechar\m@ne
4409 \read1 to \bbl@line
4410 \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.

```
4411 \if T\ifeof1F\fi T\relax
4412 \ifx\bbl@line\@empty\else
4413 \edef\bbl@line{\bbl@line\space\space\space}%
4414 \expandafter\process@line\bbl@line\relax
4415 \fi
4416 \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.

```
4417 \begingroup
4418 \def\bbl@elt#1#2#3#4{%
4419 \global\language=#2\relax
4420 \gdef\language#1}%
4421 \def\bbl@elt##1##2##3##4{}}%
4422 \bbl@languages
4423 \endgroup
4424 \fi
4425 \closein1
```

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

```
4426 \if/\the\toks@\else
4427 \errhelp{language.dat loads no language, only synonyms}
4428 \errmessage{Orphan language synonym}
4429 \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.

```
4430 \let\bbl@line\@undefined
4431 \let\process@line\@undefined
4432 \let\process@synonym\@undefined
4433 \let\process@language\@undefined
4434 \let\bbl@get@enc\@undefined
4435 \let\bbl@hyph@enc\@undefined
4436 \let\bbl@tempa\@undefined
4437 \let\bbl@hook@loadkernel\@undefined
4438 \let\bbl@hook@everylanguage\@undefined
4439 \let\bbl@hook@loadpatterns\@undefined
4440 \let\bbl@hook@loadexceptions\@undefined
4441 \</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].

```
4442 <(*More package options)> ≡
4443 \chardef\bbl@bidimode\z@
4444 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4445 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
```

```

4446 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4447 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4448 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4449 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4450 <</More package options>>

```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to `babel`, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading message, which is replaced by a more explanatory one.

```

4451 <<(*Font selection)>> ≡
4452 \bbl@trace{Font handling with fontspec}
4453 \ifx\ExplSyntaxOn\@undefined\else
4454   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4455     \in@{, #1,}{, no-script, language-not-exist,}%
4456     \ifin\else\bbl@tempfs@nx{#1}{#2}\fi}
4457   \def\bbl@fs@warn@nxx#1#2#3{%
4458     \in@{, #1,}{, no-script, language-not-exist,}%
4459     \ifin\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4460   \def\bbl@loadfontspec{%
4461     \let\bbl@loadfontspec\relax
4462     \ifx\fontspec\@undefined
4463       \usepackage{fontspec}%
4464     \fi}%
4465 \fi
4466 \@onlypreamble\babelfont
4467 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4468   \bbl@foreach{#1}{%
4469     \expandafter\ifx\csname date##1\endcsname\relax
4470       \IfFileExists{babel-##1.tex}%
4471         {\babelprovide{##1}}%
4472       {}%
4473     \fi}%
4474   \edef\bbl@tempa{#1}%
4475   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4476   \bbl@loadfontspec
4477   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4478   \bbl@bblfont}
4479 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4480   \bbl@ifunset{\bbl@tempb family}%
4481     {\bbl@providefam{\bbl@tempb}}%
4482     {}%
4483   % For the default font, just in case:
4484   \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
4485   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4486   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save \bbl@rmdflt@
4487     \bbl@exp{%
4488       \let\<\bbl@tempb dflt@\languagename>\<\bbl@tempb dflt@>%
4489       \\\bbl@font@set\<\bbl@tempb dflt@\languagename>%
4490       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4491   {\bbl@foreach\bbl@tempa{% ie \bbl@rmdflt@lang / *scrt
4492     \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:

```

4493 \def\bbl@providefam#1{%
4494   \bbl@exp{%
4495     \\\newcommand\<#1default>{}% Just define it
4496     \\\bbl@add@list\\bbl@font@fams{#1}%
4497     \\\DeclareRobustCommand\<#1family>{%
4498       \\\not@math@alphabet\<#1family>\relax
4499       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4500       \\\fontfamily\<#1default>%

```

```

4501 \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4502 \\selectfont}%
4503 \\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.

```

4504 \def\bbl@nostdfont#1{%
4505   \bbl@ifunset{bbl@WFF@\f@family}%
4506   {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
4507   \bbl@infowarn{The current font is not a babel standard family:\\%
4508     #1%
4509     \fontname\font\\%
4510     There is nothing intrinsically wrong with this warning, and\\%
4511     you can ignore it altogether if you do not need these\\%
4512     families. But if they are used in the document, you should be\\%
4513     aware 'babel' will not set Script and Language for them, so\\%
4514     you may consider defining a new family with \string\babelfont.\\%
4515     See the manual for further details about \string\babelfont.\\%
4516     Reported}}
4517   {}}%
4518 \gdef\bbl@switchfont{%
4519   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4520   \bbl@exp{% eg Arabic -> arabic
4521     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4522   \bbl@foreach\bbl@font@fams{%
4523     \bbl@ifunset{bbl@##1dflt@\languagename}% (1) language?
4524     {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4525       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4526         {}}% 123=F - nothing!
4527       {\bbl@exp{% 3=T - from generic
4528         \global\let\<bbl@##1dflt@\languagename>%
4529         \<bbl@##1dflt@>}}}%
4530       {\bbl@exp{% 2=T - from script
4531         \global\let\<bbl@##1dflt@\languagename>%
4532         \<bbl@##1dflt@*\bbl@tempa>}}}%
4533       {}}% 1=T - language, already defined
4534   \def\bbl@tempa{\bbl@nostdfont{}}}%
4535   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4536     \bbl@ifunset{bbl@##1dflt@\languagename}%
4537     {\bbl@cs{famrst@##1}%
4538     \global\bbl@csarg\let{famrst@##1}\relax}%
4539     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4540       \\bbl@add\\originalTeX{%
4541       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4542       \<##1default>\<##1family>{##1}}}%
4543       \\bbl@font@set{\<bbl@##1dflt@\languagename>% the main part!
4544       \<##1default>\<##1family>}}}%
4545   \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.

```

4546 \ifx\f@family\undefined\else % if latex
4547   \ifcase\bbl@engine % if pdftex
4548     \let\bbl@ckeckstdfonts\relax
4549   \else
4550     \def\bbl@ckeckstdfonts{%
4551       \begingroup
4552       \global\let\bbl@ckeckstdfonts\relax
4553       \let\bbl@tempa\@empty
4554       \bbl@foreach\bbl@font@fams{%
4555         \bbl@ifunset{bbl@##1dflt@}%
4556         {\@nameuse{##1family}%
4557         \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4558         \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%

```

```

4559         \space\space\fontname\font\\}}}%
4560         \bbl@csarg\edef{##1dflt@}{\f@family}%
4561         \expandafter\edef\csname ##1default\endcsname{\f@family}}}%
4562     {}}%
4563     \ifx\bbl@tempa\@empty\else
4564         \bbl@infowarn{The following font families will use the default\\%
4565             settings for all or some languages:\\%
4566             \bbl@tempa
4567             There is nothing intrinsically wrong with it, but\\%
4568             'babel' will no set Script and Language, which could\\%
4569             be relevant in some languages. If your document uses\\%
4570             these families, consider redefining them with \string\babelfont.\\%
4571             Reported}%
4572     \fi
4573 \endgroup}
4574 \fi
4575 \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.

```

4576 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4577 \bbl@xin@{<>}{#1}%
4578 \ifin@
4579 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4580 \fi
4581 \bbl@exp{%
4582     \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4583     \\bbl@ifsamestring{#2}{\f@family}%
4584     {\\#3%
4585         \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4586     \let\\bbl@tempa\relax}%
4587     {}}
4588 % TODO - next should be global?, but even local does its job. I'm
4589 % still not sure -- must investigate:
4590 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4591 \let\bbl@tempa\bbl@mapselect
4592 \let\bbl@mapselect\relax
4593 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4594 \let#4\@empty % Make sure \renewfontfamily is valid
4595 \bbl@exp{%
4596     \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4597     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4598     {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4599     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4600     {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4601     \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4602     \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4603     \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4604     \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4605     \\renewfontfamily\\#4%
4606     [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{..}{#3}
4607 \bbl@exp{%
4608     \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4609     \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4610 \begingroup
4611 #4%
4612 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4613 \endgroup
4614 \let#4\bbl@temp@fam
4615 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4616 \let\bbl@mapselect\bbl@tempa}%

```

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

```
4617 \def\bb1@font@rst#1#2#3#4{%
4618   \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.

```
4619 \def\bb1@font@fams{rm,sf,tt}
4620 <</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.

```
4621 <<{*Footnote changes}>> ≡
4622 \bb1@trace{Bidi footnotes}
4623 \ifnum\bb1@bidimode>\z@
4624   \def\bb1@footnote#1#2#3{%
4625     \ifnextchar[%
4626       {\bb1@footnote@o{#1}{#2}{#3}}%
4627       {\bb1@footnote@x{#1}{#2}{#3}}}
4628   \long\def\bb1@footnote@x#1#2#3#4{%
4629     \bgroup
4630     \select@language@x{\bb1@main@language}%
4631     \bb1@fn@footnote{#2#1{\ignorespaces#4}#3}%
4632     \egroup}
4633   \long\def\bb1@footnote@o#1#2#3[#4]#5{%
4634     \bgroup
4635     \select@language@x{\bb1@main@language}%
4636     \bb1@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4637     \egroup}
4638   \def\bb1@footnotetext#1#2#3{%
4639     \@ifnextchar[%
4640       {\bb1@footnotetext@o{#1}{#2}{#3}}%
4641       {\bb1@footnotetext@x{#1}{#2}{#3}}}
4642   \long\def\bb1@footnotetext@x#1#2#3#4{%
4643     \bgroup
4644     \select@language@x{\bb1@main@language}%
4645     \bb1@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4646     \egroup}
4647   \long\def\bb1@footnotetext@o#1#2#3[#4]#5{%
4648     \bgroup
4649     \select@language@x{\bb1@main@language}%
4650     \bb1@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4651     \egroup}
4652   \def\BabelFootnote#1#2#3#4{%
4653     \ifx\bb1@fn@footnote\undefined
4654       \let\bb1@fn@footnote\footnote
4655     \fi
4656     \ifx\bb1@fn@footnotetext\undefined
4657       \let\bb1@fn@footnotetext\footnotetext
4658     \fi
4659     \bb1@ifblank{#2}%
4660     {\def#1{\bb1@footnote{\@firstofone}{#3}{#4}}
4661      \@namedef{\bb1@stripslash#1text}%
4662      {\bb1@footnotetext{\@firstofone}{#3}{#4}}}%
4663     {\def#1{\bb1@exp{\bb1@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4664      \@namedef{\bb1@stripslash#1text}%
4665      {\bb1@exp{\bb1@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4666   \fi
4667 <</Footnote changes>>
```

Now, the code.

```

4668 <*xetex>
4669 \def\BabelStringsDefault{unicode}
4670 \let\xebbl@stop\relax
4671 \AddBabelHook{xetex}{encodedcommands}{%
4672   \def\bbl@tempa{#1}%
4673   \ifx\bbl@tempa@empty
4674     \XeTeXinputencoding"bytes"%
4675   \else
4676     \XeTeXinputencoding"#1"%
4677   \fi
4678   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4679 \AddBabelHook{xetex}{stopcommands}{%
4680   \xebbl@stop
4681   \let\xebbl@stop\relax}
4682 \def\bbl@intraspace#1 #2 #3\@@{%
4683   \bbl@csarg\gdef{xeisp@\language}%
4684     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4685 \def\bbl@intrapenalty#1\@@{%
4686   \bbl@csarg\gdef{xeipn@\language}%
4687     {\XeTeXlinebreakpenalty #1\relax}}
4688 \def\bbl@provide@intraspace{%
4689   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4690   \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{lnbrk}} \fi
4691   \ifin@
4692     \bbl@ifunset{bbl@intsp@\language}{}%
4693     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4694       \ifx\bbl@KVP@intraspace\@nnil
4695         \bbl@exp{%
4696           \\bbl@intraspace\bbl@cl{intsp}\\@@}%
4697         \fi
4698         \ifx\bbl@KVP@intrapenalty\@nnil
4699           \bbl@intrapenalty0\@@
4700         \fi
4701       \fi
4702       \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4703         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4704       \fi
4705       \ifx\bbl@KVP@intrapenalty\@nnil\else
4706         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4707       \fi
4708       \bbl@exp{%
4709         % TODO. Execute only once (but redundant):
4710         \\bbl@add\<extras\language>{%
4711           \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4712           \<bbl@xeisp@\language>%
4713           \<bbl@xeipn@\language>%
4714           \\bbl@toglobal\<extras\language>%
4715           \\bbl@add\<noextras\language>{%
4716             \XeTeXlinebreaklocale ""}%
4717           \\bbl@toglobal\<noextras\language>%
4718         \ifx\bbl@ispacesize\@undefined
4719           \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4720           \ifx\AtBeginDocument\@notprerr
4721             \expandafter\@secondoftwo % to execute right now
4722           \fi
4723           \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4724         \fi}%
4725       \fi}
4726 \ifx\DisableBabelHook\@undefined\endinput\fi
4727 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4728 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4729 \DisableBabelHook{babel-fontspec}

```



```

4730 <(Font selection)>
4731 \def\bbl@provide@extra#1{}
4732 </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 pdf_{te}x and xet_{ex}.

```

4733 <*xetex | texxet>
4734 \providecommand\bbl@provide@intraspace{}
4735 \bbl@trace{Redefinitions for bidi layout}
4736 \def\bbl@sspre@caption{%
4737   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}}
4738 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4739 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4740 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4741 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4742   \def\@hangfrom#1{%
4743     \setbox\@tempboxa\hbox{#1}}%
4744   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4745   \noindent\box\@tempboxa}
4746 \def\raggedright{%
4747   \let\@centercr
4748   \bbl@startskip\z@skip
4749   \@rightskip\@flushglue
4750   \bbl@endskip\@rightskip
4751   \parindent\z@
4752   \parfillskip\bbl@startskip}
4753 \def\raggedleft{%
4754   \let\@centercr
4755   \bbl@startskip\@flushglue
4756   \bbl@endskip\z@skip
4757   \parindent\z@
4758   \parfillskip\bbl@endskip}
4759 \fi
4760 \IfBabelLayout{lists}
4761   {\bbl@sreplace\list
4762     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4763     \def\bbl@listleftmargin{%
4764       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4765     \ifcase\bbl@engine
4766       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4767       \def\p@enumiii{\p@enumii}\theenumii{}
4768     \fi
4769     \bbl@sreplace\@verbatim
4770       {\leftskip\@totalleftmargin}%
4771       {\bbl@startskip\textwidth
4772         \advance\bbl@startskip-\linewidth}%
4773     \bbl@sreplace\@verbatim
4774       {\rightskip\z@skip}%
4775     {\bbl@endskip\z@skip}}%
4776   {}
4777 \IfBabelLayout{contents}
4778   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4779     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4780   {}
4781 \IfBabelLayout{columns}
4782   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4783     \def\bbl@outputbox#1{%

```

```

4784 \hb@xt@\textwidth{%
4785 \hskip\columnwidth
4786 \hfil
4787 {\normalcolor\vrule \@width\columnseprule}%
4788 \hfil
4789 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4790 \hskip-\textwidth
4791 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4792 \hskip\columnsep
4793 \hskip\columnwidth}}}%
4794 {}
4795 <(Footnote changes)>
4796 \IfBabelLayout{footnotes}%
4797 {\BabelFootnote\footnote\languagename{}}}%
4798 \BabelFootnote\localfootnote\languagename{}}}%
4799 \BabelFootnote\mainfootnote{}}}%
4800 {}

```

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.

```

4801 \IfBabelLayout{counters*}%
4802 {\bbl@add\bbl@opt@layout{.counters.}%
4803 \AddToHook{shipout/before}{%
4804 \let\bbl@tempa\babelsublr
4805 \let\babelsublr\@firstofone
4806 \let\bbl@save@thepage\thepage
4807 \protected@edef\thepage{\thepage}%
4808 \let\babelsublr\bbl@tempa}%
4809 \AddToHook{shipout/after}{%
4810 \let\thepage\bbl@save@thepage}}}%
4811 \IfBabelLayout{counters}%
4812 {\let\bbl@latinarabic=\@arabic
4813 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4814 \let\bbl@asciroman=\@roman
4815 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4816 \let\bbl@asciiRoman=\@Roman
4817 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
4818 \fi % end if layout
4819 </xetex | texet>

```

12.3 8-bit TeX

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

```

4820 <*texet>
4821 \def\bbl@provide@extra#1{%
4822 % == auto-select encoding ==
4823 \ifx\bbl@encoding@select@off\@empty\else
4824 \bbl@ifunset{\bbl@encoding@#1}%
4825 {\def\@elt##1{,##1,}%
4826 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4827 \count@\z@
4828 \bbl@foreach\bbl@tempe{%
4829 \def\bbl@tempd{##1}% Save last declared
4830 \advance\count@\@ne}%
4831 \ifnum\count@>\@ne
4832 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4833 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4834 \bbl@replace\bbl@tempa{ },}%
4835 \global\bbl@csarg\let{encoding@#1}\@empty
4836 \bbl@xin@{\, \bbl@tempd,}{, \bbl@tempa,}%
4837 \ifin\else % if main encoding included in ini, do nothing
4838 \let\bbl@tempb\relax
4839 \bbl@foreach\bbl@tempa{%

```

```

4840      \ifx\bbbl@tempb\relax
4841      \bbbl@xin@{,##1,}{,\bbbl@tempe,}%
4842      \ifin@def\bbbl@tempb{##1}\fi
4843      \fi}%
4844      \ifx\bbbl@tempb\relax\else
4845      \bbbl@exp{%
4846      \global\<bbbl@add>\<bbbl@preextras@#1>{\<bbbl@encoding@#1>}%
4847      \gdef\<bbbl@encoding@#1>{%
4848      \\\babel@save\\f@encoding
4849      \\\bbbl@add\\originalTeX{\selectfont}%
4850      \\\fontencoding{\bbbl@tempb}%
4851      \\\selectfont}}%
4852      \fi
4853      \fi
4854      \fi}%
4855      }%
4856      \fi}
4857      \</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, \bbbl@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).

