

Babel

Version 3.65.2536
2021/10/25

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

| | | |
|----------|--|-----------|
| I | User guide | 4 |
| 1 | The user interface | 4 |
| 1.1 | Monolingual documents | 4 |
| 1.2 | Multilingual documents | 6 |
| 1.3 | Mostly monolingual documents | 8 |
| 1.4 | Modifiers | 8 |
| 1.5 | Troubleshooting | 8 |
| 1.6 | Plain | 9 |
| 1.7 | Basic language selectors | 9 |
| 1.8 | Auxiliary language selectors | 10 |
| 1.9 | More on selection | 11 |
| 1.10 | Shorthands | 12 |
| 1.11 | Package options | 16 |
| 1.12 | The base option | 18 |
| 1.13 | ini files | 18 |
| 1.14 | Selecting fonts | 26 |
| 1.15 | Modifying a language | 28 |
| 1.16 | Creating a language | 29 |
| 1.17 | Digits and counters | 33 |
| 1.18 | Dates | 34 |
| 1.19 | Accessing language info | 35 |
| 1.20 | Hyphenation and line breaking | 36 |
| 1.21 | Transforms | 38 |
| 1.22 | Selection based on BCP 47 tags | 40 |
| 1.23 | Selecting scripts | 41 |
| 1.24 | Selecting directions | 42 |
| 1.25 | Language attributes | 46 |
| 1.26 | Hooks | 46 |
| 1.27 | Languages supported by babel with ldf files | 47 |
| 1.28 | Unicode character properties in luatex | 49 |
| 1.29 | Tweaking some features | 49 |
| 1.30 | Tips, workarounds, known issues and notes | 49 |
| 1.31 | Current and future work | 51 |
| 1.32 | Tentative and experimental code | 51 |
| 2 | Loading languages with language.dat | 51 |
| 2.1 | Format | 52 |
| 3 | The interface between the core of babel and the language definition files | 52 |
| 3.1 | Guidelines for contributed languages | 54 |
| 3.2 | Basic macros | 54 |
| 3.3 | Skeleton | 55 |
| 3.4 | Support for active characters | 56 |
| 3.5 | Support for saving macro definitions | 57 |
| 3.6 | Support for extending macros | 57 |
| 3.7 | Macros common to a number of languages | 57 |
| 3.8 | Encoding-dependent strings | 58 |
| 4 | Changes | 61 |
| 4.1 | Changes in babel version 3.9 | 61 |

| | | |
|-----------|---|------------|
| II | Source code | 62 |
| 5 | Identification and loading of required files | 62 |
| 6 | locale directory | 62 |
| 7 | Tools | 63 |
| 7.1 | Multiple languages | 67 |
| 7.2 | The Package File (<code>\LaTeX</code> , <code>babel.sty</code>) | 68 |
| 7.3 | <code>base</code> | 69 |
| 7.4 | <code>key=value</code> options and other general option | 70 |
| 7.5 | Conditional loading of shorthands | 72 |
| 7.6 | Interlude for Plain | 73 |
| 8 | Multiple languages | 73 |
| 8.1 | Selecting the language | 76 |
| 8.2 | Errors | 84 |
| 8.3 | Hooks | 87 |
| 8.4 | Setting up language files | 89 |
| 8.5 | Shorthands | 91 |
| 8.6 | Language attributes | 100 |
| 8.7 | Support for saving macro definitions | 102 |
| 8.8 | Short tags | 103 |
| 8.9 | Hyphens | 104 |
| 8.10 | Multiencoding strings | 105 |
| 8.11 | Macros common to a number of languages | 112 |
| 8.12 | Making glyphs available | 112 |
| 8.12.1 | Quotation marks | 112 |
| 8.12.2 | Letters | 114 |
| 8.12.3 | Shorthands for quotation marks | 115 |
| 8.12.4 | Umlauts and tremas | 115 |
| 8.13 | Layout | 117 |
| 8.14 | Load engine specific macros | 117 |
| 8.15 | Creating and modifying languages | 117 |
| 9 | Adjusting the Babel behavior | 139 |
| 9.1 | Cross referencing macros | 141 |
| 9.2 | Marks | 144 |
| 9.3 | Preventing clashes with other packages | 145 |
| 9.3.1 | <code>ifthen</code> | 145 |
| 9.3.2 | <code>varioref</code> | 145 |
| 9.3.3 | <code>hhline</code> | 146 |
| 9.4 | Encoding and fonts | 146 |
| 9.5 | Basic bidi support | 148 |
| 9.6 | Local Language Configuration | 152 |
| 9.7 | Language options | 152 |
| 10 | The kernel of Babel (<code>babel.def</code>, <code>common</code>) | 156 |
| 11 | Loading hyphenation patterns | 156 |
| 12 | Font handling with <code>fontspec</code> | 160 |

| | | |
|-----------|---|------------|
| 13 | Hooks for XeTeX and LuaTeX | 165 |
| 13.1 | XeTeX | 165 |
| 13.2 | Layout | 167 |
| 13.3 | LuaTeX | 168 |
| 13.4 | Southeast Asian scripts | 174 |
| 13.5 | CJK line breaking | 176 |
| 13.6 | Arabic justification | 178 |
| 13.7 | Common stuff | 182 |
| 13.8 | Automatic fonts and ids switching | 182 |
| 13.9 | Bidi | 187 |
| 13.10 | Layout | 189 |
| 13.11 | Lua: transforms | 192 |
| 13.12 | Lua: Auto bidi with basic and basic-r | 201 |
| 14 | Data for CJK | 212 |
| 15 | The ‘nil’ language | 212 |
| 16 | Support for Plain T_EX (plain.def) | 212 |
| 16.1 | Not renaming hyphen.tex | 212 |
| 16.2 | Emulating some L ^A T _E X features | 213 |
| 16.3 | General tools | 214 |
| 16.4 | Encoding related macros | 218 |
| 17 | Acknowledgements | 221 |

Troubleshooting

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

Part I

User guide

What is this document about? This user guide focuses on internationalization and localization with \LaTeX and `pdftex`, `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:

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

EXAMPLE And now a simple monolingual document in Russian (text from the Wikipedia) with `xetex` or `luatex`. Note neither `fontenc` nor `inputenc` are necessary, but the document should be encoded in UTF-8 and a so-called Unicode font must be loaded (in this example `\babelfont` is used, described below).

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

Россия, находящаяся на пересечении множества культур, а также
с учётом многонационального характера её населения, – отличается
высокой степенью этнокультурного многообразия и способностью к
межкультурному диалогу.

\end{document}
```

TROUBLESHOOTING A common source of trouble is a wrong setting of the input encoding. Depending on the \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.

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

You can also set the main language explicitly, but it is discouraged except if there a real reason to do so:

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

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

NOTE Some classes load babel with a hardcoded language option. Sometimes, the main language can be overridden with something like that before `\documentclass`:

```
\PassOptionsToPackage{main=english}{babel}
```

WARNING Languages may be set as global and as package option at the same time, but in such a case you should set explicitly the main language with the package option `main`:

```
\documentclass[italian]{book}  
\usepackage[ngerman,main=italian]{babel}
```

WARNING In the preamble the main language has *not* been selected, except hyphenation patterns and the name assigned to `\language` (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.

To switch the language there are two basic macros, described below in detail:

`\selectlanguage` is used for blocks of text, while `\foreignlanguage` is for chunks of text inside paragraphs.

EXAMPLE A full bilingual document with pdf_{TEX} follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDF_{TEX}

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
  
\usepackage[english,french]{babel}  
  
\begin{document}  
  
Plus ça change, plus c'est la même chose!  
  
\selectlanguage{english}  
  
And an English paragraph, with a short text in  
\foreignlanguage{french}{français}.  
  
\end{document}
```

EXAMPLE With x_{ETEX} and lua_{TEX}, the following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and `\today` in Danish and Vietnamese. No additional packages are required.

LUATEX/XETEX

```
\documentclass{article}  
  
\usepackage[vietnamese,danish]{babel}  
  
\begin{document}  
  
\prefacename{} -- \alsoname{} -- \today  
  
\selectlanguage{vietnamese}  
  
\prefacename{} -- \alsoname{} -- \today  
  
\end{document}
```

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

1.3 Mostly monolingual documents

New 3.39 Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of `\babelfont`, if used.)

This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that `\babelfont` does *not* load any font until required, so that it can be used just in case.

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

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

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

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

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

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

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

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

WARNING If used inside braces there might be some non-local changes, as this would be roughly equivalent to:

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

If you want a change which is really local, you must enclose this code with an additional grouping level.

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

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

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

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

New 3.44 As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

```
\foreignlanguage[date]{polish}{\today}
```

In addition, captions can be switched with `captions` (or both, of course, with `date`, `captions`). Until 3.43 you had to write something like `{\selectlanguage{..} ..}`, which was not always the most convenient way.

1.8 Auxiliary language selectors

`\begin{otherlanguage}` {*<language>*} ... **`\end{otherlanguage}`**

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

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

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

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

Spaces after the environment are ignored.

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

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

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

1.9 More on selection

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

New 3.9i In multilingual documents with many language-switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.

It defines `\text<tag1>{<text>}` to be `\foreignlanguage{<language1>}{<text>}`, and `\begin{<tag1>}` to be `\begin{otherlanguage*}{<language1>}`, and so on. Note `\<tag1>` is also allowed, but remember to set it locally inside a group.

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

EXAMPLE With

```
\babeltags{de = german}
```

you can write

```
text \textde{German text} text
```

and

```
text
\begin{de}
  German text
\end{de}
text
```

NOTE Something like `\babeltags{finnish = finnish}` is legitimate – it defines `\textfinnish` and `\finnish` (and, of course, `\begin{finnish}`).

NOTE Actually, there may be another advantage in the ‘short’ syntax `\text<tag>`, namely, it is not affected by `\MakeUppercase` (while `\foreignlanguage` is).

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

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

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

`\shorthandon` $\{\langle shorthands-list \rangle\}$
`\shorthandoff` $*\{\langle shorthands-list \rangle\}$

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

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

`\shorthandoff*{~^}`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

⁶Thanks to Enrico Gregorio

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

1.11 Package options

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

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

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

activegrave Same for `.

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

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

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

safe= none | ref | bib
Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in $\epsilon\TeX$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

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

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

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

- headfoot=** `<language>`
- By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key config is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** New 3.9l Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** New 3.9l No warnings and no *infos* are written to the log file.⁸
- strings=** `generic` | `unicode` | `encoded` | `<label>` | ``
- Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional \TeX , LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
- hyphenmap=** `off` | `first` | `select` | `other` | `other*`
- New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.⁹ It can take the following values:
- off** deactivates this feature and no case mapping is applied;
- first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at `\begin{document}`}, but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated.¹⁰
- select** sets it only at `\selectlanguage`;
- other** also sets it at `otherlanguage`;
- other*** also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.¹¹
- bidi=** `default` | `basic` | `basic-r` | `bidi-l` | `bidi-r`
- New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.24.
- layout=** New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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

1.12 The base option

With this package option `babel` just loads some basic macros (those in `switch.def`), defines `\AfterBabelLanguage` and exits. It also selects the hyphenation patterns for the last language passed as option (by its name in `language.dat`). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

`\AfterBabelLanguage` $\{\langle option-name \rangle\}\{\langle code \rangle\}$

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

```
\AfterBabelLanguage{french}\dots}
```

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

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

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

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

1.13 ini files

An alternative approach to define a language (or, more precisely, a *locale*) is by means of an `ini` file. Currently `babel` provides about 200 of these files containing the basic data required for a locale.

`ini` files are not meant only for `babel`, and they have been devised as a resource for other packages. To easy interoperability between \TeX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the `\dots` name strings).

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

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

LUATEX/XETEX

```
\documentclass{book}

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

```

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

\begin{document}

\tableofcontents

\chapter{სამხარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამხარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

```

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

```

Or also:

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym

sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq
telugu
teso
thai
tibetan
tigrinya
tongan
turkish
turkmen

| | |
|----------------|----------------|
| ukenglish | vai-latn |
| ukrainian | vai-vai |
| uppersorbian | vai-vaii |
| urdu | vai |
| usenglish | vietnam |
| usorbian | vietnamese |
| uyghur | vunjo |
| uzbek-arab | walser |
| uzbek-arabic | welsh |
| uzbek-cyrillic | westernfrisian |
| uzbek-cyrl | yangben |
| uzbek-latin | yiddish |
| uzbek-latn | yoruba |
| uzbek | zarma |
| vai-latin | zulu afrikaans |

Modifying and adding values to ini files

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

1.14 Selecting fonts

New 3.15 Babel provides a high level interface on top of `fontspec` to select fonts. There is no need to load `fontspec` explicitly – babel does it for you with the first `\babelfont`.¹³

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

TROUBLESHOOTING *Package fontspec Warning: 'Language 'LANG' not available for font 'FONT' with script 'SCRIPT' 'Default' language used instead'.*

This is *not* an error. This warning is shown by `fontspec`, not by `babel`. It can be irrelevant for English, but not for many other languages, including Urdu and Turkish. This is a useful and harmless warning, and if everything is fine with your document the best thing you can do is just to ignore it altogether.

TROUBLESHOOTING *Package babel Info: The following fonts are not babel standard families.*

This is *not* an error. `babel` assumes that if you are using `\babelfont` for a family, very likely you want to define the rest of them. If you don't, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.

Actually, there is no real need to use `\babelfont` in a monolingual document, if you set the language system in `\setmainfont` (or not, depending on what you want).

As the message explains, *there is nothing intrinsically wrong* with not defining all the families. In fact, there is nothing intrinsically wrong with not using `\babelfont` at all. But you must be aware that this may lead to some problems.

1.15 Modifying a language

Modifying the behavior of a language (say, the chapter “caption”), is sometimes necessary, but not always trivial. In the case of caption names a specific macro is provided, because this is perhaps the most frequent change:

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

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

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

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

NOTE There are a few alternative methods:

- With data imported from `ini` files, you can modify the values of specific keys, like:

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

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

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

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

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do so. This redefinition is not activated until the language is selected.

- The ‘new way’, which is found in bulgarian, azerbaijani, spanish, french, turkish, icelandic, vietnamese and a few more, as well as in languages created with \babelprovide and its key import, is:

```
\renewcommand\spanishchaptername{Foo}
```

This redefinition is immediate.

NOTE Do *not* redefine a caption in the following way:

```
\AtBeginDocument{\renewcommand\contentsname{Foo}}
```

The changes may be discarded with a language selector, and the original value restored.

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

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

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

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

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

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

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

1.16 Creating a language

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

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

If the language *⟨language-name⟩* has not been loaded as class or package option and there are no *⟨options⟩*, it creates an “empty” one with some defaults in its internal structure: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3. In either case, caption, date and language system are not defined.

If no ini file is imported with import, *⟨language-name⟩* is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the ini file corresponding to that name; the same applies to OpenType language and script.

Conveniently, some options allow to fill the language, and babel warns you about what to do if there is a missing string. Very likely you will find alerts like that in the log file:

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

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

EXAMPLE If you need a language named arhinish:

```
\usepackage[danish]{babel}
\babelprovide{arhinish}
\setlocalecaption{arhinish}{chapter}{Chapitula}
\setlocalecaption{arhinish}{refname}{Refirenke}
\renewcommand\arhinishhyphenmins{22}
```

EXAMPLE Locales with names based on BCP 47 codes can be created with something like:

```
\babelprovide[import=en-US]{enUS}
```

Note, however, mixing ways to identify locales can lead to problems. For example, is yi the name of the language spoken by the Yi people or is it the code for Yiddish?

The main language is not changed (danish in this example). So, you must add

`\selectlanguage{arhinish}` or other selectors where necessary.

If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

import= *<language-tag>*

New 3.13 Imports data from an ini file, including captions and date (also line breaking rules in newly defined languages). For example:

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

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like `\'` or `\ss`) ones.

New 3.23 It may be used without a value. In such a case, the ini file set in the corresponding `babel-<language>.tex` (where `<language>` is the last argument in `\babelprovide`) is imported. See the list of recognized languages above. So, the previous example can be written:

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

There are about 250 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages may show a warning about the current lack of suitability of some features.

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

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

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

Loads only the strings. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remember there is an alternative syntax for the latter:

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

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

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

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

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

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

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

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

Same for `\Alph`.

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

onchar= `ids` | `fonts`

New 3.38 This option is much like an ‘event’ called when a character belonging to the script of this locale is found (as its name implies, it acts on characters, not on spaces). There are currently two ‘actions’, which can be used at the same time (separated by a space): with `ids` the `\language` and the `\localeid` are set to the values of this locale; with `fonts`, the fonts are changed to those of this locale (as set with `\babelfont`). This option is not compatible with `mapfont`. Characters can be added or modified with `\babelcharproperty`.

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

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

Sets the interword space for the writing system of the language, in em units (so, `0.1 0` is `0em` plus `.1em`). Like `\spaceskip`, the em unit applied is that of the current text (more precisely, the previous glyph). Currently used only in Southeast Asian scripts, like Thai, and CJK.

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

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

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

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

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

mapfont= `direction`

Assigns the font for the writing direction of this language (only with `bidi=basic`). Whenever possible, instead of this option use `onchar`, based on the script, which usually

makes more sense. More precisely, what `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

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

1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

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

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

The styles are:

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

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

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

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

\localeinfo `{\field}`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

There are also some differences with `LaTeX`: (1) the character used is that set for the current font, while in `LaTeX` it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in `LaTeX`, but it can be changed to another value by redefining `\babenullhyphen`; (3) a break after the hyphen is forbidden if preceded by a

glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones. It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of \lccodes's done in \extras<lang> as well as the language-specific encoding (not set in the preamble by default). Multiple \babelhyphenation's are allowed. For example:

```
\babelhyphenation{Wal-hal-la Dar-bhan-ga}
```

Listed words are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.

NOTE Using \babelhyphenation with Southeast Asian scripts is mostly pointless. But with \babelpatterns (below) you may fine-tune line breaking (only luatex). Even if there are no patterns for the language, you can add at least some typical cases.

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

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

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

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

New 3.9m *In luatex only*,¹⁴ adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras<lang> as well as the language-specific encoding (not set in the preamble by default). Multiple \babelpatterns's are allowed.

Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.

New 3.31 (Only luatex.) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (**New 3.32** it is disabled in verbatim mode, or more precisely when the

¹⁴With luatex exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and babel only provides the most basic tools.

hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the intraspace.

New 3.27 Interword spacing for Thai, Lao and Khemer is activated automatically if a language with one of those scripts are loaded with `\babelprovide`. See the sample on the babel repository. With both Unicode engines, spacing is based on the “current” em unit (the size of the previous char in luatex, and the font size set by the last `\selectfont` in xetex).

1.21 Transforms

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

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

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

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

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

| | | |
|--|------------------------------------|---|
| Arabic | <code>transliteration.dad</code> | Applies the transliteration system devised by Yannis Haralambous for dad (simple and T _E X-friendly). Not yet complete, but sufficient for most texts. |
| Croatian | <code>digraphs.ligatures</code> | Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry. |
| Czech, Polish, Portuguese, Slovak, Spanish | <code>hyphen.repeat</code> | Explicit hyphens behave like <code>\babelhyphen{repeat}</code> . |
| Czech, Polish, Slovak | <code>oneletter.nobreak</code> | Converts a space after a non-syllabic preposition or conjunction into a non-breaking space. |
| Greek | <code>diaeresis.hyphen</code> | Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants. |
| Greek | <code>transliteration.omega</code> | The provided combinations are not exactly the same, but this transform follows the syntax of Omega: <code>=</code> for the circumflex, <code>v</code> for digamma, and so on. For better compatibility with Levy’s system, <code>~</code> (as ‘string’) is an alternative to <code>=</code> . <code>'</code> is tonos in Monotonic Greek, but <code>oxia</code> in Polytonic and Ancient Greek. |

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

| | | |
|-----------------|------------------------|---|
| Hindi, Sanskrit | transliteration.hk | The Harvard-Kyoto system to romanize Devanagari. |
| Hindi, Sanskrit | punctuation.space | Inserts a space before the following four characters: !?;. |
| Hungarian | digraphs.hyphen | Hyphenates the long digraphs <i>ccs</i> , <i>ddz</i> , <i>ggy</i> , <i>lly</i> , <i>nny</i> , <i>ssz</i> , <i>tty</i> and <i>zsz</i> as <i>cs-cs</i> , <i>dz-dz</i> , etc. |
| Indic scripts | danda.nobreak | Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu. |
| Arabic, Persian | kashida.plain | Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59. |
| Serbian | transliteration.gajica | (Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj. |

`\babelposthyphenation` `{⟨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 ([îû]), the replacement could be {1|îû|íú}, which maps *î* to *í*, and *û* to *ú*, so that the diaeresis is removed.

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

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

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

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

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

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

1.22 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{
  autoload.bcp47 = on,
```

```

    autoload.bcp47.options = import
}

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}

```

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

`autoload.bcp47` with values on and off.

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

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

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

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

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

1.23 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.¹⁶

Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was `LY1`), and therefore it has been deprecated.¹⁷

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

¹⁷But still defined for backwards compatibility.

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

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

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

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling `bidi` writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

    Most Arabic speakers consider the two varieties to be two registers
    of one language, although the two registers can be referred to in
    Arabic as \textit{fuṣḥā l-‘aṣr} (MSA) and
    \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

NOTE Boxes are “black boxes”. Numbers inside an `\hbox` (for example in a `\ref`) do not know anything about the surrounding chars. So, `\ref{A}-\ref{B}` are not rendered in the visual order A-B, but in the wrong one B-A (because the hyphen does not “see” the digits inside the `\hbox`es). If you need `\ref` ranges, the best option is to define a dedicated macro like this (to avoid explicit direction changes in the body; here `\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.¹⁸

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

WARNING As of April 2019 there is a bug with `\parshape` in luatex (a 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** .

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

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

extras is used for miscellaneous readjustments which do not fit into the previous groups. Currently redefines in luatex `\underline` and `\LaTeX2e` **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 `rl` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

\BabelPatchSection `{\section-name}`

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

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

New 3.17 Something like:

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

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

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

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

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

(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}{\footnote}
```

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

1.25 Language attributes

`\languageattribute`

This is a user-level command, to be used in the preamble of a document (after `\usepackage[...]{babel}`), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

Very often, using a *modifier* in a package option is better.

Several language definition files use their own methods to set options. For example, `french` uses `\frenchsetup`, `magyar` (1.5) uses `\magyarOptions`; modifiers provided by `spanish` have no attribute counterparts. Macros setting options are also used (eg, `\ProsodicMarksOn` in `latin`).

1.26 Hooks

New 3.9a A hook is a piece of code to be executed at certain events. Some hooks are predefined when `luatex` and `xetex` are used.

`\AddBabelHook` [`<lang>`] [`<name>`] [`<event>`] [`<code>`]

The same name can be applied to several events. Hooks with a certain `<name>` may be enabled and disabled for all defined events with `\EnableBabelHook{<name>}`, `\DisableBabelHook{<name>}`. Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`).

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

Current events are the following; in some of them you can use one to three $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ parameters (`#1`, `#2`, `#3`), with the meaning given:

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.27 Languages supported by babel with ldf files

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

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

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

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

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

```

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

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

```

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

```

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

```

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

```

1.29 Tweaking some features

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

New 3.36 Sometimes you might need to disable some babel features. Currently this macro understands the following keys (and only for luatex), with values on or off: bidi.text, bidi.mirroring, bidi.mapdigits, layout.lists, layout.tabular, linebreak.sea, linebreak.cjk, justify.arabic. For example, you can set \backslash 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 \backslash ref inside the argument of \backslash chapter (or just use \backslash ref inside \backslash MakeUppercase), \LaTeX will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use \backslash lowercase{ \backslash ref{foo}} inside the argument of \backslash chapter, or, if you will not use shorthands in labels, set the safe option to none or bib.

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

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

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

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is a similar issue with floats, too. There is no known workaround.
- `Babel` does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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`.

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

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

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with language.dat

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

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.

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.

2.1 Format

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

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

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

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

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

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

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

A typical error when using `babel` is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.

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

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

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

Some recommendations:

- The preferred shorthand is `"`, which is not used in \LaTeX (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras\langle lang \rangle` except for `umlauthigh` and `friends`, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁶
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

²⁶But not removed, for backward compatibility.

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 ttf, vf, ps1, otf, mf files and the like, but also fd ones.
- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.
- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

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

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

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

3.3 Skeleton

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

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

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

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



```

\bb1@declare@ttribute{<language>}{<attrib>}{%
  \expandafter\addto\expandafter\extras<language>
  \expandafter{\extras<attrib><language>}%
  \let\captions<language>\captions<attrib><language>}

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

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

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

`\bb1@activate`
`\bb1@deactivate`

The command `\bb1@activate` is used to change the way an active character expands. `\bb1@activate` ‘switches on’ the active behavior of the character. `\bb1@deactivate` lets the active character expand to its former (mostly) non-active self.

`\declare@shorthand`

The macro `\declare@shorthand` is used to define the various shorthands. It takes three

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

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

3.5 Support for saving macro definitions

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

`\babel@save`

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

`\babel@savevariable`

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

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

3.6 Support for extending macros

`\addto`

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

3.7 Macros common to a number of languages

`\bbl@allowhyphens`

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

`\allowhyphens`

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

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

`\set@low@box`

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

`\save@sf@q`

Sometimes it is necessary to preserve the `\spacefactor`. For this purpose the macro

²⁷This mechanism was introduced by Bernd Raichle.

`\save@sf@q` is available. It takes one argument, saves the current spacefactor, executes the argument, and restores the spacefactor.

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

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

The $\langle category \rangle$ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.²⁸ It may be empty, too, but in such a case

²⁸In future releases further categories may be added.

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\monthinname{Jänner}

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

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

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

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

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

\EndBabelCommands
```

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

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

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

`\EndBabelCommands` Marks the end of the series of blocks.

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

`\SetString` `{\macro-name}{\string}`
Adds `\macro-name` to the current category, and defines globally `\lang-macro-name` to `\code` (after applying the transformation corresponding to the current charset or defined with the hook `stringprocess`).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

`\SetStringLoop` `{\macro-name}{\string-list}`
A convenient way to define several ordered names at once. For example, to define `\abmoniname`, `\abmoniiname`, etc. (and similarly with `abday`):

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

#1 is replaced by the roman numeral.

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

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

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

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

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

```

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

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap `{\to-lower-macros}`

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

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

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

```

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

```

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

4 Changes

4.1 Changes in `babel` version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.

- ' (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop can happen. It worked incorrectly with `^` (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that led to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with `babel` were not recognized when called as global options.