```

4858      \<!*luatex>
4859      \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4860      \bbbl@trace{Read language.dat}
4861      \ifx\bbbl@readstream\@undefined
4862      \csname newread\endcsname\bbbl@readstream
4863      \fi
4864      \begingroup
4865      \toks@{}
4866      \count@% 0=start, 1=0th, 2=normal
4867      \def\bbbl@process@line#1#2 #3 #4 {%

```

```

4868 \ifx=#1%
4869 \bbl@process@synonym{#2}%
4870 \else
4871 \bbl@process@language{#1#2}{#3}{#4}%
4872 \fi
4873 \ignorespaces}
4874 \def\bbl@manylang{%
4875 \ifnum\bbl@last>\@ne
4876 \bbl@info{Non-standard hyphenation setup}%
4877 \fi
4878 \let\bbl@manylang\relax}
4879 \def\bbl@process@language#1#2#3{%
4880 \ifcase\count@
4881 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4882 \or
4883 \count@\tw@
4884 \fi
4885 \ifnum\count@=\tw@
4886 \expandafter\addlanguage\csname l@#1\endcsname
4887 \language\allocationnumber
4888 \chardef\bbl@last\allocationnumber
4889 \bbl@manylang
4890 \let\bbl@elt\relax
4891 \xdef\bbl@languages{%
4892 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4893 \fi
4894 \the\toks@
4895 \toks@{}}
4896 \def\bbl@process@synonym@aux#1#2{%
4897 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4898 \let\bbl@elt\relax
4899 \xdef\bbl@languages{%
4900 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4901 \def\bbl@process@synonym#1{%
4902 \ifcase\count@
4903 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4904 \or
4905 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{%
4906 \else
4907 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4908 \fi}
4909 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4910 \chardef\l@english\z@
4911 \chardef\l@USenglish\z@
4912 \chardef\bbl@last\z@
4913 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
4914 \gdef\bbl@languages{%
4915 \bbl@elt{english}{0}{\hyphen.tex}}%
4916 \bbl@elt{USenglish}{0}{}}
4917 \else
4918 \global\let\bbl@languages@format\bbl@languages
4919 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4920 \ifnum#2>\z@\else
4921 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4922 \fi}%
4923 \xdef\bbl@languages{\bbl@languages}%
4924 \fi
4925 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4926 \bbl@languages
4927 \openin\bbl@readstream=language.dat
4928 \ifeof\bbl@readstream
4929 \bbl@warning{I couldn't find language.dat. No additional\\%
4930 patterns loaded. Reported}%

```

```

4931 \else
4932 \loop
4933 \endlinechar\m@ne
4934 \read\bb1@readstream to \bb1@line
4935 \endlinechar`\^^M
4936 \if T\ifeof\bb1@readstream F\fi T\relax
4937 \ifx\bb1@line\empty\else
4938 \edef\bb1@line{\bb1@line\space\space\space}%
4939 \expandafter\bb1@process@line\bb1@line\relax
4940 \fi
4941 \repeat
4942 \fi
4943 \endgroup
4944 \bb1@trace{Macros for reading patterns files}
4945 \def\bb1@get@enc#1:#2:#3\@@{\def\bb1@hyph@enc{#2}}
4946 \ifx\babelcatcodetablenum\undefined
4947 \ifx\newcatcodetable\undefined
4948 \def\babelcatcodetablenum{5211}
4949 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4950 \else
4951 \newcatcodetable\babelcatcodetablenum
4952 \newcatcodetable\bb1@pattcodes
4953 \fi
4954 \else
4955 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4956 \fi
4957 \def\bb1@luapatterns#1#2{%
4958 \bb1@get@enc#1::\@@@
4959 \setbox\z@\hbox\bgroup
4960 \begingroup
4961 \savecatcodetable\babelcatcodetablenum\relax
4962 \initcatcodetable\bb1@pattcodes\relax
4963 \catcodetable\bb1@pattcodes\relax
4964 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4965 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4966 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4967 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4968 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4969 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
4970 \input #1\relax
4971 \catcodetable\babelcatcodetablenum\relax
4972 \endgroup
4973 \def\bb1@tempa{#2}%
4974 \ifx\bb1@tempa\empty\else
4975 \input #2\relax
4976 \fi
4977 \egroup}%
4978 \def\bb1@patterns@lua#1{%
4979 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4980 \csname l@#1\endcsname
4981 \edef\bb1@tempa{#1}%
4982 \else
4983 \csname l@#1:\f@encoding\endcsname
4984 \edef\bb1@tempa{#1:\f@encoding}%
4985 \fi\relax
4986 \@namedef{luatexhyphen@loaded@the\language}{}% Temp
4987 \@ifundefined{bb1@hyphendata@the\language}%
4988 {\def\bb1@elt##1##2##3##4{%
4989 \ifnum##2=\csname l@bb1@tempa\endcsname % #2=spanish, dutch:OT1...
4990 \def\bb1@tempb{##3}%
4991 \ifx\bb1@tempb\empty\else % if not a synonymous
4992 \def\bb1@tempc{##3}{##4}}%
4993 \fi

```

```

4994      \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4995      \fi}%
4996      \bbl@languages
4997      \@ifundefined{bbl@hyphendata@the\language}%
4998      {\bbl@info{No hyphenation patterns were set for\%
4999      language '\bbl@tempa'. Reported}}%
5000      {\expandafter\expandafter\expandafter\bbl@luapatterns
5001      \csname bbl@hyphendata@the\language\endcsname}}}%
5002 \endinput\fi
5003 % Here ends \ifx\AddBabelHook\@undefined
5004 % A few lines are only read by hyphen.cfg
5005 \ifx\DisableBabelHook\@undefined
5006   \AddBabelHook{luatex}{everylanguage}{%
5007     \def\process@language##1##2##3{%
5008       \def\process@line####1####2 ####3 ####4 {}}
5009   \AddBabelHook{luatex}{loadpatterns}{%
5010     \input #1\relax
5011     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5012     {{#1}{}}}
5013   \AddBabelHook{luatex}{loadexceptions}{%
5014     \input #1\relax
5015     \def\bbl@tempb##1##2{{##1}{#1}}%
5016     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5017     {\expandafter\expandafter\expandafter\bbl@tempb
5018     \csname bbl@hyphendata@the\language\endcsname}}
5019 \endinput\fi
5020 % Here stops reading code for hyphen.cfg
5021 % The following is read the 2nd time it's loaded
5022 \begingroup % TODO - to a lua file
5023 \catcode`\%=12
5024 \catcode`\'=12
5025 \catcode`\%=12
5026 \catcode`\:=12
5027 \directlua{
5028   Babel = Babel or {}
5029   function Babel.bytes(line)
5030     return line:gsub("(.)",
5031       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5032   end
5033   function Babel.begin_process_input()
5034     if luatexbase and luatexbase.add_to_callback then
5035       luatexbase.add_to_callback('process_input_buffer',
5036         Babel.bytes, 'Babel.bytes')
5037     else
5038       Babel.callback = callback.find('process_input_buffer')
5039       callback.register('process_input_buffer', Babel.bytes)
5040     end
5041   end
5042   function Babel.end_process_input ()
5043     if luatexbase and luatexbase.remove_from_callback then
5044       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5045     else
5046       callback.register('process_input_buffer', Babel.callback)
5047     end
5048   end
5049   function Babel.addpatterns(pp, lg)
5050     local lg = lang.new(lg)
5051     local pats = lang.patterns(lg) or ''
5052     lang.clear_patterns(lg)
5053     for p in pp:gmatch('[^s]+') do
5054       ss = ''
5055       for i in string.utfcharacters(p:gsub('%d', '')) do
5056         ss = ss .. '%d?' .. i

```

```

5057     end
5058     ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5059     ss = ss:gsub('%%d%?$', '%%.')
5060     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5061     if n == 0 then
5062         tex.sprint(
5063             [[\string\csname\space bbl@info\endcsname{New pattern: }]
5064             .. p .. [{}]])
5065         pats = pats .. ' ' .. p
5066     else
5067         tex.sprint(
5068             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
5069             .. p .. [{}]])
5070     end
5071 end
5072 lang.patterns(lg, pats)
5073 end
5074 Babel.characters = Babel.characters or {}
5075 Babel.ranges = Babel.ranges or {}
5076 function Babel.hlist_has_bidi(head)
5077     local has_bidi = false
5078     local ranges = Babel.ranges
5079     for item in node.traverse(head) do
5080         if item.id == node.id'glyph' then
5081             local itemchar = item.char
5082             local chardata = Babel.characters[itemchar]
5083             local dir = chardata and chardata.d or nil
5084             if not dir then
5085                 for nn, et in ipairs(ranges) do
5086                     if itemchar < et[1] then
5087                         break
5088                     elseif itemchar <= et[2] then
5089                         dir = et[3]
5090                         break
5091                     end
5092                 end
5093             end
5094             if dir and (dir == 'al' or dir == 'r') then
5095                 has_bidi = true
5096             end
5097         end
5098     end
5099     return has_bidi
5100 end
5101 function Babel.set_chranges_b (script, chrng)
5102     if chrng == '' then return end
5103     texio.write('Replacing ' .. script .. ' script ranges')
5104     Babel.script_blocks[script] = {}
5105     for s, e in string.gmatch(chrng..' ', '(.-%).%.(.)%s') do
5106         table.insert(
5107             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5108     end
5109 end
5110 function Babel.discard_sublr(str)
5111     if str:find( [[\string\indexentry]] ) and
5112        str:find( [[\string\babelsublr]] ) then
5113         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5114             function(m) return m:sub(2,-2) end )
5115     end
5116     return str
5117 end
5118 }
5119 \endgroup

```

```

5120 \ifx\newattribute\@undefined\else
5121   \newattribute\bbl@attr@locale
5122   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5123   \AddBabelHook{luatex}{beforeextras}{%
5124     \setattribute\bbl@attr@locale\localeid}
5125 \fi
5126 \def\BabelStringsDefault{unicode}
5127 \let\luabbl@stop\relax
5128 \AddBabelHook{luatex}{encodedcommands}{%
5129   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5130   \ifx\bbl@tempa\bbl@tempb\else
5131     \directlua{Babel.begin_process_input()}%
5132     \def\luabbl@stop{%
5133       \directlua{Babel.end_process_input()}}%
5134   \fi}%
5135 \AddBabelHook{luatex}{stopcommands}{%
5136   \luabbl@stop
5137   \let\luabbl@stop\relax}
5138 \AddBabelHook{luatex}{patterns}{%
5139   \@ifundefined{bbl@hyphendata@the\language}%
5140     {\def\bbl@elt##1##2##3##4{%
5141       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5142       \def\bbl@tempb{##3}%
5143       \ifx\bbl@tempb\@empty\else % if not a synonymous
5144         \def\bbl@tempc{##3}{##4}}%
5145       \fi
5146       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5147     \fi}%
5148   \bbl@languages
5149   \@ifundefined{bbl@hyphendata@the\language}%
5150     {\bbl@info{No hyphenation patterns were set for\%
5151       language '#2'. Reported}}%
5152     {\expandafter\expandafter\expandafter\bbl@luapatterns
5153       \csname bbl@hyphendata@the\language\endcsname}}}%
5154   \@ifundefined{bbl@patterns@}{}%
5155   \begin{group}
5156     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5157     \ifin@ \else
5158       \ifx\bbl@patterns@\@empty\else
5159         \directlua{ Babel.addpatterns(
5160           [[\bbl@patterns@]], \number\language) }%
5161       \fi
5162       \@ifundefined{bbl@patterns@#1}%
5163       \@empty
5164       {\directlua{ Babel.addpatterns(
5165         [[\space\csname bbl@patterns@#1\endcsname]],
5166         \number\language) }}%
5167       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5168     \fi
5169   \end{group}%
5170   \bbl@exp{%
5171     \bbl@ifunset{bbl@prehc@languagename}{}%
5172     {\bbl@ifblank{\bbl@cs{prehc@languagename}}{}}%
5173     {\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.

```

5174 \@onlypreamble\babelpatterns
5175 \AtEndOfPackage{%
5176   \newcommand\babelpatterns[2][\@empty]{%
5177     \ifx\bbl@patterns@\relax
5178       \let\bbl@patterns@\@empty

```



```

5179 \fi
5180 \ifx\bb1@pttnlist\@empty\else
5181 \bb1@warning{%
5182 You must not intermingle \string\selectlanguage\space and\\%
5183 \string\babelpatterns\space or some patterns will not\\%
5184 be taken into account. Reported}%
5185 \fi
5186 \ifx\@empty#1%
5187 \protected@edef\bb1@patterns@\bb1@patterns@\space#2}%
5188 \else
5189 \edef\bb1@tempb{\zap@space#1 \@empty}%
5190 \bb1@for\bb1@tempa\bb1@tempb{%
5191 \bb1@fixname\bb1@tempa
5192 \bb1@iflanguage\bb1@tempa{%
5193 \bb1@csarg\protected@edef{patterns@\bb1@tempa}{%
5194 \@ifundefined{bb1@patterns@\bb1@tempa}%
5195 \@empty
5196 {\csname bbl@patterns@\bb1@tempa\endcsname\space}%
5197 #2}}}%
5198 \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.

```

5199% TODO - to a lua file
5200\directlua{
5201 Babel = Babel or {}
5202 Babel.linebreaking = Babel.linebreaking or {}
5203 Babel.linebreaking.before = {}
5204 Babel.linebreaking.after = {}
5205 Babel.locale = {} % Free to use, indexed by \localeid
5206 function Babel.linebreaking.add_before(func)
5207 tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname]])
5208 table.insert(Babel.linebreaking.before, func)
5209 end
5210 function Babel.linebreaking.add_after(func)
5211 tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname]])
5212 table.insert(Babel.linebreaking.after, func)
5213 end
5214 }
5215\def\bb1@intraspace#1 #2 #3\@{%
5216 \directlua{
5217 Babel = Babel or {}
5218 Babel.intraspaces = Babel.intraspaces or {}
5219 Babel.intraspaces['\csname bbl@sbc@languagename\endcsname'] = %
5220 {b = #1, p = #2, m = #3}
5221 Babel.locale_props[\the\localeid].intraspace = %
5222 {b = #1, p = #2, m = #3}
5223 }}
5224\def\bb1@intrapenalty#1\@{%
5225 \directlua{
5226 Babel = Babel or {}
5227 Babel.intrapenalties = Babel.intrapenalties or {}
5228 Babel.intrapenalties['\csname bbl@sbc@languagename\endcsname'] = #1
5229 Babel.locale_props[\the\localeid].intrapenalty = #1
5230 }}
5231\begingroup
5232\catcode`\%=12
5233\catcode`\^=14
5234\catcode`\'=12

```

```

5235 \catcode`\~ = 12
5236 \gdef\bbl@seaintraspace{^
5237 \let\bbl@seaintraspace\relax
5238 \directlua{
5239   Babel = Babel or {}
5240   Babel.sea_enabled = true
5241   Babel.sea_ranges = Babel.sea_ranges or {}
5242   function Babel.set_chranges (script, chrng)
5243     local c = 0
5244     for s, e in string.gmatch(chrng..' ', '(.-%.-%.-%s') do
5245       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5246       c = c + 1
5247     end
5248   end
5249   function Babel.sea_disc_to_space (head)
5250     local sea_ranges = Babel.sea_ranges
5251     local last_char = nil
5252     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5253     for item in node.traverse(head) do
5254       local i = item.id
5255       if i == node.id'glyph' then
5256         last_char = item
5257       elseif i == 7 and item.subtype == 3 and last_char
5258         and last_char.char > 0x0C99 then
5259         quad = font.getfont(last_char.font).size
5260         for lg, rg in pairs(sea_ranges) do
5261           if last_char.char > rg[1] and last_char.char < rg[2] then
5262             lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5263             local intraspace = Babel.intraspaces[lg]
5264             local intrapenalty = Babel.intrapenalties[lg]
5265             local n
5266             if intrapenalty ~= 0 then
5267               n = node.new(14, 0)      ^% penalty
5268               n.penalty = intrapenalty
5269               node.insert_before(head, item, n)
5270             end
5271             n = node.new(12, 13)      ^% (glue, spaceskip)
5272             node.setglue(n, intraspace.b * quad,
5273               intraspace.p * quad,
5274               intraspace.m * quad)
5275             node.insert_before(head, item, n)
5276             node.remove(head, item)
5277           end
5278         end
5279       end
5280     end
5281   end
5282 }^^
5283 \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.

```

5284 \catcode`\% = 14
5285 \gdef\bbl@cjkkintraspace{%
5286 \let\bbl@cjkkintraspace\relax
5287 \directlua{
5288   Babel = Babel or {}