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The `babel` package after unpacking consists of the following files:

`switch.def` defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

`charset` the encoding used in the `ini` file.

`version` of the `ini` file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.65.2536>>
2 <<date=2021/10/25>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1{#2}}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@c1#1{\csname bbl@#1\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19   \fi}
20 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\empty\else#3\fi}}
```

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

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

`\bbl@afterelse` Because the code that is used in the handling of active characters may need to look ahead, we take
`\bbl@afterfi`

extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³⁰. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```
27 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
28 \long\def\bbl@afterfi#1\fi{\fi#1}
```

`\bbl@exp` Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here `\` stands for `\noexpand` and `<.>` for `\noexpand` applied to a built macro name (the latter does not define the macro if undefined to `\relax`, because it is created locally). The result may be followed by extra arguments, if necessary.

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

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

```
39 \def\bbl@tempa#1{%
40   \long\def\bbl@trim##1##2{%
41     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
42   \def\bbl@trim@c{%
43     \ifx\bbl@trim@a\@sptoken
44       \expandafter\bbl@trim@b
45     \else
46       \expandafter\bbl@trim@b\expandafter#1%
47     \fi}%
48   \long\def\bbl@trim@b##1 \@nil{\bbl@trim@i##1}}
49 \bbl@tempa{ }
50 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
51 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}
```

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

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

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

```

66      \bbl@afterfi\expandafter\@secondoftwo
67      \fi
68      \else
69      \expandafter\@firstoftwo
70      \fi}}
71 \endgroup

```

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

```

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

```

For each element in the comma separated `<key>=<value>` list, execute `<code>` with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the `<key>` alone, it passes `\@empty` (ie, the macro thus named, not an empty argument, which is what you get with `<key>=` and no value).

```

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

```

A *for* loop. Each item (trimmed), is #1. It cannot be nested (it’s doable, but we don’t need it).

```

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

```

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

```

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

```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace `elax` by `ho`, then `\relax` becomes `\rho`). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not*

work is in `\bbl@TG@@date`, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with `\bbl@replace`; I'm not sure ckecking the replacement is really necessary or just paranoia).

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

177 \ifx\IfFormatAtLeastTF\@undefined
178   \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
179 \else
180   \let\bbl@ifformatlater\IfFormatAtLeastTF
181 \fi

```

The following adds some code to `\extras...` both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with `#`'s. Used to deal with `alph`, `Alph` and `frenchspacing` when there are already changes (with `\babel@save`).

```

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

```

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

```

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

```

7.1 Multiple languages

`\language` Plain \TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The

following block is used in `switch.def` and `hyphen.cfg`; the latter may seem redundant, but remember `babel` doesn't require loading `switch.def` in the format.

```
200 <<*Define core switching macros>> ≡
201 \ifx\language\undefined
202   \csname newcount\endcsname\language
203 \fi
204 <</Define core switching macros>>
```

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

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

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

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

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

7.2 The Package File (\LaTeX , `babel.sty`)

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

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

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

```

238 \def\{\MessageBreak}%
239 \GenericWarning
240 {(babel) \@spaces\@spaces\@spaces}%
241 {Package babel Info: #1}%
242 \endgroup}
243 \def\bbl@info#1{%
244 \begingroup
245 \def\{\MessageBreak}%
246 \PackageInfo{babel}{#1}%
247 \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.

```

248 <<Basic macros>>
249 \@ifpackagewith{babel}{silent}
250 {\let\bbl@info\@gobble
251 \let\bbl@infowarn\@gobble
252 \let\bbl@warning\@gobble}
253 {}
254 %
255 \def\AfterBabelLanguage#1{%
256 \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.

```

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

```

7.3 base

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

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

```

275 \bbl@trace{Defining option 'base'}
276 \@ifpackagewith{babel}{base}{%
277 \let\bbl@onlyswitch\@empty
278 \let\bbl@provide@locale\relax
279 \input babel.def

```

```

280 \let\bbl@onlyswitch\@undefined
281 \ifx\directlua\@undefined
282 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
283 \else
284 \input luababel.def
285 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
286 \fi
287 \DeclareOption{base}{}%
288 \DeclareOption{showlanguages}{}%
289 \ProcessOptions
290 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
291 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
292 \global\let\@ifl@ter@@\@ifl@ter
293 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
294 \endinput}{}%

```

7.4 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.

```

295 \bbl@trace{key=value and another general options}
296 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
297 \def\bbl@tempb#1.#2{% Remove trailing dot
298   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
299 \def\bbl@tempd#1.#2@nnil{% TODO. Refactor lists?
300   \ifx\@empty#2%
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302   \else
303     \in@{,provide=}{, #1}%
304     \ifin@
305       \edef\bbl@tempc{%
306         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
307     \else
308       \in@{=}{ #1}%
309       \ifin@
310         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
311       \else
312         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
313         \bbl@csarg\edef{mod#1}{\bbl@tempb#2}%
314       \fi
315     \fi
316   \fi}
317 \let\bbl@tempc\@empty
318 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty@nnil}
319 \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.

```

320 \DeclareOption{KeepShorthandsActive}{}
321 \DeclareOption{activeacute}{}
322 \DeclareOption{activegrave}{}
323 \DeclareOption{debug}{}
324 \DeclareOption{noconfigs}{}
325 \DeclareOption{showlanguages}{}
326 \DeclareOption{silent}{}

```

```

327% \DeclareOption{mono}{}
328 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
329 \chardef\bbl@iniflag\z@
330 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
331 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\@tw@} % add = 2
332 \DeclareOption{provide**=*}{\chardef\bbl@iniflag\@thr@@} % add + main
333% A separate option
334 \let\bbl@autoload@options\@empty
335 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
336% Don't use. Experimental. TODO.
337 \newif\ifbbl@single
338 \DeclareOption{selectors=off}{\bbl@singletrue}
339 <<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.

```

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

```

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

```

346 \def\bbl@tempa#1=#2\bbl@tempa{%
347   \bbl@csarg\ifx{opt@#1}\@nnil
348     \bbl@csarg\edef{opt@#1}{#2}%
349   \else
350     \bbl@error
351     {Bad option '#1=#2'. Either you have misspelled the\\%
352     key or there is a previous setting of '#1'. Valid\\%
353     keys are, among others, 'shorthands', 'main', 'bidi',\\%
354     'strings', 'config', 'headfoot', 'safe', 'math'.}%
355     {See the manual for further details.}
356   \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.

```

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

```

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

```

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

```



```

373      \def\bbl@opt@provide{#2}%
374      \bbl@replace\bbl@opt@provide{;}{,}%
375      \fi}
376 \fi
377 %

```

7.5 Conditional loading of shorthands

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

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

```

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

```

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

```

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

```

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

```

399   \edef\bbl@opt@shorthands{%
400     \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

401   \bbl@ifshorthand{'}%
402     {\PassOptionsToPackage{activeacute}{babel}}{}
403   \bbl@ifshorthand{`}%
404     {\PassOptionsToPackage{activegrave}{babel}}{}
405 \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.

```

406 \ifx\bbl@opt@headfoot\@nnil\else
407   \@addto@macro\@resetactivechars{%
408     \set@typeset@protect
409     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
410     \let\protect\noexpand}
411 \fi

```

For the option safe we use a different approach – \bbl@opt@safe says which macros are redefined (B for bibs and R for refs). By default, both are set.

```
412 \ifx\bbl@opt@safe\undefined
413   \def\bbl@opt@safe{BR}
414 \fi
```

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

```
415 \ifx\bbl@opt@main\@nnil\else
416   \edef\bbl@language@opts{%
417     \ifx\bbl@language@opts\empty\else\bbl@language@opts,\fi
418     \bbl@opt@main}
419 \fi
```

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

```
420 \bbl@trace{Defining IfBabelLayout}
421 \ifx\bbl@opt@layout\@nnil
422   \newcommand\IfBabelLayout[3]{#3}%
423 \else
424   \newcommand\IfBabelLayout[1]{%
425     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
426     \ifin@
427       \expandafter\@firstoftwo
428     \else
429       \expandafter\@secondoftwo
430     \fi}
431 \fi
432 </package>
433 <*core>
```

7.6 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*.

```
434 \ifx\ldf@quit\undefined\else
435 \endinput\fi % Same line!
436 <<Make sure ProvidesFile is defined>>
437 \ProvidesFile{babel.def}[<<date>>] <<version>> Babel common definitions]
438 \ifx\AtBeginDocument\undefined % TODO. change test.
439   <<Emulate LaTeX>>
440 \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.

```
441 </core>
442 <*package | core>
```

8 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.

```
443 \def\bbl@version{<<version>>}
444 \def\bbl@date{<<date>>}
445 <<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.

```

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

```

461 \def\bbl@fixname#1{%
462   \begingroup
463   \def\bbl@tempe{l@}%
464   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
465   \bbl@tempd
466     {\lowercase\expandafter{\bbl@tempd}%
467     {\uppercase\expandafter{\bbl@tempd}%
468     \@empty
469     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
470     {\uppercase\expandafter{\bbl@tempd}}}%
471     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
472     {\lowercase\expandafter{\bbl@tempd}}}%
473     \@empty
474   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
475   \bbl@tempd
476   \bbl@exp{\bbl@usehooks{language}{\{#1\}}}
477 \def\bbl@iflanguage#1{%
478   \@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`.

```

479 \def\bbl@bcpcase#1#2#3#4\@#5{%
480   \ifx\@empty#3%
481     \uppercase{\def#5{#1#2}}%
482   \else
483     \uppercase{\def#5{#1}}%
484     \lowercase{\def#5{#5#2#3#4}}%
485   \fi}
486 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
487   \let\bbl@bcp\relax
488   \lowercase{\def\bbl@tempa{#1}}%

```



```

548 \expandafter\ifx\csize date\language\endcsize\relax
549 \IfFileExists{babel-\language.tex}%
550 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
551 {}%
552 \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.

```

553 \def\iflanguage#1{%
554 \bbl@iflanguage{#1}{%
555 \ifnum\csize l@#1\endcsize=\language
556 \expandafter\@firstoftwo
557 \else
558 \expandafter\@secondoftwo
559 \fi}}

```

8.1 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.

```

560 \let\bbl@select@type\z@
561 \edef\selectlanguage{%
562 \noexpand\protect
563 \expandafter\noexpand\csize selectlanguage \endcsize}

```

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`.

```
564 \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.

```
565 \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.

```
566 \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`

```

567 \def\bbl@push@language{%
568 \ifx\language\@undefined\else
569 \ifx\currentgrouplevel\@undefined
570 \xdef\bbl@language@stack{\language+\bbl@language@stack}%
571 \else

```

```

572 \ifnum\currentgrouplevel=\z@
573 \xdef\bbl@language@stack{\language+}%
574 \else
575 \xdef\bbl@language@stack{\language+\bbl@language@stack}%
576 \fi
577 \fi
578 \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`.

```

579 \def\bbl@pop@lang#1+#2\@@{%
580 \edef\language{#1}%
581 \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).

```

582 \let\bbl@ifrestoring\@secondoftwo
583 \def\bbl@pop@language{%
584 \expandafter\bbl@pop@lang\bbl@language@stack\@@
585 \let\bbl@ifrestoring\@firstoftwo
586 \expandafter\bbl@set@language\expandafter{\language}%
587 \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).

```

588 \chardef\localeid\z@
589 \def\bbl@id@last{0} % No real need for a new counter
590 \def\bbl@id@assign{%
591 \bbl@ifunset{bbl@id@\language}%
592 {\count@bbl@id@last\relax
593 \advance\count@@ne
594 \bbl@csarg\chardef{id@\language}\count@
595 \edef\bbl@id@last{the\count@}%
596 \ifcase\bbl@engine\or
597 \directlua{
598 Babel = Babel or {}
599 Babel.locale_props = Babel.locale_props or {}
600 Babel.locale_props[\bbl@id@last] = {}
601 Babel.locale_props[\bbl@id@last].name = '\language'
602 }%
603 \fi}%
604 {}%
605 \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

606 \expandafter\def\csname selectlanguage \endcsname#1{%
607 \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw\fi
608 \bbl@push@language

```

```

609 \aftergroup\bbbl@pop@language
610 \bbbl@set@language{#1}}

```

`\bbbl@set@language` The macro `\bbbl@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. `\bbbl@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).

```

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

```

```

655 \def\select@language#1{% from set@, babel@aux
656 % set hmap
657 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
658 % set name
659 \edef\language#1}%
660 \bbl@fixname\language
661 % TODO. name@map must be here?
662 \bbl@provide@locale
663 \bbl@iflanguage\language{%
664   \expandafter\ifx\csname date\language\endcsname\relax
665     \bbl@error
666     {Unknown language '\language'. Either you have\\%
667       misspelled its name, it has not been installed,\\%
668       or you requested it in a previous run. Fix its name,\\%
669       install it or just rerun the file, respectively. In\\%
670       some cases, you may need to remove the aux file}%
671     {You may proceed, but expect wrong results}%
672   \else
673     % set type
674     \let\bbl@select@type\z@
675     \expandafter\bbl@switch\expandafter{\language}%
676     \fi}}
677 \def\babel@aux#1#2{%
678   \select@language{#1}%
679   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
680     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
681 \def\babel@toc#1#2{%
682   \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.

```

683 \newif\ifbbl@usedategroup
684 \def\bbl@switch#1{% from select@, foreign@
685 % make sure there is info for the language if so requested
686 \bbl@ensureinfo{#1}%
687 % restore
688 \originalTeX
689 \expandafter\def\expandafter\originalTeX\expandafter{%
690   \csname noextras#1\endcsname
691   \let\originalTeX\empty
692   \babel@beginsave}%
693 \bbl@usehooks{afterreset}}}%
694 \languageshorthands{none}%
695 % set the locale id
696 \bbl@id@assign
697 % switch captions, date
698 % No text is supposed to be added here, so we remove any
699 % spurious spaces.
700 \bbl@bsphack

```



```

701 \ifcase\bbl@select@type
702 \csname captions#1\endcsname\relax
703 \csname date#1\endcsname\relax
704 \else
705 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
706 \ifin@
707 \csname captions#1\endcsname\relax
708 \fi
709 \bbl@xin@{,date,}{, \bbl@select@opts,}%
710 \ifin@ % if \foreign... within \<lang>date
711 \csname date#1\endcsname\relax
712 \fi
713 \fi
714 \bbl@esphack
715 % switch extras
716 \bbl@usehooks{beforeextras}{}%
717 \csname extras#1\endcsname\relax
718 \bbl@usehooks{afterextras}{}%
719 % > babel-ensure
720 % > babel-sh-<short>
721 % > babel-bidi
722 % > babel-fontspec
723 % hyphenation - case mapping
724 \ifcase\bbl@opt@hyphenmap\or
725 \def\BabelLower##1##2{\lccode##1=##2\relax}%
726 \ifnum\bbl@hymapsel>4\else
727 \csname\language @bbl@hyphenmap\endcsname
728 \fi
729 \chardef\bbl@opt@hyphenmap\z@
730 \else
731 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
732 \csname\language @bbl@hyphenmap\endcsname
733 \fi
734 \fi
735 \let\bbl@hymapsel\@cclv
736 % hyphenation - select rules
737 \ifnum\csname l@\language\endcsname=\l@unhyphenated
738 \edef\bbl@tempa{u}%
739 \else
740 \edef\bbl@tempa{\bbl@c1{lnbrk}}%
741 \fi
742 % linebreaking - handle u, e, k (v in the future)
743 \bbl@xin@{/u}{/\bbl@tempa}%
744 \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
745 \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
746 \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
747 \ifin@
748 % unhyphenated/kashida/elongated = allow stretching
749 \language\l@unhyphenated
750 \babel@savevariable\emergencystretch
751 \emergencystretch\maxdimen
752 \babel@savevariable\hbadness
753 \hbadness\M
754 \else
755 % other = select patterns
756 \bbl@patterns{#1}%
757 \fi
758 % hyphenation - mins
759 \babel@savevariable\lefthyphenmin

```

```

760 \babel@savevariable\rightthyphenmin
761 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
762 \set@hyphenmins\tw@\thr@\relax
763 \else
764 \expandafter\expandafter\expandafter\set@hyphenmins
765 \csname #1hyphenmins\endcsname\relax
766 \fi}

```

otherlanguage The `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

767 \long\def\otherlanguage#1{%
768 \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\thr@\fi
769 \csname selectlanguage\endcsname{#1}%
770 \ignorespaces}

```

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

```

771 \long\def\endotherlanguage{%
772 \global\@ignoretrue\ignorespaces}

```

otherlanguage* The `otherlanguage` environment is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as ‘figure’. This environment makes use of `\foreign@language`.

```

773 \expandafter\def\csname otherlanguage*\endcsname{%
774 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
775 \def\bbl@otherlanguage@s[#1]#2{%
776 \ifnum\bbl@hymapsel=\ccclv\chardef\bbl@hymapsel4\relax\fi
777 \def\bbl@select@opts{#1}%
778 \foreign@language{#2}}

```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```

779 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

\foreignlanguage The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.

Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras<lang>` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

(3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in `vmode` and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into `hmode` with the surrounding `lang`, and with `\foreignlanguage*` with the new `lang`.

```

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

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the other `language*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bbl@switch`.

```

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

```

`\bbl@patterns` This macro selects the hyphenation patterns by changing the `\language` register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.

It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

```

830 \let\bbl@hyphlist\@empty
831 \let\bbl@hyphenation@\relax
832 \let\bbl@pttnlist\@empty
833 \let\bbl@patterns@\relax
834 \let\bbl@hymapsel=\@cclv
835 \def\bbl@patterns#1{%
836   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
837     \csname l@#1\endcsname
838     \edef\bbl@tempa{#1}%
839   \else
840     \csname l@#1:\f@encoding\endcsname
841     \edef\bbl@tempa{#1:\f@encoding}%
842   \fi
843   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
844   % > luatex
845   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
846     \begingroup
847       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
848     \ifin@else
849       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
850       \hyphenation{%
851         \bbl@hyphenation@
852         \@ifundefined{bbl@hyphenation@#1}%
853         \@empty
854         {\space\csname bbl@hyphenation@#1\endcsname}}%
855       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
856     \fi
857   \endgroup}}

```

hyphenrules The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change \language and when the hyphenation rules specified were not loaded it has no effect. Note however, \lccode's and font encodings are not set at all, so in most cases you should use `otherlanguage*`.

```

858 \def\hyphenrules#1{%
859   \edef\bbl@tempf{#1}%
860   \bbl@fixname\bbl@tempf
861   \bbl@iflanguage\bbl@tempf{%
862     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
863     \ifx\languageshorthands\undefined\else
864       \languageshorthands{none}%
865     \fi
866     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
867       \set@hyphenmins\tw@\thr@@\relax
868     \else
869       \expandafter\expandafter\expandafter\set@hyphenmins
870       \csname\bbl@tempf hyphenmins\endcsname\relax
871     \fi}}
872 \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.

```

873 \def\providehyphenmins#1#2{%

```

```

874 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
875   \@namedef{#1hyphenmins}{#2}%
876 \fi}

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

877 \def\set@hyphenmins#1#2{%
878   \lefthyphenmin#1\relax
879   \righthyphenmin#2\relax}

\ProvidesLanguage The identification code for each file is something that was introduced in  $\text{\TeX}$  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.

880 \ifx\ProvidesFile\@undefined
881   \def\ProvidesLanguage#1[#2 #3 #4]{%
882     \wlog{Language: #1 #4 #3 <#2>}%
883   }
884 \else
885   \def\ProvidesLanguage#1{%
886     \begingroup
887     \catcode`\ 10 %
888     \@makeother\/%
889     \ifnextchar[%]
890       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
891   \def\@provideslanguage#1[#2]{%
892     \wlog{Language: #1 #2}%
893     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
894   \endgroup}
895 \fi

\originalTeX The macro \originalTeX should be known to  $\text{\TeX}$  at this moment. As it has to be expandable we \let
it to \@empty instead of \relax.

896 \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.

897 \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’:

898 \providecommand\setlocale{%
899   \bbl@error
900   {Not yet available}%
901   {Find an armchair, sit down and wait}}
902 \let\uselocale\setlocale
903 \let\locale\setlocale
904 \let\selectlocale\setlocale
905 \let\localename\setlocale
906 \let\textlocale\setlocale
907 \let\textlanguage\setlocale
908 \let\languagetext\setlocale

```

8.2 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.

`\@nopatterns`

`\@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 $\text{\TeX 2}\epsilon$, 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.

```

909 \edef\bbl@nulllanguage{\string\language=0}
910 \def\bbl@nocaption{\protect\bbl@nocaption@i}
911 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
912   \global\@namedef{#2}{\textbf{?#1?}}%
913   \@nameuse{#2}%
914   \edef\bbl@tempa{#1}%
915   \bbl@sreplace\bbl@tempa{name}{}}%
916   \bbl@warning{% TODO.
917     \@backslashchar#1 not set for '\language'. Please,\%
918     define it after the language has been loaded\\%
919     (typically in the preamble) with:\\%
920     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
921     Reported}}
922 \def\bbl@tentative{\protect\bbl@tentative@i}
923 \def\bbl@tentative@i#1{%
924   \bbl@warning{%
925     Some functions for '#1' are tentative.\\%
926     They might not work as expected and their behavior\\%
927     could change in the future.\\%
928     Reported}}
929 \def\@nolanerr#1{%
930   \bbl@error
931   {You haven't defined the language '#1' yet.\\%
932     Perhaps you misspelled it or your installation\\%
933     is not complete}%
934   {Your command will be ignored, type <return> to proceed}}
935 \def\@nopatterns#1{%
936   \bbl@warning
937   {No hyphenation patterns were preloaded for\\%
938     the language '#1' into the format.\\%
939     Please, configure your TeX system to add them and\\%
940     rebuild the format. Now I will use the patterns\\%
941     preloaded for \bbl@nulllanguage\space instead}}
942 \let\bbl@usehooks\@gobbles
943 \ifx\bbl@onlyswitch\@empty\endinput\fi
944 % Here ended switch.def

```

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

```

945 \ifx\directlua\@undefined\else
946   \ifx\bbl@luapatterns\@undefined
947     \input luababel.def
948   \fi
949 \fi
950 <<Basic macros>>
951 \bbl@trace{Compatibility with language.def}
952 \ifx\bbl@languages\@undefined
953   \ifx\directlua\@undefined
954     \openin1 = language.def % TODO. Remove hardcoded number
955     \ifeof1
956       \closein1
957       \message{I couldn't find the file language.def}
958     \else
959       \closein1

```

```

960 \begingroup
961 \def\addlanguage#1#2#3#4#5{%
962 \expandafter\ifx\csname lang@#1\endcsname\relax\else
963 \global\expandafter\let\csname l@#1\expandafter\endcsname
964 \csname lang@#1\endcsname
965 \fi}%
966 \def\uselanguage#1{%
967 \input language.def
968 \endgroup
969 \fi
970 \fi
971 \chardef\l@english\z@
972 \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.

```

973 \def\addto#1#2{%
974 \ifx#1\@undefined
975 \def#1{#2}%
976 \else
977 \ifx#1\relax
978 \def#1{#2}%
979 \else
980 {\toks@\expandafter{#1#2}%
981 \xdef#1{\the\toks@}}%
982 \fi
983 \fi}

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. `TODO`. Always used with additional expansions. Move them here? Move the macro to `basic`?

```

984 \def\bbl@withactive#1#2{%
985 \begingroup
986 \lccode`~=`#2\relax
987 \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`.

```

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

```

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

```

993 \def\bbl@redefine@long#1{%
994 \edef\bbl@tempa{\bbl@stripslash#1}%
995 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
996 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
997 \@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_`.

```

998 \def\bbl@redefineroast#1{%
999   \edef\bbl@tempa{\bbl@stripslash#1}%
1000   \bbl@ifunset{\bbl@tempa\space}%
1001   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1002     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1003   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1004   \@namedef{\bbl@tempa\space}}
1005 \@onlypreamble\bbl@redefineroast

```

8.3 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.

```

1006 \bbl@trace{Hooks}
1007 \newcommand\AddBabelHook[3][{}]{%
1008   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1009   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1010   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1011   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1012   {\bbl@csarg\bbl@add{\ev@#3@#1}{\bbl@elth{#2}}}%
1013   {\bbl@csarg\let{\ev@#2@#3@#1}\relax}%
1014   \bbl@csarg\newcommand{\ev@#2@#3@#1}{\bbl@tempb}}
1015 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{\hk@#1}\@firstofone}
1016 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{\hk@#1}\@gobble}
1017 \def\bbl@usehooks#1#2{%
1018   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1019   \def\bbl@elth##1{%
1020     \bbl@cs{\hk@##1}{\bbl@cs{\ev@##1@#1@#2}}%
1021     \bbl@cs{\ev@#1@}%
1022     \ifx\language\@undefined\else % Test required for Plain (?)
1023       \ifx\UseHook\@undefined\else\UseHook{babel/\language/#1}\fi
1024       \def\bbl@elth##1{%
1025         \bbl@cs{\hk@##1}{\bbl@cl{\ev@##1@#1@#2}}%
1026         \bbl@cl{\ev@#1@}%
1027       \fi}

```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for `hyphen.cfg` are also loaded (just in case you need them for some reason).

```

1028 \def\bbl@evargs{% <- don't delete this comma
1029   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1030   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1031   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1032   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1033   beforestart=0,language=2}
1034 \ifx\NewHook\@undefined\else
1035   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1036   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1037 \fi

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro `\bbl@e@<language>` contains `\bbl@ensure{\<include>}{\<exclude>}{\<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in

the exclude list. If the fontenc is given (and not \relax), the \fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

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

```

```

1093      \fi
1094      \expandafter\bbbl@tempa
1095      \fi}%
1096      \bbbl@tempa#1\@empty}
1097 \def\bbbl@captionslist{%
1098   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1099   \contentsname\listfigurename\listtablename\indexname\figurename
1100   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1101   \alsoname\proofname\glossaryname}

```

8.4 Setting up language files

`\LdfInit` `\LdfInit` macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before.

At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save its name as the last called option, even if not loaded.

Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions with the `\let` primitive. Therefore we store its current catcode and restore it later on.

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing #2 through `string`. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\@undefined`.

If so, we call `\ldf@quit` to set the main language, restore the category code of the @-sign and call `\endinput`

When #2 was *not* a control sequence we construct one and compare it with `\relax`.