```

```

5289 require('babel-data-cjk.lua')
5290 Babel.cjk_enabled = true
5291 function Babel.cjk_linebreak(head)
5292     local GLYPH = node.id'glyph'
5293     local last_char = nil
5294     local quad = 655360          % 10 pt = 655360 = 10 * 65536
5295     local last_class = nil
5296     local last_lang = nil
5297
5298     for item in node.traverse(head) do
5299         if item.id == GLYPH then
5300
5301             local lang = item.lang
5302
5303             local LOCALE = node.get_attribute(item,
5304                 Babel.attr_locale)
5305             local props = Babel.locale_props[LOCALE]
5306
5307             local class = Babel.cjk_class[item.char].c
5308
5309             if props.cjk_quotes and props.cjk_quotes[item.char] then
5310                 class = props.cjk_quotes[item.char]
5311             end
5312
5313             if class == 'cp' then class = 'cl' end % ]] as CL
5314             if class == 'id' then class = 'I' end
5315
5316             local br = 0
5317             if class and last_class and Babel.cjk_breaks[last_class][class] then
5318                 br = Babel.cjk_breaks[last_class][class]
5319             end
5320
5321             if br == 1 and props.linebreak == 'c' and
5322                 lang ~= \the\l@nohyphenation\space and
5323                 last_lang ~= \the\l@nohyphenation then
5324                 local intrapenalty = props.intrapenalty
5325                 if intrapenalty ~= 0 then
5326                     local n = node.new(14, 0)      % penalty
5327                     n.penalty = intrapenalty
5328                     node.insert_before(head, item, n)
5329                 end
5330                 local intraspace = props.intraspace
5331                 local n = node.new(12, 13)         % (glue, spaceskip)
5332                 node.setglue(n, intraspace.b * quad,
5333                     intraspace.p * quad,
5334                     intraspace.m * quad)
5335                 node.insert_before(head, item, n)
5336             end
5337
5338             if font.getfont(item.font) then
5339                 quad = font.getfont(item.font).size
5340             end
5341             last_class = class
5342             last_lang = lang
5343         else % if penalty, glue or anything else
5344             last_class = nil
5345         end
5346     end
5347     lang.hyphenate(head)
5348 end
5349 }%
5350 \bbl@luahyphenate}
5351 \gdef\bbl@luahyphenate{%

```

```

5352 \let\bbl@luahyphenate\relax
5353 \directlua{
5354   luatexbase.add_to_callback('hyphenate',
5355   function (head, tail)
5356     if Babel.linebreaking.before then
5357       for k, func in ipairs(Babel.linebreaking.before) do
5358         func(head)
5359       end
5360     end
5361     if Babel.cjk_enabled then
5362       Babel.cjk_linebreak(head)
5363     end
5364     lang.hyphenate(head)
5365     if Babel.linebreaking.after then
5366       for k, func in ipairs(Babel.linebreaking.after) do
5367         func(head)
5368       end
5369     end
5370     if Babel.sea_enabled then
5371       Babel.sea_disc_to_space(head)
5372     end
5373   end,
5374   'Babel.hyphenate')
5375 }
5376 }
5377 \endgroup
5378 \def\bbl@provide@intraspace{%
5379   \bbl@ifunset{\bbl@intsp@{language}}{%
5380     {\xexpandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5381       \bbl@xin@{/c}{\bbl@cl{lnbrk}}%
5382       \ifin@           % cjk
5383       \bbl@cjk@intraspace
5384       \directlua{
5385         Babel = Babel or {}
5386         Babel.locale_props = Babel.locale_props or {}
5387         Babel.locale_props[\the\localeid].linebreak = 'c'
5388       }%
5389       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5390       \ifx\bbl@KVP@intrapenalty\@nnil
5391         \bbl@intrapenalty0\@
5392       \fi
5393     \else           % sea
5394       \bbl@sea@intraspace
5395       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5396       \directlua{
5397         Babel = Babel or {}
5398         Babel.sea_ranges = Babel.sea_ranges or {}
5399         Babel.set_chranges('\bbl@cl{sbcpr}',
5400                           '\bbl@cl{chrng}')
5401       }%
5402       \ifx\bbl@KVP@intrapenalty\@nnil
5403         \bbl@intrapenalty0\@
5404       \fi
5405     \fi
5406   \fi
5407   \ifx\bbl@KVP@intrapenalty\@nnil\else
5408     \xexpandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5409   \fi}}

```

12.7 Arabic justification

```

5410 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5411 \def\bblar@chars{%

```

```

5412 0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5413 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5414 0640,0641,0642,0643,0644,0645,0646,0647,0649}
5415 \def\bblar@elongated{%
5416 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5417 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5418 0649,064A}
5419 \begingroup
5420 \catcode\_ =11 \catcode\`:=11
5421 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5422 \endgroup
5423 \gdef\bbl@arabicjust{%
5424 \let\bbl@arabicjust\relax
5425 \newattribute\bblar@kashida
5426 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5427 \bblar@kashida=\z@
5428 \bbl@patchfont{{\bbl@parsejalt}}%
5429 \directlua{
5430 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5431 Babel.arabic.elong_map[\the\localeid] = {}
5432 luatexbase.add_to_callback('post_linebreak_filter',
5433 Babel.arabic.justify, 'Babel.arabic.justify')
5434 luatexbase.add_to_callback('hpack_filter',
5435 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5436 }%
5437 % Save both node lists to make replacement. TODO. Save also widths to
5438 % make computations
5439 \def\bblar@fetchjalt#1#2#3#4{%
5440 \bbl@exp{\bbl@foreach{#1}}{%
5441 \bbl@ifunset{\bblar@JE##1}%
5442 {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5443 {\setbox\z@\hbox{^^^200d\char"\@nameuse{\bblar@JE##1}#2}}%
5444 \directlua{%
5445 local last = nil
5446 for item in node.traverse(tex.box[0].head) do
5447 if item.id == node.id'glyph' and item.char > 0x600 and
5448 not (item.char == 0x200D) then
5449 last = item
5450 end
5451 end
5452 Babel.arabic.#3['##1#4'] = last.char
5453 }}
5454 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5455 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5456 % positioning?
5457 \gdef\bbl@parsejalt{%
5458 \ifx\addfontfeature\undefined\else
5459 \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
5460 \ifin@
5461 \directlua{%
5462 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5463 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5464 tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5465 end
5466 }%
5467 \fi
5468 \fi}
5469 \gdef\bbl@parsejalti{%
5470 \begingroup
5471 \let\bbl@parsejalt\relax % To avoid infinite loop
5472 \edef\bbl@tempb{\fontid\font}%
5473 \bblar@nofswarn
5474 \bblar@fetchjalt\bblar@elongated{{from}}}%

```

```

5475 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5476 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5477 \addfontfeature{RawFeature+=jalt}%
5478 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5479 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5480 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5481 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5482 \directlua{%
5483     for k, v in pairs(Babel.arabic.from) do
5484         if Babel.arabic.dest[k] and
5485             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5486             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5487                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5488         end
5489     end
5490 }%
5491 \endgroup}
5492 %
5493 \begingroup
5494 \catcode`#=11
5495 \catcode`~=11
5496 \directlua{
5497
5498 Babel.arabic = Babel.arabic or {}
5499 Babel.arabic.from = {}
5500 Babel.arabic.dest = {}
5501 Babel.arabic.justify_factor = 0.95
5502 Babel.arabic.justify_enabled = true
5503
5504 function Babel.arabic.justify(head)
5505     if not Babel.arabic.justify_enabled then return head end
5506     for line in node.traverse_id(node.id'hlist', head) do
5507         Babel.arabic.justify_hlist(head, line)
5508     end
5509     return head
5510 end
5511
5512 function Babel.arabic.justify_hbox(head, gc, size, pack)
5513     local has_inf = false
5514     if Babel.arabic.justify_enabled and pack == 'exactly' then
5515         for n in node.traverse_id(12, head) do
5516             if n.stretch_order > 0 then has_inf = true end
5517         end
5518         if not has_inf then
5519             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5520         end
5521     end
5522     return head
5523 end
5524
5525 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5526     local d, new
5527     local k_list, k_item, pos_inline
5528     local width, width_new, full, k_curr, wt_pos, goal, shift
5529     local subst_done = false
5530     local elong_map = Babel.arabic.elong_map
5531     local last_line
5532     local GLYPH = node.id'glyph'
5533     local KASHIDA = Babel.attr_kashida
5534     local LOCALE = Babel.attr_locale
5535
5536     if line == nil then
5537         line = {}

```

```

5538     line.glue_sign = 1
5539     line.glue_order = 0
5540     line.head = head
5541     line.shift = 0
5542     line.width = size
5543 end
5544
5545 % Exclude last line. todo. But-- it discards one-word lines, too!
5546 % ? Look for glue = 12:15
5547 if (line.glue_sign == 1 and line.glue_order == 0) then
5548     elongs = {}      % Stores elongated candidates of each line
5549     k_list = {}      % And all letters with kashida
5550     pos_inline = 0   % Not yet used
5551
5552     for n in node.traverse_id(GLYPH, line.head) do
5553         pos_inline = pos_inline + 1 % To find where it is. Not used.
5554
5555         % Elongated glyphs
5556         if elong_map then
5557             local locale = node.get_attribute(n, LOCALE)
5558             if elong_map[locale] and elong_map[locale][n.font] and
5559                 elong_map[locale][n.font][n.char] then
5560                 table.insert(elongs, {node = n, locale = locale} )
5561                 node.set_attribute(n.prev, KASHIDA, 0)
5562             end
5563         end
5564
5565         % Tatwil
5566         if Babel.kashida_wts then
5567             local k_wt = node.get_attribute(n, KASHIDA)
5568             if k_wt > 0 then % todo. parameter for multi inserts
5569                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5570             end
5571         end
5572
5573     end % of node.traverse_id
5574
5575     if #elongs == 0 and #k_list == 0 then goto next_line end
5576     full = line.width
5577     shift = line.shift
5578     goal = full * Babel.arabic.justify_factor % A bit crude
5579     width = node.dimensions(line.head) % The 'natural' width
5580
5581     % == Elongated ==
5582     % Original idea taken from 'chickenize'
5583     while (#elongs > 0 and width < goal) do
5584         subst_done = true
5585         local x = #elongs
5586         local curr = elongs[x].node
5587         local oldchar = curr.char
5588         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5589         width = node.dimensions(line.head) % Check if the line is too wide
5590         % Substitute back if the line would be too wide and break:
5591         if width > goal then
5592             curr.char = oldchar
5593             break
5594         end
5595         % If continue, pop the just substituted node from the list:
5596         table.remove(elongs, x)
5597     end
5598
5599     % == Tatwil ==
5600     if #k_list == 0 then goto next_line end

```

```

5601
5602 width = node.dimensions(line.head)    % The 'natural' width
5603 k_curr = #k_list
5604 wt_pos = 1
5605
5606 while width < goal do
5607     subst_done = true
5608     k_item = k_list[k_curr].node
5609     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5610         d = node.copy(k_item)
5611         d.char = 0x0640
5612         line.head, new = node.insert_after(line.head, k_item, d)
5613         width_new = node.dimensions(line.head)
5614         if width > goal or width == width_new then
5615             node.remove(line.head, new) % Better compute before
5616             break
5617         end
5618         width = width_new
5619     end
5620     if k_curr == 1 then
5621         k_curr = #k_list
5622         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5623     else
5624         k_curr = k_curr - 1
5625     end
5626 end
5627
5628 ::next_line::
5629
5630 % Must take into account marks and ins, see luatex manual.
5631 % Have to be executed only if there are changes. Investigate
5632 % what's going on exactly.
5633 if subst_done and not gc then
5634     d = node.hpack(line.head, full, 'exactly')
5635     d.shift = shift
5636     node.insert_before(head, line, d)
5637     node.remove(head, line)
5638 end
5639 end % if process line
5640 end
5641 }
5642 \endgroup
5643 \fi\fi % Arabic just block

```

12.8 Common stuff

```

5644 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5645 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
5646 \DisableBabelHook{babel-fontspec}
5647 <(<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 from 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.

```

5648 % TODO - to a lua file
5649 \directlua{
5650 Babel.script_blocks = {
5651     ['dflt'] = {},

```



```

5652 ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5653           {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5654 ['Armn'] = {{0x0530, 0x058F}},
5655 ['Beng'] = {{0x0980, 0x09FF}},
5656 ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5657 ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5658 ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5659           {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5660 ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5661 ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5662           {0xAB00, 0xAB2F}},
5663 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5664 % Don't follow strictly Unicode, which places some Coptic letters in
5665 % the 'Greek and Coptic' block
5666 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5667 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5668           {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5669           {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5670           {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5671           {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5672           {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5673 ['Hebr'] = {{0x0590, 0x05FF}},
5674 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5675           {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5676 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5677 ['Knda'] = {{0x0C80, 0x0CFF}},
5678 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5679           {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5680           {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5681 ['Lao'] = {{0x0E80, 0x0EFF}},
5682 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5683           {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5684           {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5685 ['Mahj'] = {{0x11150, 0x1117F}},
5686 ['Mlym'] = {{0x0D00, 0x0D7F}},
5687 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5688 ['Orya'] = {{0x0B00, 0x0B7F}},
5689 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5690 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5691 ['Taml'] = {{0x0B80, 0x0BFF}},
5692 ['Telu'] = {{0x0C00, 0x0C7F}},
5693 ['Tfng'] = {{0x2D30, 0x2D7F}},
5694 ['Thai'] = {{0x0E00, 0x0E7F}},
5695 ['Tibt'] = {{0x0F00, 0x0FFF}},
5696 ['Vaii'] = {{0xA500, 0xA63F}},
5697 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5698 }
5699
5700 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyr1
5701 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5702 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5703
5704 function Babel.locale_map(head)
5705   if not Babel.locale_mapped then return head end
5706
5707   local LOCALE = Babel.attr_locale
5708   local GLYPH = node.id('glyph')
5709   local inmath = false
5710   local toloc_save
5711   for item in node.traverse(head) do
5712     local toloc
5713     if not inmath and item.id == GLYPH then
5714       % Optimization: build a table with the chars found

```

```

5715     if Babel.chr_to_loc[item.char] then
5716         toloc = Babel.chr_to_loc[item.char]
5717     else
5718         for lc, maps in pairs(Babel.loc_to_scr) do
5719             for _, rg in pairs(maps) do
5720                 if item.char >= rg[1] and item.char <= rg[2] then
5721                     Babel.chr_to_loc[item.char] = lc
5722                     toloc = lc
5723                     break
5724                 end
5725             end
5726         end
5727     end
5728     % Now, take action, but treat composite chars in a different
5729     % fashion, because they 'inherit' the previous locale. Not yet
5730     % optimized.
5731     if not toloc and
5732         (item.char >= 0x0300 and item.char <= 0x036F) or
5733         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5734         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5735         toloc = toloc_save
5736     end
5737     if toloc and Babel.locale_props[toloc] and
5738         Babel.locale_props[toloc].letters and
5739         tex.getcatcode(item.char) \string~= 11 then
5740         toloc = nil
5741     end
5742     if toloc and toloc > -1 then
5743         if Babel.locale_props[toloc].lg then
5744             item.lang = Babel.locale_props[toloc].lg
5745             node.set_attribute(item, LOCALE, toloc)
5746         end
5747         if Babel.locale_props[toloc]['/'..item.font] then
5748             item.font = Babel.locale_props[toloc]['/'..item.font]
5749         end
5750         toloc_save = toloc
5751     end
5752     elseif not inmath and item.id == 7 then % Apply recursively
5753         item.replace = item.replace and Babel.locale_map(item.replace)
5754         item.pre      = item.pre and Babel.locale_map(item.pre)
5755         item.post      = item.post and Babel.locale_map(item.post)
5756     elseif item.id == node.id'math' then
5757         inmath = (item.subtype == 0)
5758     end
5759 end
5760 return head
5761 end
5762 }

```

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

```

5763 \newcommand\babelcharproperty[1]{%
5764   \count@=#1\relax
5765   \ifvmode
5766     \expandafter\bbl@chprop
5767   \else
5768     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5769               vertical mode (preamble or between paragraphs)}%
5770     {See the manual for futher info}%
5771   \fi}
5772 \newcommand\bbl@chprop[3][\the\count@]{%
5773   \@tempcnta=#1\relax
5774   \bbl@ifunset{\bbl@chprop@#2}%

```

```

5775     {\bbl@error{No property named '#2'. Allowed values are\\%
5776                 direction (bc), mirror (bmg), and linebreak (lb)}}%
5777     {See the manual for futher info}}%
5778     {}%
5779     \loop
5780         \bbl@cs{chprop@#2}{#3}%
5781         \ifnum\count@<\@tempcnta
5782             \advance\count@\@ne
5783         \repeat}
5784 \def\bbl@chprop@direction#1{%
5785     \directlua{
5786         Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5787         Babel.characters[\the\count@]['d'] = '#1'
5788     }}
5789 \let\bbl@chprop@bc\bbl@chprop@direction
5790 \def\bbl@chprop@mirror#1{%
5791     \directlua{
5792         Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5793         Babel.characters[\the\count@]['m'] = '\number#1'
5794     }}
5795 \let\bbl@chprop@bmg\bbl@chprop@mirror
5796 \def\bbl@chprop@linebreak#1{%
5797     \directlua{
5798         Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5799         Babel.cjk_characters[\the\count@]['c'] = '#1'
5800     }}
5801 \let\bbl@chprop@lb\bbl@chprop@linebreak
5802 \def\bbl@chprop@locale#1{%
5803     \directlua{
5804         Babel.chr_to_loc = Babel.chr_to_loc or {}
5805         Babel.chr_to_loc[\the\count@] =
5806             \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5807     }}

```

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.

```

5808 \directlua{
5809     Babel.nohyphenation = \the\l@nohyphenation
5810 }

```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ becomes $\text{function}(m)$ `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 $\text{function}(m)$ `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).

```

5811 \begingroup
5812 \catcode`\~ = 12
5813 \catcode`\% = 12
5814 \catcode`\& = 14
5815 \catcode`\| = 12
5816 \gdef\babelprehyphenation{%%
5817     \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}[]]}
5818 \gdef\babelposthyphenation{%%
5819     \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}[]]}
5820 \gdef\bbl@postlinebreak{\bbl@settransform{2}}[] & WIP
5821 \gdef\bbl@settransform#1[#2]#3#4#5{%%
5822     \ifcase#1
5823         \bbl@activateprehyphen
5824     \or
5825         \bbl@activateposthyphen

```

```

5826 \fi
5827 \begingroup
5828 \def\babeltempa{\bbl@add@list\babeltempb}&%
5829 \let\babeltempb\empty
5830 \def\bbl@tempa{#5}&%
5831 \bbl@replace\bbl@tempa{,}{,}%&% TODO. Ugly trick to preserve {}
5832 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5833   \bbl@ifsamestring{##1}{remove}&%
5834   {\bbl@add@list\babeltempb{nil}}&%
5835   {\directlua{
5836     local rep = {[##1]=}
5837     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5838     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5839     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5840     if #1 == 0 or #1 == 2 then
5841       rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5842         'space = {' .. '%2, %3, %4' .. '}')
5843       rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5844         'spacefactor = {' .. '%2, %3, %4' .. '}')
5845       rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5846     else
5847       rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5848       rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5849       rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5850     end
5851     tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
5852   }}&%
5853 \bbl@foreach\babeltempb{&%
5854   \bbl@forkv{##1}{&%
5855     \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,&%
5856     no,post,penalty,kashida,space,spacefactor,}&%
5857     \ifin\else
5858     \bbl@error
5859     {Bad option '###1' in a transform.\\&%
5860     I'll ignore it but expect more errors}&%
5861     {See the manual for further info.}&%
5862   \fi}}&%
5863 \let\bbl@kv@attribute\relax
5864 \let\bbl@kv@label\relax
5865 \let\bbl@kv@font\relax
5866 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
5867 \ifx\bbl@kv@attribute\relax
5868   \ifx\bbl@kv@label\relax\else
5869     \bbl@ifunset{\bbl@ATR@\bbl@kv@label @#3}&%
5870     {\bbl@csarg\newattribute{ATR@\bbl@kv@label @#3}}&%
5871     {}&%
5872     \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3}&%
5873     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@one
5874   \fi
5875 \else
5876   \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5877 \fi
5878 \directlua{
5879   local lbr = Babel.linebreaking.replacements[#1]
5880   local u = unicode.utf8
5881   local id, attr, label
5882   if #1 == 0 or #1 == 2 then
5883     id = \the\csname bbl@id@@#3\endcsname\space
5884   else
5885     id = \the\csname l@#3\endcsname\space
5886   end
5887   \ifx\bbl@kv@attribute\relax
5888     attr = -1

```

```

5889 \else
5890   attr = luatexbase.registernumber'\bbl@kv@attribute'
5891 \fi
5892 \ifx\bbl@kv@label\relax\else %% Same refs:
5893   label = [==[\bbl@kv@label]==]
5894 \fi
5895 %% Convert pattern:
5896 local patt = string.gsub([==[#4]==], '%s', '')
5897 if #1 == 0 or #1 == 2 then
5898   patt = string.gsub(patt, '|', ' ')
5899 end
5900 if not u.find(patt, '()', nil, true) then
5901   patt = '()' .. patt .. '()'
5902 end
5903 if #1 == 1 then
5904   patt = string.gsub(patt, '%(%)^', '^()')
5905   patt = string.gsub(patt, '%$$(%)', '()$')
5906 end
5907 patt = u.gsub(patt, '{(.)}',
5908   function (n)
5909     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5910   end)
5911 patt = u.gsub(patt, '{(%x%x%x%x+)}',
5912   function (n)
5913     return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5914   end)
5915 lbkr[id] = lbkr[id] or {}
5916 table.insert(lbkr[id],
5917   { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5918 }%%
5919 \endgroup}
5920 \endgroup
5921 \DeclareRobustCommand\enablelocaletransform[2][\language]{%
5922   \bbl@csarg\setattribute{ATR@#2@#1}\@ne}
5923 \DeclareRobustCommand\disablelocaletransform[2][\language]{%
5924   \bbl@csarg\unsetattribute{ATR@#2@#1}}
5925 \def\bbl@activateposthyphen{%
5926   \let\bbl@activateposthyphen\relax
5927   \directlua{
5928     require('babel-transforms.lua')
5929     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5930   }}
5931 \def\bbl@activateprehyphen{%
5932   \let\bbl@activateprehyphen\relax
5933   \directlua{
5934     require('babel-transforms.lua')
5935     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5936   }}

```

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.

```

5937 \def\bbl@activate@preotf{%
5938   \let\bbl@activate@preotf\relax % only once
5939   \directlua{
5940     Babel = Babel or {}
5941     %
5942     function Babel.pre_otfload_v(head)
5943       if Babel.numbers and Babel.digits_mapped then
5944         head = Babel.numbers(head)
5945       end

```

```

5946     if Babel.bidi_enabled then
5947         head = Babel.bidi(head, false, dir)
5948     end
5949     return head
5950 end
5951 %
5952 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5953     if Babel.numbers and Babel.digits_mapped then
5954         head = Babel.numbers(head)
5955     end
5956     if Babel.bidi_enabled then
5957         head = Babel.bidi(head, false, dir)
5958     end
5959     return head
5960 end
5961 %
5962 luatexbase.add_to_callback('pre_linebreak_filter',
5963     Babel.pre_otfload_v,
5964     'Babel.pre_otfload_v',
5965     luatexbase.priority_in_callback('pre_linebreak_filter',
5966         'luaotfload.node_processor') or nil)
5967 %
5968 luatexbase.add_to_callback('hpack_filter',
5969     Babel.pre_otfload_h,
5970     'Babel.pre_otfload_h',
5971     luatexbase.priority_in_callback('hpack_filter',
5972         'luaotfload.node_processor') or nil)
5973 }}

```

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=`.

```

5974 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5975     \let\bbl@beforeforeign\leavevmode
5976     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5977     \RequirePackage{luatexbase}
5978     \bbl@activate@preotf
5979     \directlua{
5980         require('babel-data-bidi.lua')
5981         \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5982             require('babel-bidi-basic.lua')
5983         \or
5984             require('babel-bidi-basic-r.lua')
5985         \fi}
5986     % TODO - to locale_props, not as separate attribute
5987     \newattribute\bbl@attr@dir
5988     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5989     % TODO. I don't like it, hackish:
5990     \bbl@exp{\output{\bodydir\pagedir\the\output}}
5991     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5992 \fi\fi
5993 \chardef\bbl@thetextdir\z@
5994 \chardef\bbl@thepardir\z@
5995 \def\bbl@getluadir#1{%
5996     \directlua{
5997         if tex.#1dir == 'TLT' then
5998             tex.sprint('0')
5999         elseif tex.#1dir == 'TRT' then
6000             tex.sprint('1')
6001         end}}
6002 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6003     \ifcase#3\relax
6004         \ifcase\bbl@getluadir{#1}\relax\else

```

```

6005      #2 TLT\relax
6006      \fi
6007    \else
6008      \ifcase\bbbl@getluadir{#1}\relax
6009      #2 TRT\relax
6010      \fi
6011    \fi}
6012 \def\bbbl@thedir{0}
6013 \def\bbbl@texkdir#1{%
6014   \bbbl@setluadir{text}\texkdir{#1}%
6015   \chardef\bbbl@thetexkdir#1\relax
6016   \edef\bbbl@thedir{\the\numexpr\bbbl@thepardir*3+#1}%
6017   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*3+#1}}
6018 \def\bbbl@pardir#1{%
6019   \bbbl@setluadir{par}\pardir{#1}%
6020   \chardef\bbbl@thepardir#1\relax}
6021 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}
6022 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}
6023 \def\bbbl@dirparastext{\pardir\the\texkdir\relax}%   %%%
6024 %
6025 \ifnum\bbbl@bidimode>\z@
6026   \def\bbbl@insidemath{0}%
6027   \def\bbbl@everymath{\def\bbbl@insidemath{1}}
6028   \def\bbbl@everydisplay{\def\bbbl@insidemath{2}}
6029   \frozen@everymath\expandafter{%
6030     \expandafter\bbbl@everymath\the\frozen@everymath}
6031   \frozen@everydisplay\expandafter{%
6032     \expandafter\bbbl@everydisplay\the\frozen@everydisplay}
6033   \AtBeginDocument{
6034     \directlua{
6035       function Babel.math_box_dir(head)
6036         if not (token.get_macro('bbbl@insidemath') == '0') then
6037           if Babel.hlist_has_bidi(head) then
6038             local d = node.new(node.id'dir')
6039             d.dir = '+TRT'
6040             node.insert_before(head, node.has_glyph(head), d)
6041             for item in node.traverse(head) do
6042               node.set_attribute(item,
6043                 Babel.attr_dir, token.get_macro('bbbl@thedir'))
6044             end
6045           end
6046         end
6047         return head
6048       end
6049       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6050         "Babel.math_box_dir", 0)
6051     }%%
6052   \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`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolum` still fails.

```

6053 \bbl@trace{Redefinitions for bidi layout}
6054 %
6055 <{*More package options}> ≡
6056 \chardef\bbl@eqnpos\z@
6057 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6058 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6059 <{/More package options}>
6060 %
6061 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6062 \ifnum\bbl@bidimode>\z@
6063   \ifx\matheqdirmode\undefined\else
6064     \matheqdirmode\@ne
6065   \fi
6066   \let\bbl@eqnodir\relax
6067   \def\bbl@eqdel{()}
6068   \def\bbl@eqnum{%
6069     {\normalfont\normalcolor
6070     \expandafter\@firstoftwo\bbl@eqdel
6071     \theequation
6072     \expandafter\@secondoftwo\bbl@eqdel}}
6073   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6074   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6075   \def\bbl@eqno@flip#1{%
6076     \ifdim\predisplaysize=-\maxdimen
6077       \eqno
6078       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6079     \else
6080       \leqno\hbox{#1}%
6081     \fi}
6082   \def\bbl@leqno@flip#1{%
6083     \ifdim\predisplaysize=-\maxdimen
6084       \leqno
6085       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6086     \else
6087       \eqno\hbox{#1}%
6088     \fi}
6089   \AtBeginDocument{%
6090     \ifx\maketag@@@\undefined % Normal equation, eqnarray
6091       \AddToHook{env/equation/begin}{%
6092         \ifnum\bbl@thetextdir>\z@
6093           \let\eqnnum\bbl@eqnum
6094           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6095           \chardef\bbl@thetextdir\z@
6096           \bbl@add\normalfont{\bbl@eqnodir}%
6097           \ifcase\bbl@eqnpos
6098             \let\bbl@puteqno\bbl@eqno@flip
6099           \or
6100             \let\bbl@puteqno\bbl@leqno@flip
6101           \fi
6102         \fi}%
6103     \ifnum\bbl@eqnpos=\tw@\else
6104       \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6105     \fi
6106     \AddToHook{env/eqnarray/begin}{%
6107       \ifnum\bbl@thetextdir>\z@
6108         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6109         \chardef\bbl@thetextdir\z@
6110         \bbl@add\normalfont{\bbl@eqnodir}%
6111       \ifnum\bbl@eqnpos=\@ne
6112         \def\@eqnnum{%
6113           \setbox\z@\hbox{\bbl@eqnum}%
6114           \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6115       \else

```



```

6116         \let\@eqnnum\bb1@eqnum
6117     \fi
6118 \fi}
6119 % Hack. YA luatex bug?:
6120 \expandafter\bb1@sreplace\csname] \endcsname{${$}{\eqno\kern.001pt${$}}%
6121 \else % amstex
6122     \ifx\bb1@noamsmath\undefined
6123         \bb1@exp{% Hack to hide maybe undefined conditionals:
6124             \chardef\bb1@eqnpos=0%
6125             \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6126         \ifnum\bb1@eqnpos=\@ne
6127             \let\bb1@ams@lap\hbox
6128         \else
6129             \let\bb1@ams@lap\llap
6130         \fi
6131         \ExplSyntaxOn
6132         \bb1@sreplace\intertext@{\normalbaselines}%
6133         {\normalbaselines
6134             \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6135         \ExplSyntaxOff
6136         \def\bb1@ams@tagbox#1#2#{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6137         \ifx\bb1@ams@lap\hbox % leqno
6138             \def\bb1@ams@flip#1{%
6139                 \hbox to 0.01pt{\hss\hbox to\displaywidth{#{1}\hss}}}%
6140         \else % eqno
6141             \def\bb1@ams@flip#1{%
6142                 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}%
6143         \fi
6144         \def\bb1@ams@preset#1{%
6145             \ifnum\bb1@thetextdir>\z@
6146                 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6147                 \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6148                 \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6149             \fi}%
6150         \ifnum\bb1@eqnpos=\tw@\else
6151             \def\bb1@ams@equation{%
6152                 \ifnum\bb1@thetextdir>\z@
6153                     \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6154                     \chardef\bb1@thetextdir\z@
6155                     \bb1@add\normalfont{\bb1@eqnodir}%
6156                     \ifcase\bb1@eqnpos
6157                         \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6158                     \or
6159                         \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6160                     \fi
6161                 \fi}%
6162             \AddToHook{env/equation/begin}{\bb1@ams@equation}%
6163             \AddToHook{env/equation*/begin}{\bb1@ams@equation}%
6164         \fi
6165         \AddToHook{env/cases/begin}{\bb1@ams@preset\bb1@ams@lap}%
6166         \AddToHook{env/multline/begin}{\bb1@ams@preset\hbox}%
6167         \AddToHook{env/gather/begin}{\bb1@ams@preset\bb1@ams@lap}%
6168         \AddToHook{env/gather*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6169         \AddToHook{env/align/begin}{\bb1@ams@preset\bb1@ams@lap}%
6170         \AddToHook{env/align*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6171         \AddToHook{env/eqnalign/begin}{\bb1@ams@preset\hbox}%
6172         % Hackish, for proper alignment. Don't ask me why it works!:
6173         \bb1@exp{% Avoid a 'visible' conditional
6174             \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6175         \AddToHook{env/flalign/begin}{\bb1@ams@preset\hbox}%
6176         \AddToHook{env/split/before}{%
6177             \ifnum\bb1@thetextdir>\z@
6178                 \bb1@ifsamestring\@currentenv{equation}%

```

```

6179         {\ifx\bbbl@ams@lap\hbox % leqno
6180         \def\bbbl@ams@flip#1{%
6181             \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6182         \else
6183             \def\bbbl@ams@flip#1{%
6184                 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#{1}}}}%
6185             \fi}%
6186         }%
6187     \fi}%
6188 \fi
6189 \fi}
6190 \fi
6191 \def\bbbl@provide@extra#1{%
6192     % == Counters: mapdigits ==
6193     % Native digits
6194     \ifx\bbbl@KVP@mapdigits\@nnil\else
6195         \bbbl@ifunset{\bbbl@dgnat\language\language\language}%
6196         {\RequirePackage{luatexbase}%
6197          \bbbl@activate@preotf
6198          \directlua{
6199              Babel = Babel or {} %%% -> presets in luababel
6200              Babel.digits_mapped = true
6201              Babel.digits = Babel.digits or {}
6202              Babel.digits[\the\localeid] =
6203                  table.pack(string.utfvalue('\bbbl@cl{dgnat}'))
6204              if not Babel.numbers then
6205                  function Babel.numbers(head)
6206                      local LOCALE = Babel.attr_locale
6207                      local GLYPH = node.id'glyph'
6208                      local inmath = false
6209                      for item in node.traverse(head) do
6210                          if not inmath and item.id == GLYPH then
6211                              local temp = node.get_attribute(item, LOCALE)
6212                              if Babel.digits[temp] then
6213                                  local chr = item.char
6214                                  if chr > 47 and chr < 58 then
6215                                      item.char = Babel.digits[temp][chr-47]
6216                                  end
6217                              end
6218                          elseif item.id == node.id'math' then
6219                              inmath = (item.subtype == 0)
6220                          end
6221                      end
6222                      return head
6223                  end
6224              end
6225          }}%
6226     \fi
6227     % == transforms ==
6228     \ifx\bbbl@KVP@transforms\@nnil\else
6229         \def\bbbl@elt##1##2##3{%
6230             \in@{${transforms.}}{##1}%
6231             \ifin@
6232                 \def\bbbl@tempa{##1}%
6233                 \bbbl@replace\bbbl@tempa{transforms.}%}%
6234                 \bbbl@carg\bbbl@transforms{babel\bbbl@tempa}{##2}{##3}%
6235             \fi}%
6236         \csname bbl@inidata@\language\language\language\endcsname
6237         \bbbl@release@transforms\relax % \relax closes the last item.
6238     \fi}
6239 \ifx\bbbl@opt@layout\@nnil\endinput\fi % if no layout
6240 %
6241 \ifnum\bbbl@bidimode>\z@