Finally we check `\originalTeX`.

```

1102 \bbbl@trace{Macros for setting language files up}
1103 \def\bbbl@ldfinit{%
1104   \let\bbbl@screset\@empty
1105   \let\BabelStrings\bbbl@opt@string
1106   \let\BabelOptions\@empty
1107   \let\BabelLanguages\relax
1108   \ifx\originalTeX\@undefined
1109     \let\originalTeX\@empty
1110   \else
1111     \originalTeX
1112   \fi}
1113 \def\LdfInit#1#2{%
1114   \chardef\atcatcode=\catcode`\@
1115   \catcode`\@=11\relax
1116   \chardef\eqcatcode=\catcode`\=
1117   \catcode`\==12\relax
1118   \expandafter\if\expandafter\@backslashchar
1119     \expandafter\@car\string#2\@nil
1120   \ifx#2\@undefined\else
1121     \ldf@quit{#1}%
1122   \fi
1123 \else
1124   \expandafter\ifx\csname#2\endcsname\relax\else
1125     \ldf@quit{#1}%
1126   \fi
1127 \fi
1128 \bbbl@ldfinit}

```

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

```
1129 \def\ldf@quit#1{%
1130   \expandafter\main@language\expandafter{#1}%
1131   \catcode`\@=\atcatcode \let\atcatcode\relax
1132   \catcode`\==\eqcatcode \let\eqcatcode\relax
1133   \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.

```
1134 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1135   \bbl@afterlang
1136   \let\bbl@afterlang\relax
1137   \let\BabelModifiers\relax
1138   \let\bbl@screset\relax}%
1139 \def\ldf@finish#1{%
1140   \loadlocalcfg{#1}%
1141   \bbl@afterldf{#1}%
1142   \expandafter\main@language\expandafter{#1}%
1143   \catcode`\@=\atcatcode \let\atcatcode\relax
1144   \catcode`\==\eqcatcode \let\eqcatcode\relax}
```

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

```
1145 \@onlypreamble\LdfInit
1146 \@onlypreamble\ldf@quit
1147 \@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.

```
1148 \def\main@language#1{%
1149   \def\bbl@main@language{#1}%
1150   \let\language\bbl@main@language % TODO. Set localename
1151   \bbl@id@assign
1152   \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`.

```
1153 \def\bbl@beforestart{%
1154   \def\@nolanerr##1{%
1155     \bbl@warning{Undefined language '##1' in aux.\Reported}}%
1156   \bbl@usehooks{beforestart}}}%
1157 \global\let\bbl@beforestart\relax}
1158 \AtBeginDocument{%
1159   {\@nameuse{bbl@beforestart}}% Group!
1160   \if@filesw
1161     \providecommand\babel@aux[2]{}%
1162     \immediate\write\@mainaux{%
1163       \string\providecommand\string\babel@aux[2]{}%
1164       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1165   \fi
1166   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1167   \ifbbl@single % must go after the line above.
1168     \renewcommand\selectlanguage[1]{}%
1169     \renewcommand\foreignlanguage[2]{#2}%
```

```

1170 \global\let\babel@aux\@gobbletwo % Also as flag
1171 \fi
1172 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1173 \def\select@language@x#1{%
1174 \ifcase\bbl@select@type
1175 \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1176 \else
1177 \select@language{#1}%
1178 \fi}

```

8.5 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 \TeX 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.

```

1179 \bbl@trace{Shorthands}
1180 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1181 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1182 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1183 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1184 \begingroup
1185 \catcode`#1\active
1186 \nfss@catcodes
1187 \ifnum\catcode`#1=\active
1188 \endgroup
1189 \bbl@add\nfss@catcodes{\@makeother#1}%
1190 \else
1191 \endgroup
1192 \fi
1193 \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.

```

1194 \def\bbl@remove@special#1{%
1195 \begingroup
1196 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1197 \else\noexpand##1\noexpand##2\fi}%
1198 \def\do{\x\do}%
1199 \def\@makeother{\x\@makeother}%
1200 \edef\x{\endgroup
1201 \def\noexpand\dospecials{\dospecials}%
1202 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1203 \def\noexpand\@sanitize{\@sanitize}%
1204 \fi}%
1205 \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).

```
1206 \def\bbl@active@def#1#2#3#4{%
1207   \@namedef{#3#1}{%
1208     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1209       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1210     \else
1211       \bbl@afterfi\csname#2@sh@#1\endcsname
1212     \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.

```
1213 \long\@namedef{#3@arg#1}##1{%
1214   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1215     \bbl@afterelse\csname#4#1\endcsname##1%
1216   \else
1217     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1218   \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.

```
1219 \def\@initiate@active@char#1#2#3{%
1220   \bbl@ifunset{active@char\string#1}%
1221   {\bbl@withactive
1222     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1223   {}}

```

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).

```
1224 \def\@initiate@active@char#1#2#3{%
1225   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1226   \ifx#1\@undefined
1227     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1228   \else
1229     \bbl@csarg\let{oridef@#2}#1%
1230     \bbl@csarg\edef{oridef@#2}{%
1231       \let\noexpand#1%
1232       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1233   \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*").

```
1234 \ifx#1#3\relax
1235   \expandafter\let\csname normal@char#2\endcsname#3%

```

```

1236 \else
1237   \bbl@info{Making #2 an active character}%
1238   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1239   \@namedef{normal@char#2}{%
1240     \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1241   \else
1242     \@namedef{normal@char#2}{#3}%
1243   \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).

```

1244   \bbl@restoreactive{#2}%
1245   \AtBeginDocument{%
1246     \catcode`#2\active
1247     \if@filesw
1248       \immediate\write\mainaux{\catcode`\string#2\active}%
1249     \fi}%
1250   \expandafter\bbl@add@special\csname#2\endcsname
1251   \catcode`#2\active
1252 \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⟩`).

```

1253 \let\bbl@tempa\@firstoftwo
1254 \if\string^#2%
1255   \def\bbl@tempa{\noexpand\textormath}%
1256 \else
1257   \ifx\bbl@mathnormal\@undefined\else
1258     \let\bbl@tempa\bbl@mathnormal
1259   \fi
1260 \fi
1261 \expandafter\edef\csname active@char#2\endcsname{%
1262   \bbl@tempa
1263     {\noexpand\if@safe@actives
1264       \noexpand\expandafter
1265       \expandafter\noexpand\csname normal@char#2\endcsname
1266     \noexpand\else
1267       \noexpand\expandafter
1268       \expandafter\noexpand\csname bbl@doactive#2\endcsname
1269     \noexpand\fi}%
1270   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1271 \bbl@csarg\edef{doactive#2}{%
1272   \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!).

```

1273 \bbl@csarg\edef{active@#2}{%
1274   \noexpand\active@prefix\noexpand#1%
1275   \expandafter\noexpand\csname active@char#2\endcsname}%

```

```

1276 \bbl@csarg\edef{normal@#2}{%
1277   \noexpand\active@prefix\noexpand#1%
1278   \expandafter\noexpand\csname normal@char#2\endcsname}%
1279 \expandafter\let\expandafter#1\csname bbl@normal@#2\endcsname

```

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn't exist we check for a shorthand with an argument.

```

1280 \bbl@active@def#2\user@group{user@active}{language@active}%
1281 \bbl@active@def#2\language@group{language@active}{system@active}%
1282 \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.

```

1283 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1284   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1285 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1286   {\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.

```

1287 \if\string'#2%
1288   \let\prim@s\bbl@prim@s
1289   \let\active@math@prime#1%
1290 \fi
1291 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}

```

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

```

1292 <<{*More package options}>> ≡
1293 \DeclareOption{math=active}{}
1294 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1295 <</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*.

```

1296 \@ifpackagewith{babel}{KeepShorthandsActive}%
1297   {\let\bbl@restoreactive\@gobble}%
1298   {\def\bbl@restoreactive#1{%
1299     \bbl@exp{%
1300       \\\AfterBabelLanguage\\CurrentOption
1301       {\catcode`#1=\the\catcode`#1\relax}%
1302       \\\AtEndOfPackage
1303       {\catcode`#1=\the\catcode`#1\relax}}}%
1304   \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.

```

1305 \def\bbl@sh@select#1#2{%

```

```

1306 \expandafter\ifx\cshname#1@sh#2@sel\endcshname\relax
1307 \bbl@afterelse\bbl@scndcs
1308 \else
1309 \bbl@afterfi\cshname#1@sh#2@sel\endcshname
1310 \fi}

```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincshname` is available. If there is, the expansion will be more robust.

```

1311 \begingroup
1312 \bbl@ifunset{ifincshname}% TODO. Ugly. Correct? Only Plain?
1313 {\gdef\active@prefix#1{%
1314   \ifx\protect\@typeset@protect
1315   \else
1316     \ifx\protect\@unexpandable@protect
1317       \noexpand#1%
1318     \else
1319       \protect#1%
1320     \fi
1321   \expandafter\@gobble
1322 \fi}}
1323 {\gdef\active@prefix#1{%
1324   \ifincshname
1325     \string#1%
1326     \expandafter\@gobble
1327   \else
1328     \ifx\protect\@typeset@protect
1329     \else
1330       \ifx\protect\@unexpandable@protect
1331         \noexpand#1%
1332       \else
1333         \protect#1%
1334       \fi
1335       \expandafter\expandafter\expandafter\@gobble
1336     \fi
1337   \fi}}
1338 \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⟩`.

```

1339 \newif\if@safe@actives
1340 \@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.

```

1341 \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`.

```

1342 \chardef\bbl@activated\z@
1343 \def\bbl@activate#1{%
1344   \chardef\bbl@activated\@ne
1345   \bbl@withactive{\expandafter\let\expandafter}#1%

```



```

1346 \csname bbl@active@\string#1\endcsname}
1347 \def\bbl@deactivate#1{%
1348 \chardef\bbl@activated\tw@
1349 \bbl@withactive{\expandafter\let\expandafter}#1%
1350 \csname bbl@normal@\string#1\endcsname}

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
1351 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1352 \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.

1353 \def\babel@texpdf#1#2#3#4{%
1354 \ifx\texorpdfstring\undefined
1355 \textormath{#1}{#3}%
1356 \else
1357 \texorpdfstring{\textormath{#1}{#3}}{#2}%
1358 % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1359 \fi}
1360 %
1361 \def\declare@shorthand#1#2{\@decl@short{#1}#2\nil}
1362 \def\@decl@short#1#2#3\nil#4{%
1363 \def\bbl@tempa{#3}%
1364 \ifx\bbl@tempa\empty
1365 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1366 \bbl@ifunset{#1@sh@\string#2@}{}%
1367 {\def\bbl@tempa{#4}%
1368 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1369 \else
1370 \bbl@info
1371 {Redefining #1 shorthand \string#2\%
1372 in language \CurrentOption}%
1373 \fi}%
1374 \@namedef{#1@sh@\string#2@}{#4}%
1375 \else
1376 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1377 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1378 {\def\bbl@tempa{#4}%
1379 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1380 \else
1381 \bbl@info
1382 {Redefining #1 shorthand \string#2\string#3\%
1383 in language \CurrentOption}%
1384 \fi}%
1385 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1386 \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.

```

```

1387 \def\textormath{%
1388   \ifmmode
1389     \expandafter\@secondoftwo
1390   \else
1391     \expandafter\@firstoftwo
1392   \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’.

1393 \def\user@group{user}
1394 \def\language@group{english} % TODO. I don't like defaults
1395 \def\system@group{system}

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

1396 \def\useshorthands{%
1397   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
1398 \def\bb1@usesh@s#1{%
1399   \bb1@usesh@x
1400   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
1401   {#1}}
1402 \def\bb1@usesh@x#1#2{%
1403   \bb1@ifshorthand{#2}%
1404   {\def\user@group{user}%
1405     \initiate@active@char{#2}%
1406     #1%
1407     \bb1@activate{#2}}%
1408   {\bb1@error
1409     {I can't declare a shorthand turned off (\string#2)}
1410     {Sorry, but you can't use shorthands which have been\\
1411       turned off in the package options}}}

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

1412 \def\user@language@group{user@\language@group}
1413 \def\bb1@set@user@generic#1#2{%
1414   \bb1@ifunset{user@generic@active#1}%
1415   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
1416     \bb1@active@def#1\user@group{user@generic@active}{language@active}%
1417     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1418       \expandafter\noexpand\csname normal@char#1\endcsname}%
1419     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1420       \expandafter\noexpand\csname user@active#1\endcsname}}%
1421   \@empty}
1422 \newcommand\defineshorthand[3][user]{%
1423   \edef\bb1@tempa{\zap@space#1 \@empty}%
1424   \bb1@for\bb1@tempb\bb1@tempa{%
1425     \if*\expandafter\@car\bb1@tempb\@nil
1426       \edef\bb1@tempb{user@\expandafter@gobble\bb1@tempb}%
1427       \@expandtwoargs
1428       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
1429     \fi
1430     \declare@shorthand{\bb1@tempb}{#2}{#3}}}

```

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

```
1431 \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`.