```

```

6242 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6243 \bbl@exp{%
6244 \def\\bbl@insidemath{0}%
6245 \mathdir\the\bodydir
6246 #1% Once entered in math, set boxes to restore values
6247 \<ifmmode>%
6248 \everyvbox{%
6249 \the\everyvbox
6250 \bodydir\the\bodydir
6251 \mathdir\the\mathdir
6252 \everyhbox{\the\everyhbox}%
6253 \everyvbox{\the\everyvbox}}%
6254 \everyhbox{%
6255 \the\everyhbox
6256 \bodydir\the\bodydir
6257 \mathdir\the\mathdir
6258 \everyhbox{\the\everyhbox}%
6259 \everyvbox{\the\everyvbox}}%
6260 \<fi>}}%
6261 \def\@hangfrom#1{%
6262 \setbox\@tempboxa\hbox{#1}%
6263 \hangindent\wd\@tempboxa
6264 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6265 \shapemode\@ne
6266 \fi
6267 \noindent\box\@tempboxa}
6268 \fi
6269 \IfBabelLayout{tabular}
6270 {\let\bbl@OL@tabular\@tabular
6271 \bbl@replace\@tabular{$}\{\bbl@nextfake$}%
6272 \let\bbl@NL@tabular\@tabular
6273 \AtBeginDocument{%
6274 \ifx\bbl@NL@tabular\@tabular\else
6275 \bbl@replace\@tabular{$}\{\bbl@nextfake$}%
6276 \let\bbl@NL@tabular\@tabular
6277 \fi}}
6278 {}
6279 \IfBabelLayout{lists}
6280 {\let\bbl@OL@list\list
6281 \bbl@sreplace\list{\parshape}\{\bbl@listparshape}%
6282 \let\bbl@NL@list\list
6283 \def\bbl@listparshape#1#2#3{%
6284 \parshape #1 #2 #3 %
6285 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6286 \shapemode\tw@
6287 \fi}}
6288 {}
6289 \IfBabelLayout{graphics}
6290 {\let\bbl@pictresetdir\relax
6291 \def\bbl@pictsetdir#1{%
6292 \ifcase\bbl@thetextdir
6293 \let\bbl@pictresetdir\relax
6294 \else
6295 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6296 \or\textdir TLT
6297 \else\bodydir TLT \textdir TLT
6298 \fi
6299 % \(\text|par)dir required in pgf:
6300 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6301 \fi}%
6302 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6303 \directlua{
6304 Babel.get_picture_dir = true

```

```

6305 Babel.picture_has_bidi = 0
6306 %
6307 function Babel.picture_dir (head)
6308   if not Babel.get_picture_dir then return head end
6309   if Babel.hlist_has_bidi(head) then
6310     Babel.picture_has_bidi = 1
6311   end
6312   return head
6313 end
6314 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6315   "Babel.picture_dir")
6316 }%
6317 \AtBeginDocument{%
6318   \def\LS@rot{%
6319     \setbox\@outputbox\vbox{%
6320       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6321   \long\def\put(#1,#2)#3{%
6322     \@killglue
6323     % Try:
6324     \ifx\bbl@pictresetdir\relax
6325       \def\bbl@tempc{0}%
6326     \else
6327       \directlua{
6328         Babel.get_picture_dir = true
6329         Babel.picture_has_bidi = 0
6330       }%
6331       \setbox\z@\hb@xt@\z@{%
6332         \@defaultunitsset\@tempdimc{#1}\unitlength
6333         \kern\@tempdimc
6334         #3\hss}% TODO: #3 executed twice (below). That's bad.
6335       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6336       \fi
6337       % Do:
6338       \@defaultunitsset\@tempdimc{#2}\unitlength
6339       \raise\@tempdimc\hb@xt@\z@{%
6340         \@defaultunitsset\@tempdimc{#1}\unitlength
6341         \kern\@tempdimc
6342         {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6343       \ignorespaces}%
6344     \MakeRobust\put}%
6345 \AtBeginDocument
6346   {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6347   \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6348     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6349     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6350     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6351   \fi
6352   \ifx\tikzpicture\@undefined\else
6353     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6354     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6355     \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6356   \fi
6357   \ifx\tcolorbox\@undefined\else
6358     \def\tcb@drawing@env@begin{%
6359       \csname tcb@before@tcb@split@state\endcsname
6360       \bbl@pictsetdir\tw@
6361       \begin{\kvtcb@graphenv}%
6362       \tcb@bbdraw%
6363       \tcb@apply@graph@patches
6364     }%
6365     \def\tcb@drawing@env@end{%
6366       \end{\kvtcb@graphenv}%
6367     \bbl@pictresetdir

```

```

6368      \csname tcb@after@\tcb@split@state\endcsname
6369      }%
6370      \fi
6371    }}
6372  {}

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.

6373 \IfBabelLayout{counters*}%
6374   {\bbl@add\bbl@opt@layout{.counters.}%
6375    \directlua{
6376      luatexbase.add_to_callback("process_output_buffer",
6377        Babel.discard_subl_r , "Babel.discard_subl_r") }%
6378   }%
6379 \IfBabelLayout{counters}%
6380   {\let\bbl@OL@textsuperscript\@textsuperscript
6381    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6382    \let\bbl@latin@arabic=\@arabic
6383    \let\bbl@OL@arabic=\@arabic
6384    \def\@arabic#1{\babelsubl_r{\bbl@latin@arabic#1}}%
6385    \@ifpackagewith{babel}{bidi=default}%
6386    {\let\bbl@asciroman=\@roman
6387     \let\bbl@OL@roman\@roman
6388     \def\@roman#1{\babelsubl_r{\ensureascii{\bbl@asciroman#1}}}%
6389     \let\bbl@asciRoman=\@Roman
6390     \let\bbl@OL@roman\@Roman
6391     \def\@Roman#1{\babelsubl_r{\ensureascii{\bbl@asciRoman#1}}}%
6392     \let\bbl@OL@labelenumii\labelenumii
6393     \def\labelenumii{\theenumii}%
6394     \let\bbl@OL@p@enumiii\p@enumiii
6395     \def\p@enumiii{\p@enumii}\theenumii{}}}%
6396   <<Footnote changes>>
6397 \IfBabelLayout{footnotes}%
6398   {\let\bbl@OL@footnote\footnote
6399    \BabelFootnote\footnote\language{}{}}%
6400    \BabelFootnote\localfootnote\language{}{}}%
6401    \BabelFootnote\mainfootnote{}{}}}%
6402   {}

```

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.

```

6403 \IfBabelLayout{extras}%
6404   {\let\bbl@OL@underline\underline
6405    \bbl@sreplace\underline{\$}\@underline{\bbl@nextfake$\@underline}%
6406    \let\bbl@OL@LaTeX2e\LaTeX2e
6407    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6408      \if b\expandafter\car\series\@nil\boldmath\fi
6409      \babelsubl_r{
6410        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6411   }%
6412 </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.

```

6413 (*transforms)
6414 Babel.linebreaking.replacements = {}
6415 Babel.linebreaking.replacements[0] = {} -- pre
6416 Babel.linebreaking.replacements[1] = {} -- post
6417 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6418
6419 -- Discretionaries contain strings as nodes
6420 function Babel.str_to_nodes(fn, matches, base)
6421   local n, head, last
6422   if fn == nil then return nil end
6423   for s in string.utfvalues(fn(matches)) do
6424     if base.id == 7 then
6425       base = base.replace
6426     end
6427     n = node.copy(base)
6428     n.char = s
6429     if not head then
6430       head = n
6431     else
6432       last.next = n
6433     end
6434     last = n
6435   end
6436   return head
6437 end
6438
6439 Babel.fetch_subtext = {}
6440
6441 Babel.ignore_pre_char = function(node)
6442   return (node.lang == Babel.nohyphenation)
6443 end
6444
6445 -- Merging both functions doesn't seem feasible, because there are too
6446 -- many differences.
6447 Babel.fetch_subtext[0] = function(head)
6448   local word_string = ''
6449   local word_nodes = {}
6450   local lang
6451   local item = head
6452   local inmath = false
6453
6454   while item do
6455
6456     if item.id == 11 then
6457       inmath = (item.subtype == 0)
6458     end
6459
6460     if inmath then
6461       -- pass
6462     elseif item.id == 29 then
6463       local locale = node.get_attribute(item, Babel.attr_locale)
6464
6465       if lang == locale or lang == nil then
6466         lang = lang or locale
6467         if Babel.ignore_pre_char(item) then
6468           word_string = word_string .. Babel.us_char
6469         else
6470           word_string = word_string .. unicode.utf8.char(item.char)
6471         end
6472       end

```

```

6473         word_nodes[#word_nodes+1] = item
6474     else
6475         break
6476     end
6477
6478 elseif item.id == 12 and item.subtype == 13 then
6479     word_string = word_string .. ' '
6480     word_nodes[#word_nodes+1] = item
6481
6482 -- Ignore leading unrecognized nodes, too.
6483 elseif word_string ~= '' then
6484     word_string = word_string .. Babel.us_char
6485     word_nodes[#word_nodes+1] = item -- Will be ignored
6486 end
6487
6488 item = item.next
6489 end
6490
6491 -- Here and above we remove some trailing chars but not the
6492 -- corresponding nodes. But they aren't accessed.
6493 if word_string:sub(-1) == ' ' then
6494     word_string = word_string:sub(1,-2)
6495 end
6496 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6497 return word_string, word_nodes, item, lang
6498 end
6499
6500 Babel.fetch_subtext[1] = function(head)
6501     local word_string = ''
6502     local word_nodes = {}
6503     local lang
6504     local item = head
6505     local inmath = false
6506
6507     while item do
6508
6509         if item.id == 11 then
6510             inmath = (item.subtype == 0)
6511         end
6512
6513         if inmath then
6514             -- pass
6515
6516         elseif item.id == 29 then
6517             if item.lang == lang or lang == nil then
6518                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6519                     lang = lang or item.lang
6520                     word_string = word_string .. unicode.utf8.char(item.char)
6521                     word_nodes[#word_nodes+1] = item
6522                 end
6523             else
6524                 break
6525             end
6526
6527         elseif item.id == 7 and item.subtype == 2 then
6528             word_string = word_string .. '='
6529             word_nodes[#word_nodes+1] = item
6530
6531         elseif item.id == 7 and item.subtype == 3 then
6532             word_string = word_string .. '|'
6533             word_nodes[#word_nodes+1] = item
6534
6535         -- (1) Go to next word if nothing was found, and (2) implicitly

```

```

6536 -- remove leading USs.
6537 elseif word_string == '' then
6538     -- pass
6539
6540 -- This is the responsible for splitting by words.
6541 elseif (item.id == 12 and item.subtype == 13) then
6542     break
6543
6544 else
6545     word_string = word_string .. Babel.us_char
6546     word_nodes[#word_nodes+1] = item -- Will be ignored
6547 end
6548
6549 item = item.next
6550 end
6551
6552 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6553 return word_string, word_nodes, item, lang
6554 end
6555
6556 function Babel.pre_hyphenate_replace(head)
6557     Babel.hyphenate_replace(head, 0)
6558 end
6559
6560 function Babel.post_hyphenate_replace(head)
6561     Babel.hyphenate_replace(head, 1)
6562 end
6563
6564 Babel.us_char = string.char(31)
6565
6566 function Babel.hyphenate_replace(head, mode)
6567     local u = unicode.utf8
6568     local lbkr = Babel.linebreaking.replacements[mode]
6569     if mode == 2 then mode = 0 end -- WIP
6570
6571     local word_head = head
6572
6573     while true do -- for each subtext block
6574
6575         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6576
6577         if Babel.debug then
6578             print()
6579             print((mode == 0) and '####<' or '####>', w)
6580         end
6581
6582         if nw == nil and w == '' then break end
6583
6584         if not lang then goto next end
6585         if not lbkr[lang] then goto next end
6586
6587         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6588         -- loops are nested.
6589         for k=1, #lbkr[lang] do
6590             local p = lbkr[lang][k].pattern
6591             local r = lbkr[lang][k].replace
6592             local attr = lbkr[lang][k].attr or -1
6593
6594             if Babel.debug then
6595                 print('*****', p, mode)
6596             end
6597
6598             -- This variable is set in some cases below to the first *byte*

```



```

6599 -- after the match, either as found by u.match (faster) or the
6600 -- computed position based on sc if w has changed.
6601 local last_match = 0
6602 local step = 0
6603
6604 -- For every match.
6605 while true do
6606     if Babel.debug then
6607         print('====')
6608     end
6609     local new -- used when inserting and removing nodes
6610
6611     local matches = { u.match(w, p, last_match) }
6612
6613     if #matches < 2 then break end
6614
6615     -- Get and remove empty captures (with ()'s, which return a
6616     -- number with the position), and keep actual captures
6617     -- (from (...)), if any, in matches.
6618     local first = table.remove(matches, 1)
6619     local last = table.remove(matches, #matches)
6620     -- Non re-fetched substrings may contain \31, which separates
6621     -- subsubstrings.
6622     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6623
6624     local save_last = last -- with A()BC()D, points to D
6625
6626     -- Fix offsets, from bytes to unicode. Explained above.
6627     first = u.len(w:sub(1, first-1)) + 1
6628     last = u.len(w:sub(1, last-1)) -- now last points to C
6629
6630     -- This loop stores in a small table the nodes
6631     -- corresponding to the pattern. Used by 'data' to provide a
6632     -- predictable behavior with 'insert' (w_nodes is modified on
6633     -- the fly), and also access to 'remove'd nodes.
6634     local sc = first-1 -- Used below, too
6635     local data_nodes = {}
6636
6637     local enabled = true
6638     for q = 1, last-first+1 do
6639         data_nodes[q] = w_nodes[sc+q]
6640         if enabled
6641             and attr > -1
6642             and not node.has_attribute(data_nodes[q], attr)
6643         then
6644             enabled = false
6645         end
6646     end
6647
6648     -- This loop traverses the matched substring and takes the
6649     -- corresponding action stored in the replacement list.
6650     -- sc = the position in substr nodes / string
6651     -- rc = the replacement table index
6652     local rc = 0
6653
6654     while rc < last-first+1 do -- for each replacement
6655         if Babel.debug then
6656             print('.....', rc + 1)
6657         end
6658         sc = sc + 1
6659         rc = rc + 1
6660
6661         if Babel.debug then

```

```

6662         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6663         local ss = ''
6664         for itt in node.traverse(head) do
6665             if itt.id == 29 then
6666                 ss = ss .. unicode.utf8.char(itt.char)
6667             else
6668                 ss = ss .. '{' .. itt.id .. '}'
6669             end
6670         end
6671         print('*****', ss)
6672     end
6673
6674     local crep = r[rc]
6675     local item = w_nodes[sc]
6676     local item_base = item
6677     local placeholder = Babel.us_char
6678     local d
6679
6680
6681     if crep and crep.data then
6682         item_base = data_nodes[crep.data]
6683     end
6684
6685     if crep then
6686         step = crep.step or 0
6687     end
6688
6689     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6690         last_match = save_last    -- Optimization
6691         goto next
6692     end
6693
6694     elseif crep == nil or crep.remove then
6695         node.remove(head, item)
6696         table.remove(w_nodes, sc)
6697         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6698         sc = sc - 1    -- Nothing has been inserted.
6699         last_match = utf8.offset(w, sc+1+step)
6700         goto next
6701
6702     elseif crep and crep.kashida then -- Experimental
6703         node.set_attribute(item,
6704             Babel.attr_kashida,
6705             crep.kashida)
6706         last_match = utf8.offset(w, sc+1+step)
6707         goto next
6708
6709     elseif crep and crep.string then
6710         local str = crep.string(matches)
6711         if str == '' then -- Gather with nil
6712             node.remove(head, item)
6713             table.remove(w_nodes, sc)
6714             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6715             sc = sc - 1    -- Nothing has been inserted.
6716         else
6717             local loop_first = true
6718             for s in string.utfvalues(str) do
6719                 d = node.copy(item_base)
6720                 d.char = s
6721                 if loop_first then
6722                     loop_first = false
6723                     head, new = node.insert_before(head, item, d)
6724                     if sc == 1 then
6725                         word_head = head
6726                     end
6727                 end
6728             end
6729         end
6730     end
6731
6732     goto next
6733 end

```

```

6725         end
6726         w_nodes[sc] = d
6727         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6728     else
6729         sc = sc + 1
6730         head, new = node.insert_before(head, item, d)
6731         table.insert(w_nodes, sc, new)
6732         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6733     end
6734     if Babel.debug then
6735         print('.....', 'str')
6736         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6737     end
6738     end -- for
6739     node.remove(head, item)
6740 end -- if ''
6741 last_match = utf8.offset(w, sc+1+step)
6742 goto next
6743
6744 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6745     d = node.new(7, 0) -- (disc, discretionary)
6746     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6747     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6748     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6749     d.attr = item_base.attr
6750     if crep.pre == nil then -- TeXbook p96
6751         d.penalty = crep.penalty or tex.hyphenpenalty
6752     else
6753         d.penalty = crep.penalty or tex.exhyphenpenalty
6754     end
6755     placeholder = '|'
6756     head, new = node.insert_before(head, item, d)
6757
6758 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6759     -- ERROR
6760
6761 elseif crep and crep.penalty then
6762     d = node.new(14, 0) -- (penalty, userpenalty)
6763     d.attr = item_base.attr
6764     d.penalty = crep.penalty
6765     head, new = node.insert_before(head, item, d)
6766
6767 elseif crep and crep.space then
6768     -- 655360 = 10 pt = 10 * 65536 sp
6769     d = node.new(12, 13) -- (glue, spaceskip)
6770     local quad = font.getfont(item_base.font).size or 655360
6771     node.setglue(d, crep.space[1] * quad,
6772                  crep.space[2] * quad,
6773                  crep.space[3] * quad)
6774     if mode == 0 then
6775         placeholder = ' '
6776     end
6777     head, new = node.insert_before(head, item, d)
6778
6779 elseif crep and crep.spacefactor then
6780     d = node.new(12, 13) -- (glue, spaceskip)
6781     local base_font = font.getfont(item_base.font)
6782     node.setglue(d,
6783                  crep.spacefactor[1] * base_font.parameters['space'],
6784                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6785                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6786     if mode == 0 then
6787         placeholder = ' '

```

```

6788         end
6789         head, new = node.insert_before(head, item, d)
6790
6791     elseif mode == 0 and crep and crep.space then
6792         -- ERROR
6793
6794     end -- ie replacement cases
6795
6796     -- Shared by disc, space and penalty.
6797     if sc == 1 then
6798         word_head = head
6799     end
6800     if crep.insert then
6801         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6802         table.insert(w_nodes, sc, new)
6803         last = last + 1
6804     else
6805         w_nodes[sc] = d
6806         node.remove(head, item)
6807         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6808     end
6809
6810     last_match = utf8.offset(w, sc+1+step)
6811
6812     ::next::
6813
6814 end -- for each replacement
6815
6816 if Babel.debug then
6817     print('.....', '/')
6818     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6819 end
6820
6821 end -- for match
6822
6823 end -- for patterns
6824
6825 ::next::
6826 word_head = nw
6827 end -- for substring
6828 return head
6829 end
6830
6831 -- This table stores capture maps, numbered consecutively
6832 Babel.capture_maps = {}
6833
6834 -- The following functions belong to the next macro
6835 function Babel.capture_func(key, cap)
6836     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
6837     local cnt
6838     local u = unicode.utf8
6839     ret, cnt = ret:gsub('{{([0-9])|([^\]|+)|(.-)}}', Babel.capture_func_map)
6840     if cnt == 0 then
6841         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
6842             function (n)
6843                 return u.char(tonumber(n, 16))
6844             end)
6845     end
6846     ret = ret:gsub("%[%[%]]%.", '')
6847     ret = ret:gsub("%.%[%[%]]%", '')
6848     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6849 end
6850

```

```

6851 function Babel.capt_map(from, mapno)
6852   return Babel.capture_maps[mapno][from] or from
6853 end
6854
6855 -- Handle the {n|abc|ABC} syntax in captures
6856 function Babel.capture_func_map(capno, from, to)
6857   local u = unicode.utf8
6858   from = u.gsub(from, '{(%x%x%x%x+)}',
6859     function (n)
6860       return u.char(tonumber(n, 16))
6861     end)
6862   to = u.gsub(to, '{(%x%x%x%x+)}',
6863     function (n)
6864       return u.char(tonumber(n, 16))
6865     end)
6866   local froms = {}
6867   for s in string.utfcharacters(from) do
6868     table.insert(froms, s)
6869   end
6870   local cnt = 1
6871   table.insert(Babel.capture_maps, {})
6872   local mlen = table.getn(Babel.capture_maps)
6873   for s in string.utfcharacters(to) do
6874     Babel.capture_maps[mlen][froms[cnt]] = s
6875     cnt = cnt + 1
6876   end
6877   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6878     (mlen) .. ").." .. "["
6879 end
6880
6881 -- Create/Extend reversed sorted list of kashida weights:
6882 function Babel.capture_kashida(key, wt)
6883   wt = tonumber(wt)
6884   if Babel.kashida_wts then
6885     for p, q in ipairs(Babel.kashida_wts) do
6886       if wt == q then
6887         break
6888       elseif wt > q then
6889         table.insert(Babel.kashida_wts, p, wt)
6890         break
6891       elseif table.getn(Babel.kashida_wts) == p then
6892         table.insert(Babel.kashida_wts, wt)
6893       end
6894     end
6895   else
6896     Babel.kashida_wts = { wt }
6897   end
6898   return 'kashida = ' .. wt
6899 end
6900 </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.

```
6901 <*basic-r>
6902 Babel = Babel or {}
6903
6904 Babel.bidi_enabled = true
6905
6906 require('babel-data-bidi.lua')
6907
6908 local characters = Babel.characters
6909 local ranges = Babel.ranges
6910
6911 local DIR = node.id("dir")
6912
6913 local function dir_mark(head, from, to, outer)
6914   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6915   local d = node.new(DIR)
6916   d.dir = '+' .. dir
6917   node.insert_before(head, from, d)
6918   d = node.new(DIR)
6919   d.dir = '-' .. dir
6920   node.insert_after(head, to, d)
6921 end
6922
6923 function Babel.bidi(head, ispar)
6924   local first_n, last_n          -- first and last char with nums
6925   local last_es                  -- an auxiliary 'last' used with nums
6926   local first_d, last_d          -- first and last char in L/R block
6927   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):

```
6928   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6929   local strong_lr = (strong == 'l') and 'l' or 'r'
6930   local outer = strong
6931
6932   local new_dir = false
```

```

6933 local first_dir = false
6934 local inmath = false
6935
6936 local last_lr
6937
6938 local type_n = ''
6939
6940 for item in node.traverse(head) do
6941
6942   -- three cases: glyph, dir, otherwise
6943   if item.id == node.id'glyph'
6944     or (item.id == 7 and item.subtype == 2) then
6945
6946     local itemchar
6947     if item.id == 7 and item.subtype == 2 then
6948       itemchar = item.replace.char
6949     else
6950       itemchar = item.char
6951     end
6952     local chardata = characters[itemchar]
6953     dir = chardata and chardata.d or nil
6954     if not dir then
6955       for nn, et in ipairs(ranges) do
6956         if itemchar < et[1] then
6957           break
6958         elseif itemchar <= et[2] then
6959           dir = et[3]
6960           break
6961         end
6962       end
6963     end
6964     dir = dir or 'l'
6965     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).

```

6966   if new_dir then
6967     attr_dir = 0
6968     for at in node.traverse(item.attr) do
6969       if at.number == Babel.attr_dir then
6970         attr_dir = at.value % 3
6971       end
6972     end
6973     if attr_dir == 1 then
6974       strong = 'r'
6975     elseif attr_dir == 2 then
6976       strong = 'al'
6977     else
6978       strong = 'l'
6979     end
6980     strong_lr = (strong == 'l') and 'l' or 'r'
6981     outer = strong_lr
6982     new_dir = false
6983   end
6984
6985   if dir == 'nsm' then dir = strong end -- W1

```

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

```

6986   dir_real = dir -- We need dir_real to set strong below
6987   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:

```

6988     if strong == 'al' then
6989         if dir == 'en' then dir = 'an' end           -- W2
6990         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6991         strong_lr = 'r'                               -- W3
6992     end

```

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

```

6993     elseif item.id == node.id'dir' and not inmath then
6994         new_dir = true
6995         dir = nil
6996     elseif item.id == node.id'math' then
6997         inmath = (item.subtype == 0)
6998     else
6999         dir = nil           -- Not a char
7000     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>.

```

7001     if dir == 'en' or dir == 'an' or dir == 'et' then
7002         if dir ~= 'et' then
7003             type_n = dir
7004         end
7005         first_n = first_n or item
7006         last_n = last_es or item
7007         last_es = nil
7008     elseif dir == 'es' and last_n then -- W3+W6
7009         last_es = item
7010     elseif dir == 'cs' then           -- it's right - do nothing
7011     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7012         if strong_lr == 'r' and type_n ~= '' then
7013             dir_mark(head, first_n, last_n, 'r')
7014         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7015             dir_mark(head, first_n, last_n, 'r')
7016             dir_mark(head, first_d, last_d, outer)
7017             first_d, last_d = nil, nil
7018         elseif strong_lr == 'l' and type_n ~= '' then
7019             last_d = last_n
7020         end
7021         type_n = ''
7022         first_n, last_n = nil, nil
7023     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:

```

7024     if dir == 'l' or dir == 'r' then
7025         if dir ~= outer then
7026             first_d = first_d or item
7027             last_d = item
7028         elseif first_d and dir ~= strong_lr then
7029             dir_mark(head, first_d, last_d, outer)
7030             first_d, last_d = nil, nil
7031         end
7032     end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all

these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

7033   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7034       item.char = characters[item.char] and
7035           characters[item.char].m or item.char
7036   elseif (dir or new_dir) and last_lr ~= item then
7037       local mir = outer .. strong_lr .. (dir or outer)
7038       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7039           for ch in node.traverse(node.next(last_lr)) do
7040               if ch == item then break end
7041               if ch.id == node.id'glyph' and characters[ch.char] then
7042                   ch.char = characters[ch.char].m or ch.char
7043               end
7044           end
7045       end
7046   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).

```

7047   if dir == 'l' or dir == 'r' then
7048       last_lr = item
7049       strong = dir_real          -- Don't search back - best save now
7050       strong_lr = (strong == 'l') and 'l' or 'r'
7051   elseif new_dir then
7052       last_lr = nil
7053   end
7054 end

```

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

```

7055   if last_lr and outer == 'r' then
7056       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7057           if characters[ch.char] then
7058               ch.char = characters[ch.char].m or ch.char
7059           end
7060       end
7061   end
7062   if first_n then
7063       dir_mark(head, first_n, last_n, outer)
7064   end
7065   if first_d then
7066       dir_mark(head, first_d, last_d, outer)
7067   end

```

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

```

7068   return node.prev(head) or head
7069 end
7070 </basic-r>

```

And here the Lua code for bidi=basic:

```

7071 <*basic>
7072 Babel = Babel or {}
7073
7074 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7075
7076 Babel.fontmap = Babel.fontmap or {}
7077 Babel.fontmap[0] = {}      -- l
7078 Babel.fontmap[1] = {}      -- r
7079 Babel.fontmap[2] = {}      -- al/an
7080
7081 Babel.bidi_enabled = true
7082 Babel.mirroring_enabled = true

```

```

7083
7084 require('babel-data-bidi.lua')
7085
7086 local characters = Babel.characters
7087 local ranges = Babel.ranges
7088
7089 local DIR = node.id('dir')
7090 local GLYPH = node.id('glyph')
7091
7092 local function insert_implicit(head, state, outer)
7093   local new_state = state
7094   if state.sim and state.eim and state.sim ~= state.eim then
7095     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7096     local d = node.new(DIR)
7097     d.dir = '+' .. dir
7098     node.insert_before(head, state.sim, d)
7099     local d = node.new(DIR)
7100     d.dir = '-' .. dir
7101     node.insert_after(head, state.eim, d)
7102   end
7103   new_state.sim, new_state.eim = nil, nil
7104   return head, new_state
7105 end
7106
7107 local function insert_numeric(head, state)
7108   local new
7109   local new_state = state
7110   if state.san and state.ean and state.san ~= state.ean then
7111     local d = node.new(DIR)
7112     d.dir = '+TLT'
7113     _, new = node.insert_before(head, state.san, d)
7114     if state.san == state.sim then state.sim = new end
7115     local d = node.new(DIR)
7116     d.dir = '-TLT'
7117     _, new = node.insert_after(head, state.ean, d)
7118     if state.ean == state.eim then state.eim = new end
7119   end
7120   new_state.san, new_state.ean = nil, nil
7121   return head, new_state
7122 end
7123
7124 -- TODO - \hbox with an explicit dir can lead to wrong results
7125 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7126 -- was s made to improve the situation, but the problem is the 3-dir
7127 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7128 -- well.
7129
7130 function Babel.bidi(head, ispar, hdir)
7131   local d -- d is used mainly for computations in a loop
7132   local prev_d = ''
7133   local new_d = false
7134
7135   local nodes = {}
7136   local outer_first = nil
7137   local inmath = false
7138
7139   local glue_d = nil
7140   local glue_i = nil
7141
7142   local has_en = false
7143   local first_et = nil
7144
7145   local has_hyperlink = false

```

```

7146
7147 local ATDIR = Babel.attr_dir
7148
7149 local save_outer
7150 local temp = node.get_attribute(head, ATDIR)
7151 if temp then
7152     temp = temp % 3
7153     save_outer = (temp == 0 and 'l') or
7154                  (temp == 1 and 'r') or
7155                  (temp == 2 and 'al')
7156 elseif ispar then -- Or error? Shouldn't happen
7157     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7158 else -- Or error? Shouldn't happen
7159     save_outer = ('TRT' == hdir) and 'r' or 'l'
7160 end
7161 -- when the callback is called, we are just _after_ the box,
7162 -- and the textdir is that of the surrounding text
7163 -- if not ispar and hdir ~= tex.textdir then
7164 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7165 -- end
7166 local outer = save_outer
7167 local last = outer
7168 -- 'al' is only taken into account in the first, current loop
7169 if save_outer == 'al' then save_outer = 'r' end
7170
7171 local fontmap = Babel.fontmap
7172
7173 for item in node.traverse(head) do
7174
7175     -- In what follows, #node is the last (previous) node, because the
7176     -- current one is not added until we start processing the neutrals.
7177
7178     -- three cases: glyph, dir, otherwise
7179     if item.id == GLYPH
7180         or (item.id == 7 and item.subtype == 2) then
7181
7182         local d_font = nil
7183         local item_r
7184         if item.id == 7 and item.subtype == 2 then
7185             item_r = item.replace -- automatic discs have just 1 glyph
7186         else
7187             item_r = item
7188         end
7189         local chardata = characters[item_r.char]
7190         d = chardata and chardata.d or nil
7191         if not d or d == 'nsm' then
7192             for nn, et in ipairs(ranges) do
7193                 if item_r.char < et[1] then
7194                     break
7195                 elseif item_r.char <= et[2] then
7196                     if not d then d = et[3]
7197                     elseif d == 'nsm' then d_font = et[3]
7198                     end
7199                     break
7200             end
7201         end
7202         end
7203         d = d or 'l'
7204
7205         -- A short 'pause' in bidi for mapfont
7206         d_font = d_font or d
7207         d_font = (d_font == 'l' and 0) or
7208                 (d_font == 'nsm' and 0) or

```

```

7209             (d_font == 'r' and 1) or
7210             (d_font == 'al' and 2) or
7211             (d_font == 'an' and 2) or nil
7212     if d_font and fontmap and fontmap[d_font][item_r.font] then
7213         item_r.font = fontmap[d_font][item_r.font]
7214     end
7215
7216     if new_d then
7217         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7218         if inmath then
7219             attr_d = 0
7220         else
7221             attr_d = node.get_attribute(item, ATDIR)
7222             attr_d = attr_d % 3
7223         end
7224         if attr_d == 1 then
7225             outer_first = 'r'
7226             last = 'r'
7227         elseif attr_d == 2 then
7228             outer_first = 'r'
7229             last = 'al'
7230         else
7231             outer_first = 'l'
7232             last = 'l'
7233         end
7234         outer = last
7235         has_en = false
7236         first_et = nil
7237         new_d = false
7238     end
7239
7240     if glue_d then
7241         if (d == 'l' and 'l' or 'r') ~= glue_d then
7242             table.insert(nodes, {glue_i, 'on', nil})
7243         end
7244         glue_d = nil
7245         glue_i = nil
7246     end
7247
7248     elseif item.id == DIR then
7249         d = nil
7250         if head ~= item then new_d = true end
7251
7252     elseif item.id == node.id'glue' and item.subtype == 13 then
7253         glue_d = d
7254         glue_i = item
7255         d = nil
7256
7257     elseif item.id == node.id'math' then
7258         inmath = (item.subtype == 0)
7259
7260     elseif item.id == 8 and item.subtype == 19 then
7261         has_hyperlink = true
7262
7263     else
7264         d = nil
7265     end
7266
7267     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7268     if last == 'al' and d == 'en' then
7269         d = 'an'          -- W3
7270     elseif last == 'al' and (d == 'et' or d == 'es') then
7271         d = 'on'          -- W6

```

```

7272 end
7273
7274 -- EN + CS/ES + EN      -- W4
7275 if d == 'en' and #nodes >= 2 then
7276   if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7277     and nodes[#nodes-1][2] == 'en' then
7278     nodes[#nodes][2] = 'en'
7279   end
7280 end
7281
7282 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7283 if d == 'an' and #nodes >= 2 then
7284   if (nodes[#nodes][2] == 'cs')
7285     and nodes[#nodes-1][2] == 'an' then
7286     nodes[#nodes][2] = 'an'
7287   end
7288 end
7289
7290 -- ET/EN                  -- W5 + W7->l / W6->on
7291 if d == 'et' then
7292   first_et = first_et or (#nodes + 1)
7293 elseif d == 'en' then
7294   has_en = true
7295   first_et = first_et or (#nodes + 1)
7296 elseif first_et then      -- d may be nil here !
7297   if has_en then
7298     if last == 'l' then
7299       temp = 'l'      -- W7
7300     else
7301       temp = 'en'     -- W5
7302     end
7303   else
7304     temp = 'on'       -- W6
7305   end
7306   for e = first_et, #nodes do
7307     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7308   end
7309   first_et = nil
7310   has_en = false
7311 end
7312
7313 -- Force mathdir in math if ON (currently works as expected only
7314 -- with 'l')
7315 if inmath and d == 'on' then
7316   d = ('TRT' == tex.mathdir) and 'r' or 'l'
7317 end
7318
7319 if d then
7320   if d == 'al' then
7321     d = 'r'
7322     last = 'al'
7323   elseif d == 'l' or d == 'r' then
7324     last = d
7325   end
7326   prev_d = d
7327   table.insert(nodes, {item, d, outer_first})
7328 end
7329
7330 outer_first = nil
7331
7332 end
7333
7334 -- TODO -- repeated here in case EN/ET is the last node. Find a