```
1432 \def\aliasshorthand#1#2{%
1433   \bbl@ifshorthand{#2}%
1434   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1435     \ifx\document\@notprerr
1436       \@notshorthand{#2}%
1437     \else
1438       \initiate@active@char{#2}%
1439       \expandafter\let\csname active@char\string#2\expandafter\endcsname
1440         \csname active@char\string#1\endcsname
1441       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1442         \csname normal@char\string#1\endcsname
1443       \bbl@activate{#2}%
1444     \fi
1445   \fi}%
1446   {\bbl@error
1447     {Cannot declare a shorthand turned off (\string#2)}
1448     {Sorry, but you cannot use shorthands which have been\\%
1449       turned off in the package options}}}
```

`\@notshorthand`

```
1450 \def\@notshorthand#1{%
1451   \bbl@error{%
1452     The character '\string #1' should be made a shorthand character;\\%
1453     add the command \string\usesshorthands\string{#1\string} to
1454     the preamble.\\%
1455     I will ignore your instruction}%
1456   {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.

```
1457 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1458 \DeclareRobustCommand*\shorthandoff{%
1459   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1460 \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.

```
1461 \def\bbl@switch@sh#1#2{%
1462   \ifx#2\@nnil\else
1463     \bbl@ifunset{\bbl@active@\string#2}%
1464     {\bbl@error
1465       {I can't switch '\string#2' on or off--not a shorthand}%
1466       {This character is not a shorthand. Maybe you made\\%
1467         a typing mistake? I will ignore your instruction.}}%
1468     {\ifcase#1%   off, on, off*
```

```

1469     \catcode`#212\relax
1470   \or
1471     \catcode`#2\active
1472     \bbl@ifunset{bbl@shdef@\string#2}%
1473     {}%
1474     {\bbl@withactive{\expandafter\let\expandafter}#2%
1475       \csname bbl@shdef@\string#2\endcsname
1476       \bbl@csarg\let{shdef@\string#2}\relax}%
1477     \ifcase\bbl@activated\or
1478       \bbl@activate{#2}%
1479     \else
1480       \bbl@deactivate{#2}%
1481     \fi
1482   \or
1483     \bbl@ifunset{bbl@shdef@\string#2}%
1484     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1485     {}%
1486     \csname bbl@oricat@\string#2\endcsname
1487     \csname bbl@oridef@\string#2\endcsname
1488   \fi}%
1489   \bbl@afterfi\bbl@switch@sh#1%
1490 \fi}

```

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

```

1491 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1492 \def\bbl@putsh#1{%
1493   \bbl@ifunset{bbl@active@\string#1}%
1494   {\bbl@putsh@i#1\@empty\@nnil}%
1495   {\csname bbl@active@\string#1\endcsname}}
1496 \def\bbl@putsh@i#1#2\@nnil{%
1497   \csname\language@group @sh@\string#1@%
1498   \ifx\@empty#2\else\string#2@\fi\endcsname}
1499 \ifx\bbl@opt@shorthands\@nnil\else
1500   \let\bbl@s@initiate@active@char\initiate@active@char
1501   \def\initiate@active@char#1{%
1502     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1503   \let\bbl@s@switch@sh\bbl@switch@sh
1504   \def\bbl@switch@sh#1#2{%
1505     \ifx#2\@nnil\else
1506       \bbl@afterfi
1507       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1508     \fi}
1509   \let\bbl@s@activate\bbl@activate
1510   \def\bbl@activate#1{%
1511     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1512   \let\bbl@s@deactivate\bbl@deactivate
1513   \def\bbl@deactivate#1{%
1514     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1515 \fi

```

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

```

1516 \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.

```

1517 \def\bbl@prim@s{%

```

```

1518 \prime\futurelet\@let@token\bbl@pr@m@s}
1519 \def\bbl@if@primes#1#2{%
1520 \ifx#1\@let@token
1521 \expandafter\@firstoftwo
1522 \else\ifx#2\@let@token
1523 \bbl@afterelse\expandafter\@firstoftwo
1524 \else
1525 \bbl@afterfi\expandafter\@secondoftwo
1526 \fi\fi}
1527 \begingroup
1528 \catcode`\^=7 \catcode`\*= \active \lccode`\*=`^^
1529 \catcode`\'=12 \catcode`\`= \active \lccode`\`=``'
1530 \lowercase{%
1531 \gdef\bbl@pr@m@s{%
1532 \bbl@if@primes" '%
1533 \pr@@@s
1534 {\bbl@if@primes*^ \pr@@@t\egroup}}
1535 \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).

```

1536 \initiate@active@char{~}
1537 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1538 \bbl@activate{~}

```

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

```

1539 \expandafter\def\csname OT1dqpos\endcsname{127}
1540 \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

```

1541 \ifx\f@encoding\@undefined
1542 \def\f@encoding{OT1}
1543 \fi

```

8.6 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.

```

1544 \bbl@trace{Language attributes}
1545 \newcommand\languageattribute[2]{%
1546 \def\bbl@tempc{#1}%
1547 \bbl@fixname\bbl@tempc
1548 \bbl@iflanguage\bbl@tempc{%
1549 \bbl@vforeach{#2}{%

```

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

```

1550 \ifx\babelknown@attribs\@undefined
1551 \in@false
1552 \else
1553 \babelxin@{,\babeltempc-##1,}{,\babelknown@attribs,}%
1554 \fi
1555 \ifin@
1556 \babelwarning{%
1557     You have more than once selected the attribute '##1'\%
1558     for language #1. Reported}%
1559 \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.

```

1560 \babelexp{%
1561     \\\babeladd@list\\babelknown@attribs{\babeltempc-##1}}%
1562 \edef\babeltempa{\babeltempc-##1}%
1563 \expandafter\babel@ifknown@trib\expandafter{\babeltempa}\babel@attributes%
1564 {\csname\babeltempc @attr##1\endcsname}%
1565 {\@attrerr{\babeltempc}{##1}}%
1566 \fi}}
1567 \@onlypreamble\languageattribute

```

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

```

1568 \newcommand*{\@attrerr}[2]{%
1569 \babelerror
1570 {The attribute #2 is unknown for language #1.}%
1571 {Your command will be ignored, type <return> to proceed}}

```

`\babeldeclare@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}`.

```

1572 \def\babeldeclare@ttribute#1#2#3{%
1573 \babelxin@{,#2,}{,\BabelModifiers,}%
1574 \ifin@
1575 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1576 \fi
1577 \babeladd@list\babel@attributes{#1-#2}%
1578 \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

`\babel@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.

```

1579 \def\babel@ifattributeset#1#2#3#4{%
1580 \ifx\babelknown@attribs\@undefined
1581 \in@false
1582 \else
1583 \babelxin@{,#1-#2,}{,\babelknown@attribs,}%
1584 \fi
1585 \ifin@
1586 \babel@afterelse#3%
1587 \else
1588 \babel@afterfi#4%
1589 \fi}

```

`\babel@ifknown@trib` 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.

```

1590 \def\bbl@ifknown@ttrib#1#2{%
1591   \let\bbl@tempa\@secondoftwo
1592   \bbl@loopx\bbl@tempb{#2}{%
1593     \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1594     \ifin@
1595       \let\bbl@tempa\@firstoftwo
1596     \else
1597       \fi}%
1598   \bbl@tempa}

```

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

```

1599 \def\bbl@clear@ttribs{%
1600   \ifx\bbl@attributes\undefined\else
1601     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1602       \expandafter\bbl@clear@ttrib\bbl@tempa.
1603     }%
1604     \let\bbl@attributes\undefined
1605   \fi}
1606 \def\bbl@clear@ttrib#1-#2.{%
1607   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1608 \AtBeginDocument{\bbl@clear@ttribs}

```

8.7 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`

```

1609 \bbl@trace{Macros for saving definitions}
1610 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1611 \newcount\babel@savecnt
1612 \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.

```

1613 \def\babel@save#1{%
1614   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1615   \toks@\expandafter{\originalTeX\let#1=}%
1616   \bbl@exp{%
1617     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1618   \advance\babel@savecnt\@ne}
1619 \def\babel@savevariable#1{%
1620   \toks@\expandafter{\originalTeX #1=}%
1621   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

³¹`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\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\sfcode`\.=\@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{\sfcode`##1=\the\sfcode`##1\relax}%
1637   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1638 \def\bbl@post@fs{%
1639   \bbl@save@sfcodes
1640   \edef\bbl@tempa{\bbl@cl{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\sfcode`##1=##2\relax
1646         \babel@savevariable{\sfcode`##1}%
1647         \sfcode`##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\sfcode`##1=##3\relax
1653         \babel@savevariable{\sfcode`##1}%
1654         \sfcode`##1=##2\relax
1655       \fi}%
1656     \bbl@fs@chars
1657   \fi\fi\fi}

```

8.8 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 otherlanguage*\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\@@}}

```

8.9 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 0pt` 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

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

no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

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

```
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     \babeinullhyphen
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}
```

8.10 Multiencoding strings

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

Tools But first, a couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```
1737 \bbl@trace{Multiencoding strings}
1738 \def\bbl@tglobal#1{\global\let#1#1}
1739 \def\bbl@recatcode#1{% TODO. Used only once?
1740   \@tempcnta="7F
1741   \def\bbl@tempa{%
1742     \ifnum\@tempcnta>"FF\else
1743       \catcode\@tempcnta=#1\relax
1744       \advance\@tempcnta\@ne
1745       \expandafter\bbl@tempa
```

```

1746   \fi}%
1747   \bbl@tempa}

```

The second one. We need to patch `\@uclclist`, but it is done once and only if `\SetCase` is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact `\@uclclist` is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually `\reserved@a`), we pass it as argument to `\bbl@uclc`. The parser is restarted inside `\<lang>\bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

```
\let\bbl@tolower\@empty\bbl@toupper\@empty
```

and starts over (and similarly when lowercasing).

```

1748 \@ifpackagewith{babel}{nocase}%
1749   {\let\bbl@patchuclc\relax}%
1750   {\def\bbl@patchuclc{%
1751     \global\let\bbl@patchuclc\relax
1752     \g@addto\macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1753     \gdef\bbl@uclc##1{%
1754       \let\bbl@encoded\bbl@encoded@uclc
1755       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1756       {##1}%
1757       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1758         \csname\language @bbl@uclc\endcsname}%
1759       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1760     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1761     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}
1762 <<(*More package options)>> ≡
1763 \DeclareOption{nocase}{}
1764 <</More package options>>

```

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

```

1765 <<(*More package options)>> ≡
1766 \let\bbl@opt@strings\@nnil % accept strings=value
1767 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1768 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1769 \def\BabelStringsDefault{generic}
1770 <</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.

```

1771 \@onlypreamble\StartBabelCommands
1772 \def\StartBabelCommands{%
1773   \begingroup
1774   \bbl@recatcode{11}%
1775   <<Macros local to BabelCommands>>
1776   \def\bbl@provstring##1##2{%
1777     \providecommand##1{##2}%
1778     \bbl@tglobal##1}%
1779   \global\let\bbl@scafter\@empty
1780   \let\StartBabelCommands\bbl@startcmds
1781   \ifx\BabelLanguages\relax
1782     \let\BabelLanguages\CurrentOption
1783   \fi
1784   \begingroup

```

```

1785 \let\bbl@screaset@nnil % local flag - disable 1st stopcommands
1786 \StartBabelCommands}
1787 \def\bbl@startcmds{%
1788   \ifx\bbl@screaset@nnil\else
1789     \bbl@usehooks{stopcommands}{}%
1790   \fi
1791 \endgroup
1792 \begingroup
1793 \@ifstar
1794   {\ifx\bbl@opt@strings\@nnil
1795     \let\bbl@opt@strings\BabelStringsDefault
1796   \fi
1797   \bbl@startcmds@i}%
1798   \bbl@startcmds@i}
1799 \def\bbl@startcmds@i#1#2{%
1800   \edef\bbl@L{\zap@space#1 \@empty}%
1801   \edef\bbl@G{\zap@space#2 \@empty}%
1802   \bbl@startcmds@ii}
1803 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1804 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1805   \let\SetString@gobbletwo
1806   \let\bbl@stringdef@gobbletwo
1807   \let\AfterBabelCommands@gobble
1808   \ifx\@empty#1%
1809     \def\bbl@sc@label{generic}%
1810     \def\bbl@encstring##1##2{%
1811       \ProvideTextCommandDefault##1{##2}%
1812       \bbl@toglobal##1%
1813       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1814     \let\bbl@sctest\in@true
1815   \else
1816     \let\bbl@sc@charset\space % <- zapped below
1817     \let\bbl@sc@fontenc\space % <- " "
1818     \def\bbl@tempa##1=##2\@nil{%
1819       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1820     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1821     \def\bbl@tempa##1 ##2{% space -> comma
1822       ##1%
1823       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1824     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1825     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1826     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1827     \def\bbl@encstring##1##2{%
1828       \bbl@foreach\bbl@sc@fontenc{%
1829         \bbl@ifunset{T@###1}%
1830         {}%
1831         {\ProvideTextCommand##1{####1}{##2}%
1832         \bbl@toglobal##1%
1833         \expandafter

```

```

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

```

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

```

1863 \def\bbl@forlang#1#2{%
1864     \bbl@for#1\bbl@L{%
1865         \bbl@xin@{\, #1,}{\, \BabelLanguages,}%
1866         \ifin@#2\relax\fi}}
1867 \def\bbl@scswitch{%
1868     \bbl@forlang\bbl@tempa{%
1869         \ifx\bbl@G\@empty\else
1870             \ifx\SetString\@gobbletwo\else
1871                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1872                 \bbl@xin@{\, \bbl@GL,}{\, \bbl@screset,}%
1873                 \ifin@\else
1874                     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1875                     \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1876                 \fi
1877             \fi
1878         \fi}}
1879 \AtEndOfPackage{%
1880     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}}{\, #2}}}%
1881     \let\bbl@scswitch\relax}
1882 \onlypreamble\EndBabelCommands
1883 \def\EndBabelCommands{%

```

```

1884 \bbl@usehooks{stopcommands}{}%
1885 \endgroup
1886 \endgroup
1887 \bbl@scafter}
1888 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

Strings The following macro is the actual definition of \SetString when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like \providescommand). With the event stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

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

```

Now, some additional stuff to be used when encoded strings are used. Captions then include \bbl@encoded for string to be expanded in case transformations. It is \relax by default, but in \MakeUppercase and \MakeLowercase its value is a modified expandable \@changed@cmd.

```

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

```

Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current definition is somewhat complicated because we need a count, but \count@ is not under our control (remember \SetString may call hooks). Instead of defining a dedicated count, we just “pre-expand” its value.

```

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

```

```

1927         \count@=\the\count@\relax}}}%
1928 <</Macros local to BabelCommands>>

```

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

```

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

```

Case mapping The command `\SetCase` provides a way to change the behavior of `\MakeUppercase` and `\MakeLowercase`. `\bbl@tempa` is set by the patched `\@uclclist` to the parsing command.

```

1932 <<*Macros local to BabelCommands>> ≡
1933   \newcommand\SetCase[3][1]{%
1934     \bbl@patchuclc
1935     \bbl@forlang\bbl@tempa{%
1936       \expandafter\bbl@encstring
1937       \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1938       \expandafter\bbl@encstring
1939       \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1940       \expandafter\bbl@encstring
1941       \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1942 <</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.