```

```

7335 -- better way of doing things:
7336 if first_et then      -- dir may be nil here !
7337     if has_en then
7338         if last == 'l' then
7339             temp = 'l'    -- W7
7340         else
7341             temp = 'en'    -- W5
7342         end
7343     else
7344         temp = 'on'        -- W6
7345     end
7346     for e = first_et, #nodes do
7347         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7348     end
7349 end
7350
7351 -- dummy node, to close things
7352 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7353
7354 ----- NEUTRAL -----
7355
7356 outer = save_outer
7357 last = outer
7358
7359 local first_on = nil
7360
7361 for q = 1, #nodes do
7362     local item
7363
7364     local outer_first = nodes[q][3]
7365     outer = outer_first or outer
7366     last = outer_first or last
7367
7368     local d = nodes[q][2]
7369     if d == 'an' or d == 'en' then d = 'r' end
7370     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7371
7372     if d == 'on' then
7373         first_on = first_on or q
7374     elseif first_on then
7375         if last == d then
7376             temp = d
7377         else
7378             temp = outer
7379         end
7380         for r = first_on, q - 1 do
7381             nodes[r][2] = temp
7382             item = nodes[r][1]    -- MIRRORING
7383             if Babel.mirroring_enabled and item.id == GLYPH
7384                 and temp == 'r' and characters[item.char] then
7385                 local font_mode = ''
7386                 if item.font > 0 and font.fonts[item.font].properties then
7387                     font_mode = font.fonts[item.font].properties.mode
7388                 end
7389                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7390                     item.char = characters[item.char].m or item.char
7391                 end
7392             end
7393         end
7394         first_on = nil
7395     end
7396
7397     if d == 'r' or d == 'l' then last = d end

```

```

7398 end
7399
7400 ----- IMPLICIT, REORDER -----
7401
7402 outer = save_outer
7403 last = outer
7404
7405 local state = {}
7406 state.has_r = false
7407
7408 for q = 1, #nodes do
7409
7410     local item = nodes[q][1]
7411
7412     outer = nodes[q][3] or outer
7413
7414     local d = nodes[q][2]
7415
7416     if d == 'nsm' then d = last end          -- W1
7417     if d == 'en' then d = 'an' end
7418     local isdir = (d == 'r' or d == 'l')
7419
7420     if outer == 'l' and d == 'an' then
7421         state.san = state.san or item
7422         state.ean = item
7423     elseif state.san then
7424         head, state = insert_numeric(head, state)
7425     end
7426
7427     if outer == 'l' then
7428         if d == 'an' or d == 'r' then      -- im -> implicit
7429             if d == 'r' then state.has_r = true end
7430             state.sim = state.sim or item
7431             state.eim = item
7432         elseif d == 'l' and state.sim and state.has_r then
7433             head, state = insert_implicit(head, state, outer)
7434         elseif d == 'l' then
7435             state.sim, state.eim, state.has_r = nil, nil, false
7436         end
7437     else
7438         if d == 'an' or d == 'l' then
7439             if nodes[q][3] then -- nil except after an explicit dir
7440                 state.sim = item -- so we move sim 'inside' the group
7441             else
7442                 state.sim = state.sim or item
7443             end
7444             state.eim = item
7445         elseif d == 'r' and state.sim then
7446             head, state = insert_implicit(head, state, outer)
7447         elseif d == 'r' then
7448             state.sim, state.eim = nil, nil
7449         end
7450     end
7451
7452     if isdir then
7453         last = d          -- Don't search back - best save now
7454     elseif d == 'on' and state.san then
7455         state.san = state.san or item
7456         state.ean = item
7457     end
7458
7459 end
7460

```

```

7461 head = node.prev(head) or head
7462
7463 ----- FIX HYPERLINKS -----
7464
7465 if has_hyperlink then
7466     local flag, linking = 0, 0
7467     for item in node.traverse(head) do
7468         if item.id == DIR then
7469             if item.dir == '+TRT' or item.dir == '+TLT' then
7470                 flag = flag + 1
7471             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7472                 flag = flag - 1
7473             end
7474             elseif item.id == 8 and item.subtype == 19 then
7475                 linking = flag
7476             elseif item.id == 8 and item.subtype == 20 then
7477                 if linking > 0 then
7478                     if item.prev.id == DIR and
7479                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7480                         d = node.new(DIR)
7481                         d.dir = item.prev.dir
7482                         node.remove(head, item.prev)
7483                         node.insert_after(head, item, d)
7484                     end
7485                 end
7486                 linking = 0
7487             end
7488         end
7489     end
7490
7491 return head
7492 end
7493 </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.

```

7494 <*nil>
7495 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7496 \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.

```

7497 \ifx\l@nil\@undefined

```



```

7498 \newlanguage\l@nil
7499 \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
7500 \let\bbl@elt\relax
7501 \edef\bbl@languages{% Add it to the list of languages
7502 \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
7503 \fi

```

This macro is used to store the values of the hyphenation parameters `\leftthyphenmin` and `\rightthyphenmin`.

```
7504 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```

\captionnil
\datenil
7505 \let\captionsnil\@empty
7506 \let\datenil\@empty

```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```

7507 \def\bbl@inidata@nil{%
7508 \bbl@elt{identification}{tag.ini}{und}%
7509 \bbl@elt{identification}{load.level}{0}%
7510 \bbl@elt{identification}{charset}{utf8}%
7511 \bbl@elt{identification}{version}{1.0}%
7512 \bbl@elt{identification}{date}{2022-05-16}%
7513 \bbl@elt{identification}{name.local}{nil}%
7514 \bbl@elt{identification}{name.english}{nil}%
7515 \bbl@elt{identification}{name.babel}{nil}%
7516 \bbl@elt{identification}{tag.bcp47}{und}%
7517 \bbl@elt{identification}{language.tag.bcp47}{und}%
7518 \bbl@elt{identification}{tag.opentype}{dflt}%
7519 \bbl@elt{identification}{script.name}{Latin}%
7520 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7521 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7522 \bbl@elt{identification}{level}{1}%
7523 \bbl@elt{identification}{encodings}{}%
7524 \bbl@elt{identification}{derivate}{no}}
7525 \@namedef{bbl@tbc@nil}{und}
7526 \@namedef{bbl@lbc@nil}{und}
7527 \@namedef{bbl@lotf@nil}{dflt}
7528 \@namedef{bbl@elname@nil}{nil}
7529 \@namedef{bbl@lname@nil}{nil}
7530 \@namedef{bbl@esname@nil}{Latin}
7531 \@namedef{bbl@sname@nil}{Latin}
7532 \@namedef{bbl@sbcp@nil}{Latn}
7533 \@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.

```

7534 \ldf@finish{nil}
7535 \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.

```

7536 \langle *Compute Julian day \rangle \equiv
7537 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7538 \def\bbl@cs@gregleap#1{%
7539 (\bbl@fpmo{#1}{4} == 0) &&
7540 (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7541 \def\bbl@cs@jd#1#2#3{% year, month, day

```

```

7542 \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7543 floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7544 floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7545 ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7546 <</Compute Julian day>>

```

15.1 Islamic

The code for the Civil calendar is based on it, too.

```

7547 <*ca-islamic>
7548 \ExplSyntaxOn
7549 <<Compute Julian day>>
7550 % == islamic (default)
7551 % Not yet implemented
7552 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

7553 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7554 ((#3 + ceil(29.5 * (#2 - 1)) +
7555 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7556 1948439.5) - 1) }
7557 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+2}}
7558 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7559 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7560 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7561 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7562 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7563 \edef\bbl@tempa{%
7564 \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7565 \edef#5{%
7566 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7567 \edef#6{\fp_eval:n{
7568 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7569 \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).

```

7570 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7571 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7572 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7573 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7574 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7575 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7576 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7577 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7578 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7579 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7580 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7581 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7582 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7583 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7584 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7585 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7586 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7587 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7588 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7589 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7590 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7591 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7592 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7593 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%

```

```

7594 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7595 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7596 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7597 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7598 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7599 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7600 65401,65431,65460,65490,65520}
7601 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7602 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7603 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7604 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7605   \ifnum#2>2014 \ifnum#2<2038
7606     \bbl@afterfi\expandafter\@gobble
7607   \fi\fi
7608   {\bbl@error{Year~out-of-range}{The~allowed~range-is-2014-2038}}%
7609   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7610     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7611   \count@\@ne
7612   \bbl@foreach\bbl@cs@umalqura@data{%
7613     \advance\count@\@ne
7614     \ifnum##1>\bbl@tempd\else
7615       \edef\bbl@tempe{\the\count@}%
7616       \edef\bbl@tempb{##1}%
7617     \fi}%
7618   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
7619   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7620   \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7621   \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7622   \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7623 \ExplSyntaxOff
7624 \bbl@add\bbl@precalendar{%
7625   \bbl@replace\bbl@ld@calendar{-civil}{}%
7626   \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7627   \bbl@replace\bbl@ld@calendar{+}{}%
7628   \bbl@replace\bbl@ld@calendar{-}{}}
7629 </ca-islamic>

```

16 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty

```

7630 <*ca-hebrew>
7631 \newcount\bbl@cntcommon
7632 \def\bbl@remainder#1#2#3{%
7633   #3=#1\relax
7634   \divide #3 by #2\relax
7635   \multiply #3 by -#2\relax
7636   \advance #3 by #1\relax}%
7637 \newif\ifbbl@divisible
7638 \def\bbl@checkifdivisible#1#2{%
7639   {\countdef\tmp=0
7640     \bbl@remainder{#1}{#2}{\tmp}%
7641     \ifnum \tmp=0
7642       \global\bbl@divisibletrue
7643     \else
7644       \global\bbl@divisiblefalse
7645     \fi}}
7646 \newif\ifbbl@gregleap
7647 \def\bbl@ifgregleap#1{%
7648   \bbl@checkifdivisible{#1}{4}%
7649   \ifbbl@divisible

```

```

7650     \bbl@checkifdivisible{#1}{100}%
7651     \ifbbl@divisible
7652         \bbl@checkifdivisible{#1}{400}%
7653         \ifbbl@divisible
7654             \bbl@gregleaptrue
7655         \else
7656             \bbl@gregleapfalse
7657         \fi
7658     \else
7659         \bbl@gregleaptrue
7660     \fi
7661 \else
7662     \bbl@gregleapfalse
7663 \fi
7664 \ifbbl@gregleap}
7665 \def\bbl@gregdayspriormonths#1#2#3{%
7666     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7667         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7668     \bbl@ifgregleap{#2}%
7669     \ifnum #1 > 2
7670         \advance #3 by 1
7671     \fi
7672 \fi
7673 \global\bbl@cntcommon=#3}%
7674 #3=\bbl@cntcommon}
7675 \def\bbl@gregdaysprioryears#1#2{%
7676     {\countdef\tmpc=4
7677     \countdef\tmpb=2
7678     \tmpb=#1\relax
7679     \advance \tmpb by -1
7680     \tmpc=\tmpb
7681     \multiply \tmpc by 365
7682     #2=\tmpc
7683     \tmpc=\tmpb
7684     \divide \tmpc by 4
7685     \advance #2 by \tmpc
7686     \tmpc=\tmpb
7687     \divide \tmpc by 100
7688     \advance #2 by -\tmpc
7689     \tmpc=\tmpb
7690     \divide \tmpc by 400
7691     \advance #2 by \tmpc
7692     \global\bbl@cntcommon=#2\relax}%
7693 #2=\bbl@cntcommon}
7694 \def\bbl@absfromgreg#1#2#3#4{%
7695     {\countdef\tmpd=0
7696     #4=#1\relax
7697     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7698     \advance #4 by \tmpd
7699     \bbl@gregdaysprioryears{#3}{\tmpd}%
7700     \advance #4 by \tmpd
7701     \global\bbl@cntcommon=#4\relax}%
7702 #4=\bbl@cntcommon}
7703 \newif\ifbbl@hebrleap
7704 \def\bbl@checkleaphebryear#1{%
7705     {\countdef\tmpa=0
7706     \countdef\tmpb=1
7707     \tmpa=#1\relax
7708     \multiply \tmpa by 7
7709     \advance \tmpa by 1
7710     \bbl@remainder{\tmpa}{19}{\tmpb}%
7711     \ifnum \tmpb < 7
7712         \global\bbl@hebrleaptrue

```

```

7713 \else
7714 \global\bbl@hebrleapfalse
7715 \fi}}
7716 \def\bbl@hebreleapsedmonths#1#2{%
7717 {\countdef\tmpa=0
7718 \countdef\tmpb=1
7719 \countdef\tmpc=2
7720 \tmpa=#1\relax
7721 \advance \tmpa by -1
7722 #2=\tmpa
7723 \divide #2 by 19
7724 \multiply #2 by 235
7725 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7726 \tmpc=\tmpb
7727 \multiply \tmpb by 12
7728 \advance #2 by \tmpb
7729 \multiply \tmpc by 7
7730 \advance \tmpc by 1
7731 \divide \tmpc by 19
7732 \advance #2 by \tmpc
7733 \global\bbl@cntcommon=#2}%
7734 #2=\bbl@cntcommon}
7735 \def\bbl@hebreleapseddays#1#2{%
7736 {\countdef\tmpa=0
7737 \countdef\tmpb=1
7738 \countdef\tmpc=2
7739 \bbl@hebreleapsedmonths{#1}{#2}%
7740 \tmpa=#2\relax
7741 \multiply \tmpa by 13753
7742 \advance \tmpa by 5604
7743 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7744 \divide \tmpa by 25920
7745 \multiply #2 by 29
7746 \advance #2 by 1
7747 \advance #2 by \tmpa
7748 \bbl@remainder{#2}{7}{\tmpa}%
7749 \ifnum \tmpc < 19440
7750 \ifnum \tmpc < 9924
7751 \else
7752 \ifnum \tmpa=2
7753 \bbl@checkleaphebrewyear{#1}% of a common year
7754 \ifbbl@hebrleap
7755 \else
7756 \advance #2 by 1
7757 \fi
7758 \fi
7759 \fi
7760 \ifnum \tmpc < 16789
7761 \else
7762 \ifnum \tmpa=1
7763 \advance #1 by -1
7764 \bbl@checkleaphebrewyear{#1}% at the end of leap year
7765 \ifbbl@hebrleap
7766 \advance #2 by 1
7767 \fi
7768 \fi
7769 \fi
7770 \else
7771 \advance #2 by 1
7772 \fi
7773 \bbl@remainder{#2}{7}{\tmpa}%
7774 \ifnum \tmpa=0
7775 \advance #2 by 1

```

```

7776 \else
7777 \ifnum \tmpa=3
7778 \advance #2 by 1
7779 \else
7780 \ifnum \tmpa=5
7781 \advance #2 by 1
7782 \fi
7783 \fi
7784 \fi
7785 \global\bbl@cntcommon=#2\relax}%
7786 #2=\bbl@cntcommon}
7787 \def\bbl@daysinhebrewyear#1#2{%
7788 {\countdef\tmpe=12
7789 \bbl@hebreleapseddays{#1}{\tmpe}%
7790 \advance #1 by 1
7791 \bbl@hebreleapseddays{#1}{#2}%
7792 \advance #2 by -\tmpe
7793 \global\bbl@cntcommon=#2}%
7794 #2=\bbl@cntcommon}
7795 \def\bbl@hebrdayspriormonths#1#2#3{%
7796 {\countdef\tmpf= 14
7797 #3=\ifcase #1\relax
7798 0 \or
7799 0 \or
7800 30 \or
7801 59 \or
7802 89 \or
7803 118 \or
7804 148 \or
7805 148 \or
7806 177 \or
7807 207 \or
7808 236 \or
7809 266 \or
7810 295 \or
7811 325 \or
7812 400
7813 \fi
7814 \bbl@checkleaphebrewyear{#2}%
7815 \ifbbl@hebrleap
7816 \ifnum #1 > 6
7817 \advance #3 by 30
7818 \fi
7819 \fi
7820 \bbl@daysinhebrewyear{#2}{\tmpf}%
7821 \ifnum #1 > 3
7822 \ifnum \tmpf=353
7823 \advance #3 by -1
7824 \fi
7825 \ifnum \tmpf=383
7826 \advance #3 by -1
7827 \fi
7828 \fi
7829 \ifnum #1 > 2
7830 \ifnum \tmpf=355
7831 \advance #3 by 1
7832 \fi
7833 \ifnum \tmpf=385
7834 \advance #3 by 1
7835 \fi
7836 \fi
7837 \global\bbl@cntcommon=#3\relax}%
7838 #3=\bbl@cntcommon}

```

```

7839 \def\bbl@absfromhebr#1#2#3#4{%
7840   {#4=#1\relax
7841     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7842     \advance #4 by #1\relax
7843     \bbl@hebreleapseddays{#3}{#1}%
7844     \advance #4 by #1\relax
7845     \advance #4 by -1373429
7846     \global\bbl@cntcommon=#4\relax}%
7847   #4=\bbl@cntcommon}
7848 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7849   {\countdef\tmpx= 17
7850     \countdef\tmpy= 18
7851     \countdef\tmpz= 19
7852     #6=#3\relax
7853     \global\advance #6 by 3761
7854     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7855     \tmpz=1 \tmpy=1
7856     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7857     \ifnum \tmpx > #4\relax
7858       \global\advance #6 by -1
7859       \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7860     \fi
7861     \advance #4 by -\tmpx
7862     \advance #4 by 1
7863     #5=#4\relax
7864     \divide #5 by 30
7865     \loop
7866       \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7867       \ifnum \tmpx < #4\relax
7868         \advance #5 by 1
7869         \tmpy=\tmpx
7870       \repeat
7871       \global\advance #5 by -1
7872       \global\advance #4 by -\tmpy}}
7873 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7874 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7875 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7876   \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7877   \bbl@hebrfromgreg
7878     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7879   {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7880   \edef#4{\the\bbl@hebryear}%
7881   \edef#5{\the\bbl@hebrmonth}%
7882   \edef#6{\the\bbl@hebrday}}
7883 \</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 LPPPL 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).

```

7884 \<ca-persian>
7885 \ExplSyntaxOn
7886 \<Compute Julian day>
7887 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7888   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7889 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
7890   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
7891   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7892     \bbl@afterfi\expandafter\@gobble

```

```

7893 \fi\fi
7894 {\bbl@error{Year~out-of-range}{The~allowed~range-is-2013-2050}}%
7895 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7896 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7897 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7898 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7899 \ifnum\bbl@tempc<\bbl@tempb
7900 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7901 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7902 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7903 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7904 \fi
7905 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7906 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7907 \edef#5{\fp_eval:n{% set Jalali month
7908 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7909 \edef#6{\fp_eval:n{% set Jalali day
7910 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}%
7911 \ExplSyntaxOff
7912 </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.

```

7913 <*ca-coptic>
7914 \ExplSyntaxOn
7915 <<Compute Julian day>>
7916 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
7917 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7918 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
7919 \edef#4{\fp_eval:n{%
7920 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7921 \edef\bbl@tempc{\fp_eval:n{%
7922 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7923 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7924 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7925 \ExplSyntaxOff
7926 </ca-coptic>
7927 <*ca-ethiopic>
7928 \ExplSyntaxOn
7929 <<Compute Julian day>>
7930 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
7931 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7932 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
7933 \edef#4{\fp_eval:n{%
7934 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7935 \edef\bbl@tempc{\fp_eval:n{%
7936 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
7937 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7938 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7939 \ExplSyntaxOff
7940 </ca-ethiopic>

```

19 Buddhist

That's very simple.

```

7941 <*ca-buddhist>
7942 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
7943 \edef#4{\number\numexpr#1+543\relax}%
7944 \edef#5{#2}%

```



```

7945 \edef#6{#3}}
7946 </ca-buddhist>

```

20 Support for Plain T_FX (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 `localhyphen.tex` or whatever they like, but they mustn't diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `inlTeX`, 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 \LaTeX sees, we need to set some category codes just to be able to change the definition of `\input`.

```

7947 (*bplain | bplain)
7948 \catcode`\{=1 % left brace is begin-group character
7949 \catcode`\}=2 % right brace is end-group character
7950 \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).

```

7951 \openin 0 hyphen.cfg
7952 \ifeof0
7953 \else
7954   \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.

```

7955 \def\input #1 {%
7956     \let\input\@
7957     \a hyphen.cfg
7958     \let\@undefined
7959 }
7960 \fi
7961 </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`.

```

7962 <bplain>\a plain.tex
7963 <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.

```

7964 <bplain>\def\fmtname{babel-plain}
7965 <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).

```
7966 <(*Emulate LaTeX)> ≡
7967 \def\@empty{}
7968 \def\loadlocalcfg#1{%
7969   \openin0#1.cfg
7970   \ifeof0
7971     \closein0
7972   \else
7973     \closein0
7974     {\immediate\write16{*****}%
7975      \immediate\write16{* Local config file #1.cfg used}%
7976      \immediate\write16{*}%
7977     }
7978     \input #1.cfg\relax
7979   \fi
7980   \@endofldf}
```

20.3 General tools

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

```
7981 \long\def\@firstofone#1{#1}
7982 \long\def\@firstoftwo#1#2{#1}
7983 \long\def\@secondoftwo#1#2{#2}
7984 \def\@nnil{\@nil}
7985 \def\@gobbletwo#1#2{}
7986 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7987 \def\@star@or@long#1{%
7988   \@ifstar
7989   {\let\@ngrel@x\relax#1}%
7990   {\let\@ngrel@x\long#1}}
7991 \let\@ngrel@x\relax
7992 \def\@car#1#2\@nil{#1}
7993 \def\@cdr#1#2\@nil{#2}
7994 \let\@typeset@protect\relax
7995 \let\protected@edef\edef
7996 \long\def\@gobble#1{}
7997 \edef\@backslashchar{\expandafter\@gobble\string\}
7998 \def\strip@prefix#1>{}
7999 \def\g@addto@macro#1#2{%
8000   \toks@\expandafter{#1#2}%
8001   \xdef#1{\the\toks@}}
8002 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8003 \def\@nameuse#1{\csname #1\endcsname}
8004 \def\@ifundefined#1{%
8005   \expandafter\ifx\csname#1\endcsname\relax
8006     \expandafter\@firstoftwo
8007   \else
8008     \expandafter\@secondoftwo
8009   \fi}
8010 \def\@expandtwoargs#1#2#3{%
8011   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8012 \def\zap@space#1 #2{%
8013   #1%
8014   \ifx#2\@empty\else\expandafter\zap@space\fi
8015   #2}
8016 \let\bbl@trace\@gobble
8017 \def\bbl@error#1#2{%
```

```

8018 \begingroup
8019 \newlinechar=`^^J
8020 \def\{^^J(babel) }%
8021 \errhelp{#2}\errmessage{\#1}%
8022 \endgroup}
8023 \def\bbl@warning#1{%
8024 \begingroup
8025 \newlinechar=`^^J
8026 \def\{^^J(babel) }%
8027 \message{\#1}%
8028 \endgroup}
8029 \let\bbl@infowarn\bbl@warning
8030 \def\bbl@info#1{%
8031 \begingroup
8032 \newlinechar=`^^J
8033 \def\{^^J}%
8034 \wlog{#1}%
8035 \endgroup}

```

$\TeX 2\epsilon$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

8036 \ifx\@preamblecmds\undefined
8037 \def\@preamblecmds{}
8038 \fi
8039 \def\@onlypreamble#1{%
8040 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8041 \@preamblecmds\do#1}}
8042 \@onlypreamble\@onlypreamble

```

Mimick \TeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

8043 \def\begindocument{%
8044 \@begindocumenthook
8045 \global\let\@begindocumenthook\undefined
8046 \def\do##1{\global\let##1\undefined}%
8047 \@preamblecmds
8048 \global\let\do\noexpand}
8049 \ifx\@begindocumenthook\undefined
8050 \def\@begindocumenthook{}
8051 \fi
8052 \@onlypreamble\@begindocumenthook
8053 \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 `\@endoflfd`.

```

8054 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8055 \@onlypreamble\AtEndOfPackage
8056 \def\@endoflfd{}
8057 \@onlypreamble\@endoflfd
8058 \let\bbl@afterlang\empty
8059 \chardef\bbl@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.

```

8060 \catcode`\&=\z@
8061 \ifx&\if@files\undefined
8062 \expandafter\let\csname if@files\expandafter\endcsname
8063 \csname iffalse\endcsname
8064 \fi
8065 \catcode`\&=4

```

Mimick \TeX 's commands to define control sequences.

```

8066 \def\newcommand{\@star@or@long\new@command}

```

```

8067 \def\new@command#1{%
8068   \@testopt{\@newcommand#1}0}
8069 \def\@newcommand#1[#2]{%
8070   \@ifnextchar [{\@xargdef#1[#2]}%
8071                 {\@argdef#1[#2]}}
8072 \long\def\@argdef#1[#2]#3{%
8073   \@yargdef#1\@ne{#2}{#3}}
8074 \long\def\@xargdef#1[#2][#3]#4{%
8075   \expandafter\def\expandafter#1\expandafter{%
8076     \expandafter\@protected@testopt\expandafter #1%
8077     \csname\string#1\expandafter\endcsname{#3}}%
8078   \expandafter\@yargdef \csname\string#1\endcsname
8079   \tw@{#2}{#4}}
8080 \long\def\@yargdef#1#2#3{%
8081   \@tempcnta#3\relax
8082   \advance \@tempcnta \@ne
8083   \let\@hash@\relax
8084   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8085   \@tempcntb #2%
8086   \@whilenum\@tempcntb <\@tempcnta
8087   \do{%
8088     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8089     \advance\@tempcntb \@ne}%
8090   \let\@hash@##%
8091   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8092 \def\providecommand{\@star@or@long\provide@command}
8093 \def\provide@command#1{%
8094   \begingroup
8095     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
8096   \endgroup
8097   \expandafter\@ifundefined\@gtempa
8098     {\def\reserved@a{\new@command#1}}%
8099     {\let\reserved@a\relax
8100     \def\reserved@a{\new@command\reserved@a}}%
8101   \reserved@a}%

8102 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8103 \def\declare@robustcommand#1{%
8104   \edef\reserved@a{\string#1}%
8105   \def\reserved@b{#1}%
8106   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8107   \edef#1{%
8108     \ifx\reserved@a\reserved@b
8109       \noexpand\x@protect
8110       \noexpand#1%
8111     \fi
8112     \noexpand\protect
8113     \expandafter\@gobble\string#1 \endcsname
8114   }%
8115   \expandafter\new@command\csname
8116     \expandafter\@gobble\string#1 \endcsname
8117 }
8118 }
8119 \def\x@protect#1{%
8120   \ifx\protect\@typeset@protect\else
8121     \@x@protect#1%
8122   \fi
8123 }
8124 \catcode\&=\z@ % Trick to hide conditionals
8125 \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`.

```

8126 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8127 \catcode`\&=4
8128 \ifx\in@\undefined
8129 \def\in@#1#2{%
8130 \def\in@@##1##2##3\in@{%
8131 \ifx\in@@##2\in@false\else\in@true\fi}%
8132 \in@@#2#1\in@\in@@}
8133 \else
8134 \let\bbl@tempa\empty
8135 \fi
8136 \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).

```

8137 \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.

```

8138 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\LaTeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

8139 \ifx\@tempcnta\undefined
8140 \csname newcount\endcsname\@tempcnta\relax
8141 \fi
8142 \ifx\@tempcntb\undefined
8143 \csname newcount\endcsname\@tempcntb\relax
8144 \fi

```

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

```

8145 \ifx\bye\undefined
8146 \advance\count10 by -2\relax
8147 \fi
8148 \ifx\@ifnextchar\undefined
8149 \def\@ifnextchar#1#2#3{%
8150 \let\reserved@d=#1%
8151 \def\reserved@a{#2}\def\reserved@b{#3}%
8152 \futurelet\@let@token\@ifnch}
8153 \def\@ifnch{%
8154 \ifx\@let@token\sptoken
8155 \let\reserved@c\@xifnch
8156 \else
8157 \ifx\@let@token\reserved@d
8158 \let\reserved@c\reserved@a
8159 \else
8160 \let\reserved@c\reserved@b
8161 \fi
8162 \fi
8163 \reserved@c}
8164 \def\:{\let\sptoken= }\: % this makes \sptoken a space token
8165 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8166 \fi
8167 \def\@testopt#1#2{%
8168 \@ifnextchar[#{1}{#1[#2]}}
8169 \def\@protected@testopt#1{%
8170 \ifx\protect\@typeset@protect
8171 \expandafter\@testopt
8172 \else
8173 \@x@protect#1%

```

```

8174 \fi}
8175 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@whilenum{#1\relax
8176 #2\relax}\fi}
8177 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8178 \else\expandafter\@gobble\fi{#1}}

```

20.4 Encoding related macros

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

```

8179 \def\DeclareTextCommand{%
8180   \@dec@text@cmd\providecommand
8181 }
8182 \def\ProvideTextCommand{%
8183   \@dec@text@cmd\providecommand
8184 }
8185 \def\DeclareTextSymbol#1#2#3{%
8186   \@dec@text@cmd\chardef#1{#2}#3\relax
8187 }
8188 \def\@dec@text@cmd#1#2#3{%
8189   \expandafter\def\expandafter#2%
8190     \expandafter{%
8191       \csname#3-cmd\expandafter\endcsname
8192       \expandafter#2%
8193       \csname#3\string#2\endcsname
8194     }%
8195 %   \let\@ifdefinable\@rc@ifdefinable
8196   \expandafter#1\csname#3\string#2\endcsname
8197 }
8198 \def\@current@cmd#1{%
8199   \ifx\protect\@typeset@protect\else
8200     \noexpand#1\expandafter\@gobble
8201   \fi
8202 }
8203 \def\@changed@cmd#1#2{%
8204   \ifx\protect\@typeset@protect
8205     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8206       \expandafter\ifx\csname ?\string#1\endcsname\relax
8207         \expandafter\def\csname ?\string#1\endcsname{%
8208           \@changed@x@err{#1}%
8209         }%
8210       \fi
8211       \global\expandafter\let
8212         \csname\cf@encoding\string#1\expandafter\endcsname
8213         \csname ?\string#1\endcsname
8214     \fi
8215     \csname\cf@encoding\string#1%
8216     \expandafter\endcsname
8217   \else
8218     \noexpand#1%
8219   \fi
8220 }
8221 \def\@changed@x@err#1{%
8222   \errhelp{Your command will be ignored, type <return> to proceed}%
8223   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8224 \def\DeclareTextCommandDefault#1{%
8225   \DeclareTextCommand#1?%
8226 }
8227 \def\ProvideTextCommandDefault#1{%
8228   \ProvideTextCommand#1?%
8229 }
8230 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8231 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8232 \def\DeclareTextAccent#1#2#3{%

```

```

8233 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8234 }
8235 \def\DeclareTextCompositeCommand#1#2#3#4{%
8236   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8237   \edef\reserved@b{\string##1}%
8238   \edef\reserved@c{%
8239     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8240   \ifx\reserved@b\reserved@c
8241     \expandafter\expandafter\expandafter\ifx
8242       \expandafter\@car\reserved@a\relax\relax\@nil
8243       \@text@composite
8244     \else
8245       \edef\reserved@b##1{%
8246         \def\expandafter\noexpand
8247           \csname#2\string#1\endcsname###1{%
8248             \noexpand\@text@composite
8249             \expandafter\noexpand\csname#2\string#1\endcsname
8250             ###1\noexpand\@empty\noexpand\@text@composite
8251             {##1}%
8252           }%
8253         }%
8254       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8255     \fi
8256     \expandafter\def\csname\expandafter\string\csname
8257       #2\endcsname\string#1-\string#3\endcsname{#4}
8258   \else
8259     \errhelp{Your command will be ignored, type <return> to proceed}%
8260     \errmessage{\string\DeclareTextCompositeCommand\space used on
8261       inappropriate command \protect#1}
8262   \fi
8263 }
8264 \def\@text@composite#1#2#3\@text@composite{%
8265   \expandafter\@text@composite@x
8266     \csname\string#1-\string#2\endcsname
8267 }
8268 \def\@text@composite@x#1#2{%
8269   \ifx#1\relax
8270     #2%
8271   \else
8272     #1%
8273   \fi
8274 }
8275 %
8276 \def\@strip@args#1:#2-#3\@strip@args{#2}
8277 \def\DeclareTextComposite#1#2#3#4{%
8278   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8279   \bgroup
8280     \lccode`\@=#4%
8281     \lowercase{%
8282       \egroup
8283       \reserved@a @%
8284     }%
8285 }
8286 %
8287 \def\UseTextSymbol#1#2{#2}
8288 \def\UseTextAccent#1#2#3{}
8289 \def\@use@text@encoding#1{}
8290 \def\DeclareTextSymbolDefault#1#2{%
8291   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8292 }
8293 \def\DeclareTextAccentDefault#1#2{%
8294   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8295 }

```

```
8296 \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.

```
8297 \DeclareTextAccent{"}{OT1}{127}
8298 \DeclareTextAccent{'}{OT1}{19}
8299 \DeclareTextAccent{^}{OT1}{94}
8300 \DeclareTextAccent`}{OT1}{18}
8301 \DeclareTextAccent~}{OT1}{126}
```

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

```
8302 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8303 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
8304 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8305 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8306 \DeclareTextSymbol{\i}{OT1}{16}
8307 \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`.

```
8308 \ifx\scriptsize\@undefined
8309   \let\scriptsize\sevenrm
8310 \fi
```

And a few more “dummy” definitions.

```
8311 \def\languagename{english}%
8312 \let\bbl@opt@shorthands\@nnil
8313 \def\bbl@ifshorthand#1#2#3#2}%
8314 \let\bbl@language@opts\@empty
8315 \ifx\babeloptionstrings\@undefined
8316   \let\bbl@opt@strings\@nnil
8317 \else
8318   \let\bbl@opt@strings\babeloptionstrings
8319 \fi
8320 \def\BabelStringsDefault{generic}
8321 \def\bbl@tempa{normal}
8322 \ifx\babeloptionmath\bbl@tempa
8323   \def\bbl@mathnormal{\noexpand\textormath}
8324 \fi
8325 \def\AfterBabelLanguage#1#2{}
8326 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8327 \let\bbl@afterlang\relax
8328 \def\bbl@opt@safe{BR}
8329 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8330 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8331 \expandafter\newif\csname ifbbl@single\endcsname
8332 \chardef\bbl@bidimode\z@
8333 </Emulate LaTeX>
```

A proxy file:

```
8334 <*plain>
8335 \input babel.def
8336 </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 Tsolomitis 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).