```

1943 <<*Macros local to BabelCommands>> ≡
1944   \newcommand\SetHyphenMap[1]{%
1945     \bbl@forlang\bbl@tempa{%
1946       \expandafter\bbl@stringdef
1947       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1948 <</Macros local to BabelCommands>>

```

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

```

1949 \newcommand\BabelLower[2]{% one to one.
1950   \ifnum\lccode#1=#2\else
1951     \babel@savevariable{\lccode#1}%
1952     \lccode#1=#2\relax
1953   \fi}
1954 \newcommand\BabelLowerMM[4]{% many-to-many
1955   \@tempcnta=#1\relax
1956   \@tempcntb=#4\relax
1957   \def\bbl@tempa{%
1958     \ifnum\@tempcnta>#2\else
1959       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1960       \advance\@tempcnta#3\relax
1961       \advance\@tempcntb#3\relax
1962       \expandafter\bbl@tempa
1963     \fi}%
1964   \bbl@tempa}
1965 \newcommand\BabelLowerM0[4]{% many-to-one
1966   \@tempcnta=#1\relax
1967   \def\bbl@tempa{%
1968     \ifnum\@tempcnta>#2\else
1969       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1970       \advance\@tempcnta#3
1971       \expandafter\bbl@tempa
1972     \fi}%

```

```
1973 \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1974 <<*More package options>> ≡
1975 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1976 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap@ne}
1977 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1978 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1979 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1980 <</More package options>>
```

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

```
1981 \AtEndOfPackage{%
1982   \ifx\bbl@opt@hyphenmap\undefined
1983     \bbl@xin@{,}{\bbl@language@opts}%
1984     \chardef\bbl@opt@hyphenmap\ifin4\else\ne\fi
1985   \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.

```
1986 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
1987   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1988 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1989   \bbl@trim@def\bbl@tempa{#2}%
1990   \bbl@xin@{.template}{\bbl@tempa}%
1991   \ifin@
1992     \bbl@ini@captions@template{#3}{#1}%
1993   \else
1994     \edef\bbl@tempd{%
1995       \expandafter\expandafter\expandafter
1996       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1997     \bbl@xin@
1998       {\expandafter\string\csname #2name\endcsname}%
1999     {\bbl@tempd}%
2000   \ifin@ % Renew caption
2001     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2002     \ifin@
2003       \bbl@exp{%
2004         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2005         {\bbl@scset\<#2name>\<#1#2name>}%
2006         {}}%
2007       \else % Old way converts to new way
2008         \bbl@ifunset{#1#2name}%
2009         {\bbl@exp{%
2010           \\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2011           \\bbl@ifsamestring{\bbl@tempa}{\language}%
2012           {\def\<#2name>\<#1#2name>}}%
2013           {}}}%
2014       {}%
2015     \fi
2016   \else
2017     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2018     \ifin@ % New way
2019     \bbl@exp{%
2020       \\bbl@add\<captions#1>\bbl@scset\<#2name>\<#1#2name>}%
2021       \\bbl@ifsamestring{\bbl@tempa}{\language}%
2022       {\bbl@scset\<#2name>\<#1#2name>}%
2023       {}}%
```



```

2024 \else % Old way, but defined in the new way
2025 \bbl@exp{%
2026 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2027 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2028 {\def\<#2name>{\<#1#2name>}}%
2029 {}}%
2030 \fi%
2031 \fi
2032 \@namedef{#1#2name}{#3}%
2033 \toks@\expandafter{\bbl@captionslist}%
2034 \bbl@exp{\in{\<#2name>}{\the\toks@}}%
2035 \ifin@else
2036 \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2037 \bbl@global\bbl@captionslist
2038 \fi
2039 \fi}
2040 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

8.11 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.

```

2041 \bbl@trace{Macros related to glyphs}
2042 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2043 \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2044 \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.

```

2045 \def\save@sf@q#1{\leavevmode
2046 \begingroup
2047 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2048 \endgroup}

```

8.12 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`.

8.12.1 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.

```

2049 \ProvideTextCommand{\quotedblbase}{OT1}{%
2050 \save@sf@q{\set@low@box{\textquotedblright\}}%
2051 \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.

```

2052 \ProvideTextCommandDefault{\quotedblbase}{%
2053 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2054 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2055 \save@sf@q{\set@low@box{\textquoteright\}}%
2056 \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.

```

2057 \ProvideTextCommandDefault{\quotesinglbase}{%
2058 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2059 \ProvideTextCommand{\guillemetleft}{OT1}{%
2060   \ifmmode
2061     \ll
2062   \else
2063     \save@sf@q{\nobreak
2064       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2065   \fi}
2066 \ProvideTextCommand{\guillemetright}{OT1}{%
2067   \ifmmode
2068     \gg
2069   \else
2070     \save@sf@q{\nobreak
2071       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2072   \fi}
2073 \ProvideTextCommand{\guillemotleft}{OT1}{%
2074   \ifmmode
2075     \ll
2076   \else
2077     \save@sf@q{\nobreak
2078       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2079   \fi}
2080 \ProvideTextCommand{\guillemotright}{OT1}{%
2081   \ifmmode
2082     \gg
2083   \else
2084     \save@sf@q{\nobreak
2085       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2086   \fi}

```

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

```

2087 \ProvideTextCommandDefault{\guillemetleft}{%
2088   \UseTextSymbol{OT1}{\guillemetleft}}
2089 \ProvideTextCommandDefault{\guillemetright}{%
2090   \UseTextSymbol{OT1}{\guillemetright}}
2091 \ProvideTextCommandDefault{\guillemotleft}{%
2092   \UseTextSymbol{OT1}{\guillemotleft}}
2093 \ProvideTextCommandDefault{\guillemotright}{%
2094   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2095 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2096   \ifmmode
2097     <%
2098   \else
2099     \save@sf@q{\nobreak
2100       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2101   \fi}
2102 \ProvideTextCommand{\guilsinglright}{OT1}{%
2103   \ifmmode
2104     >%
2105   \else
2106     \save@sf@q{\nobreak
2107       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2108   \fi}

```

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

```

2109 \ProvideTextCommandDefault{\guilsinglleft}{%
2110 \UseTextSymbol{OT1}{\guilsinglleft}}
2111 \ProvideTextCommandDefault{\guilsinglright}{%
2112 \UseTextSymbol{OT1}{\guilsinglright}}

```

8.12.2 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.

```

2113 \DeclareTextCommand{\ij}{OT1}{%
2114 i\kern-0.02em\bbl@allowhyphens j}
2115 \DeclareTextCommand{\IJ}{OT1}{%
2116 I\kern-0.02em\bbl@allowhyphens J}
2117 \DeclareTextCommand{\ij}{T1}{\char188}
2118 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2119 \ProvideTextCommandDefault{\ij}{%
2120 \UseTextSymbol{OT1}{\ij}}
2121 \ProvideTextCommandDefault{\IJ}{%
2122 \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).

```

2123 \def\crrtic@{\hrule height0.1ex width0.3em}
2124 \def\crttic@{\hrule height0.1ex width0.33em}
2125 \def\ddj@{%
2126 \setbox0\hbox{d}\dimen@=\ht0
2127 \advance\dimen@1ex
2128 \dimen@.45\dimen@
2129 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2130 \advance\dimen@ii.5ex
2131 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2132 \def\DDJ@{%
2133 \setbox0\hbox{D}\dimen@=.55\ht0
2134 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2135 \advance\dimen@ii.15ex % correction for the dash position
2136 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2137 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2138 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2139 %
2140 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2141 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2142 \ProvideTextCommandDefault{\dj}{%
2143 \UseTextSymbol{OT1}{\dj}}
2144 \ProvideTextCommandDefault{\DJ}{%
2145 \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.

```

2146 \DeclareTextCommand{\SS}{OT1}{\SS}
2147 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

8.12.3 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`

```
2148 \ProvideTextCommandDefault{\glq}{%
2149 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2150 \ProvideTextCommand{\grq}{T1}{%
2151 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2152 \ProvideTextCommand{\grq}{TU}{%
2153 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2154 \ProvideTextCommand{\grq}{OT1}{%
2155 \save@sf@q{\kern-.0125em
2156 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2157 \kern.07em\relax}}
2158 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

`\grqq`

```
2159 \ProvideTextCommandDefault{\glqq}{%
2160 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2161 \ProvideTextCommand{\grqq}{T1}{%
2162 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2163 \ProvideTextCommand{\grqq}{TU}{%
2164 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2165 \ProvideTextCommand{\grqq}{OT1}{%
2166 \save@sf@q{\kern-.07em
2167 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2168 \kern.07em\relax}}
2169 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

`\frq`

```
2170 \ProvideTextCommandDefault{\flq}{%
2171 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2172 \ProvideTextCommandDefault{\frq}{%
2173 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

`\frqq`

```
2174 \ProvideTextCommandDefault{\flqq}{%
2175 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2176 \ProvideTextCommandDefault{\frqq}{%
2177 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

8.12.4 Umlauts and tremas

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

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

```
2178 \def\umlauthigh{%
```

```

2179 \def\bbl@umlauta##1{\leavevmode\bgroup%
2180 \expandafter\accent\csname\fontencoding dqpos\endcsname
2181 ##1\bbl@allowhyphens\egroup}%
2182 \let\bbl@umlaute\bbl@umlauta}
2183 \def\umlautlow{%
2184 \def\bbl@umlauta{\protect\lower@umlaut}}
2185 \def\umlautelow{%
2186 \def\bbl@umlaute{\protect\lower@umlaut}}
2187 \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.

```

2188 \expandafter\ifx\csname U@D\endcsname\relax
2189 \csname newdimen\endcsname\U@D
2190 \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.

```

2191 \def\lower@umlaut#1{%
2192 \leavevmode\bgroup
2193 \U@D 1ex%
2194 {\setbox\z@ \hbox{%
2195 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2196 \dimen@ -.45ex\advance\dimen@ \ht\z@
2197 \ifdim 1ex < \dimen@ \fontdimen5\font\dimen@ \fi}%
2198 \expandafter\accent\csname\fontencoding dqpos\endcsname
2199 \fontdimen5\font\U@D #1%
2200 \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).

```

2201 \AtBeginDocument{%
2202 \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2203 \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2204 \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2205 \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2206 \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2207 \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2208 \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2209 \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2210 \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2211 \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2212 \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`.

```

2213 \ifx\l@english\undefined
2214 \chardef\l@english\z@
2215 \fi

```

```

2216% The following is used to cancel rules in ini files (see Amharic).
2217 \ifx\l@unhyphenated\undefined
2218   \newlanguage\l@unhyphenated
2219 \fi

```

8.13 Layout

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

```

2220 \bbl@trace{Bidi layout}
2221 \providecommand\IfBabelLayout[3]{#3}%
2222 \newcommand\BabelPatchSection[1]{%
2223   \@ifundefined{#1}{}{%
2224     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2225     \@namedef{#1}{%
2226       \@ifstar{\bbl@presec@s{#1}}{%
2227         {\@dblarg{\bbl@presec@x{#1}}}}}%
2228 \def\bbl@presec@x#1[#2]#3{%
2229   \bbl@exp{%
2230     \\select@language@x{\bbl@main@language}%
2231     \\bbl@cs{sspre@#1}%
2232     \\bbl@cs{ss@#1}%
2233     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2234     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2235     \\select@language@x{\language}}}%
2236 \def\bbl@presec@s#1#2{%
2237   \bbl@exp{%
2238     \\select@language@x{\bbl@main@language}%
2239     \\bbl@cs{sspre@#1}%
2240     \\bbl@cs{ss@#1}*%
2241     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2242     \\select@language@x{\language}}}%
2243 \IfBabelLayout{sectioning}%
2244   {\BabelPatchSection{part}%
2245    \BabelPatchSection{chapter}%
2246    \BabelPatchSection{section}%
2247    \BabelPatchSection{subsection}%
2248    \BabelPatchSection{subsubsection}%
2249    \BabelPatchSection{paragraph}%
2250    \BabelPatchSection{subparagraph}%
2251    \def\babel@toc#1{%
2252      \select@language@x{\bbl@main@language}}}%
2253 \IfBabelLayout{captions}%
2254   {\BabelPatchSection{caption}}}%

```

8.14 Load engine specific macros

```

2255 \bbl@trace{Input engine specific macros}
2256 \ifcase\bbl@engine
2257   \input txtbabel.def
2258 \or
2259   \input luababel.def
2260 \or
2261   \input xebabel.def
2262 \fi

```

8.15 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.

```
2263 \bbl@trace{Creating languages and reading ini files}
2264 \let\bbl@extend@ini@gobble
2265 \newcommand\babelprovide[2][{%
2266   \let\bbl@savelangname\language
2267   \edef\bbl@savelocaleid{\the\localeid}%
2268   % Set name and locale id
2269   \edef\language{#2}%
2270   \bbl@id@assign
2271   % Initialize keys
2272   \let\bbl@KVP@captions\@nil
2273   \let\bbl@KVP@date\@nil
2274   \let\bbl@KVP@import\@nil
2275   \let\bbl@KVP@main\@nil
2276   \let\bbl@KVP@script\@nil
2277   \let\bbl@KVP@language\@nil
2278   \let\bbl@KVP@hyphenrules\@nil
2279   \let\bbl@KVP@linebreaking\@nil
2280   \let\bbl@KVP@justification\@nil
2281   \let\bbl@KVP@mapfont\@nil
2282   \let\bbl@KVP@maparabic\@nil
2283   \let\bbl@KVP@mapdigits\@nil
2284   \let\bbl@KVP@intraspace\@nil
2285   \let\bbl@KVP@intrapenalty\@nil
2286   \let\bbl@KVP@onchar\@nil
2287   \let\bbl@KVP@transforms\@nil
2288   \global\let\bbl@release@transforms\@empty
2289   \let\bbl@KVP@alph\@nil
2290   \let\bbl@KVP@Alph\@nil
2291   \let\bbl@KVP@labels\@nil
2292   \bbl@csarg\let{KVP@labels*}\@nil
2293   \global\let\bbl@inidata\@empty
2294   \global\let\bbl@extend@ini@gobble
2295   \gdef\bbl@key@list{;}%
2296   \bbl@forkv{#1}{% TODO - error handling
2297     \in@{/}{##1}%
2298     \ifin@
2299       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2300       \bbl@renewinikey##1\@{##2}%
2301     \else
2302       \bbl@csarg\def{KVP@##1}{##2}%
2303     \fi}%
2304   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2305   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel#2}\@ne\tw@}%
2306   % == init ==
2307   \ifx\bbl@screset\@undefined
2308     \bbl@ldfinit
2309   \fi
2310   % ==
2311   \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2312   \ifcase\bbl@howloaded
2313     \let\bbl@lbkflag\@empty % new
2314   \else
2315     \ifx\bbl@KVP@hyphenrules\@nil\else
2316       \let\bbl@lbkflag\@empty
2317     \fi
2318     \ifx\bbl@KVP@import\@nil\else
2319       \let\bbl@lbkflag\@empty
```

```

2320 \fi
2321 \fi
2322 % == import, captions ==
2323 \ifx\bbbl@KVP@import\@nil\else
2324 \bbbl@exp{\bbbl@ifblank{\bbbl@KVP@import}}%
2325 {\ifx\bbbl@initoload\relax
2326 \begingroup
2327 \def\BabelBeforeIni##1##2{\gdef\bbbl@KVP@import{##1}\endinput}%
2328 \bbbl@input@texini{##2}%
2329 \endgroup
2330 \else
2331 \xdef\bbbl@KVP@import{\bbbl@initoload}%
2332 \fi}%
2333 {}%
2334 \fi
2335 \ifx\bbbl@KVP@captions\@nil
2336 \let\bbbl@KVP@captions\bbbl@KVP@import
2337 \fi
2338 % ==
2339 \ifx\bbbl@KVP@transforms\@nil\else
2340 \bbbl@replace\bbbl@KVP@transforms{ },}%
2341 \fi
2342 % == Load ini ==
2343 \ifcase\bbbl@howloaded
2344 \bbbl@provide@new{##2}%
2345 \else
2346 \bbbl@ifblank{##1}%
2347 {}% With \bbbl@load@basic below
2348 {\bbbl@provide@renew{##2}}%
2349 \fi
2350 % Post tasks
2351 % -----
2352 % == subsequent calls after the first provide for a locale ==
2353 \ifx\bbbl@inidata\@empty\else
2354 \bbbl@extend@ini{##2}%
2355 \fi
2356 % == ensure captions ==
2357 \ifx\bbbl@KVP@captions\@nil\else
2358 \bbbl@ifunset{\bbbl@extracaps@##2}%
2359 {\bbbl@exp{\bbbl@babelensure[exclude=\\today]{##2}}}%
2360 {\bbbl@exp{\bbbl@babelensure[exclude=\\today,
2361 include=\bbbl@extracaps@##2]}{##2}}%
2362 \bbbl@ifunset{\bbbl@ensure@language}%
2363 {\bbbl@exp{%
2364 \\\DeclareRobustCommand\<\bbbl@ensure@language>[1]{%
2365 \\\foreignlanguage{language}%
2366 {###1}}}%
2367 }%
2368 \bbbl@exp{%
2369 \\\bbbl@tglobal\<\bbbl@ensure@language>%
2370 \\\bbbl@tglobal\<\bbbl@ensure@language\space>%
2371 \fi
2372 % ==
2373 % At this point all parameters are defined if 'import'. Now we
2374 % execute some code depending on them. But what about if nothing was
2375 % imported? We just set the basic parameters, but still loading the
2376 % whole ini file.
2377 \bbbl@load@basic{##2}%
2378 % == script, language ==

```



```

2379 % Override the values from ini or defines them
2380 \ifx\bb1@KVP@script\@nil\else
2381   \bb1@csarg\edef\sname@#2}{\bb1@KVP@script}%
2382 \fi
2383 \ifx\bb1@KVP@language\@nil\else
2384   \bb1@csarg\edef\lname@#2}{\bb1@KVP@language}%
2385 \fi
2386 % == onchar ==
2387 \ifx\bb1@KVP@onchar\@nil\else
2388   \bb1@luahyphenate
2389   \directlua{
2390     if Babel.locale_mapped == nil then
2391       Babel.locale_mapped = true
2392       Babel.linebreaking.add_before(Babel.locale_map)
2393       Babel.loc_to_scr = {}
2394       Babel.chr_to_loc = Babel.chr_to_loc or {}
2395     end}%
2396 \bb1@xin@{ ids }{ \bb1@KVP@onchar\space}%
2397 \ifin@
2398   \ifx\bb1@starthyphens\@undefined % Needed if no explicit selection
2399     \AddBabelHook{babel-onchar}{beforestart}{\bb1@starthyphens}%
2400   \fi
2401   \bb1@exp{\bb1@add\bb1@starthyphens
2402     {\bb1@patterns@lua{\language}}}%
2403   % TODO - error/warning if no script
2404   \directlua{
2405     if Babel.script_blocks['\bb1@cl{sbc}'] then
2406       Babel.loc_to_scr[\the\localeid] =
2407         Babel.script_blocks['\bb1@cl{sbc}']
2408       Babel.locale_props[\the\localeid].lc = \the\localeid\space
2409       Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language}\space
2410     end
2411   }%
2412 \fi
2413 \bb1@xin@{ fonts }{ \bb1@KVP@onchar\space}%
2414 \ifin@
2415   \bb1@ifunset{bb1@lsys@\language}{\bb1@provide@lsys{\language}}}%
2416   \bb1@ifunset{bb1@wdir@\language}{\bb1@provide@dirs{\language}}}%
2417   \directlua{
2418     if Babel.script_blocks['\bb1@cl{sbc}'] then
2419       Babel.loc_to_scr[\the\localeid] =
2420         Babel.script_blocks['\bb1@cl{sbc}']
2421     end}%
2422 \ifx\bb1@mapselect\@undefined % TODO. almost the same as mapfont
2423   \AtBeginDocument{%
2424     \bb1@patchfont{\bb1@mapselect}%
2425     {\selectfont}}%
2426   \def\bb1@mapselect{%
2427     \let\bb1@mapselect\relax
2428     \edef\bb1@prefontid{\fontid\font}}%
2429   \def\bb1@mapdir##1{%
2430     {\def\language{##1}%
2431       \let\bb1@ifrestoring\@firstoftwo % To avoid font warning
2432       \bb1@switchfont
2433       \directlua{
2434         Babel.locale_props[\the\csname bb1@id@##1\endcsname]
2435           [\bb1@prefontid] = \fontid\font\space}}}%
2436   \fi
2437   \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%

```

```

2438 \fi
2439 % TODO - catch non-valid values
2440 \fi
2441 % == mapfont ==
2442 % For bidi texts, to switch the font based on direction
2443 \ifx\bbbl@KVP@mapfont\@nil\else
2444 \bbbl@ifsamestring{\bbbl@KVP@mapfont}{direction}}}%
2445 {\bbbl@error{Option '\bbbl@KVP@mapfont' unknown for\%
2446 mapfont. Use 'direction'.%
2447 {See the manual for details.}}}%
2448 \bbbl@ifunset{\bbbl@lsys@\language}\bbbl@provide@lsys{\language}}}%
2449 \bbbl@ifunset{\bbbl@wdir@\language}\bbbl@provide@dirs{\language}}}%
2450 \ifx\bbbl@mapselect\@undefined % TODO. See onchar.
2451 \AtBeginDocument{%
2452 \bbbl@patchfont{\bbbl@mapselect}}%
2453 {\selectfont}}%
2454 \def\bbbl@mapselect{%
2455 \let\bbbl@mapselect\relax
2456 \edef\bbbl@prefontid{\fontid\font}}%
2457 \def\bbbl@mapdir##1{%
2458 {\def\language##1}%
2459 \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
2460 \bbbl@switchfont
2461 \directlua{Babel.fontmap
2462 [\the\csname bbl@wdir@##1\endcsname]%
2463 [\bbbl@prefontid]=\fontid\font}}}%
2464 \fi
2465 \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
2466 \fi
2467 % == Line breaking: intraspace, intrapenalty ==
2468 % For CJK, East Asian, Southeast Asian, if interspace in ini
2469 \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
2470 \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2471 \fi
2472 \bbbl@provide@intraspace
2473 % == Line breaking: CJK quotes ==
2474 \ifcase\bbbl@engine\or
2475 \bbbl@xin{/c}{\bbbl@c1{lnbrk}}}%
2476 \ifin@
2477 \bbbl@ifunset{\bbbl@quote@\language}}}%
2478 {\directlua{
2479 Babel.locale_props[\the\localeid].cjk_quotes = {}
2480 local cs = 'op'
2481 for c in string.utfvalues(%
2482 [[\csname bbl@quote@\language\endcsname]]) do
2483 if Babel.cjk_characters[c].c == 'qu' then
2484 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2485 end
2486 cs = ( cs == 'op') and 'cl' or 'op'
2487 end
2488 }}%
2489 \fi
2490 \fi
2491 % == Line breaking: justification ==
2492 \ifx\bbbl@KVP@justification\@nil\else
2493 \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2494 \fi
2495 \ifx\bbbl@KVP@linebreaking\@nil\else
2496 \bbbl@xin{,\bbbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%

```

```

2497 \ifin@
2498 \bbl@csarg\xdef
2499 {lnbrk@\language\name}\expandafter\car\bbl@KVP@linebreaking\@nil}%
2500 \fi
2501 \fi
2502 \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
2503 \ifin\else\bbl@xin@{/k}{/\bbl@c1{lnbrk}}\fi
2504 \ifin\bbl@arabicjust\fi
2505 % == Line breaking: hyphenate.other.(locale|script) ==
2506 \ifx\bbl@lbkflag\@empty
2507 \bbl@ifunset\bbl@hyotl@\language\name}{}%
2508 {\bbl@csarg\bbl@replace{hyotl@\language\name}{ }{,}%
2509 \bbl@startcommands*\language\name}{}%
2510 \bbl@csarg\bbl@foreach{hyotl@\language\name}{%
2511 \ifcase\bbl@engine
2512 \ifnum##1<257
2513 \SetHyphenMap{BabelLower{##1}{##1}}%
2514 \fi
2515 \else
2516 \SetHyphenMap{BabelLower{##1}{##1}}%
2517 \fi}%
2518 \bbl@endcommands}%
2519 \bbl@ifunset\bbl@hyots@\language\name}{}%
2520 {\bbl@csarg\bbl@replace{hyots@\language\name}{ }{,}%
2521 \bbl@csarg\bbl@foreach{hyots@\language\name}{%
2522 \ifcase\bbl@engine
2523 \ifnum##1<257
2524 \global\lccode##1=##1\relax
2525 \fi
2526 \else
2527 \global\lccode##1=##1\relax
2528 \fi}}%
2529 \fi
2530 % == Counters: maparabic ==
2531 % Native digits, if provided in ini (TeX level, xe and lua)
2532 \ifcase\bbl@engine\else
2533 \bbl@ifunset\bbl@dgnat@\language\name}{%
2534 {\expandafter\ifx\csname bbl@dgnat@\language\name\endcsname\@empty\else
2535 \expandafter\expandafter\expandafter
2536 \bbl@setdigits\csname bbl@dgnat@\language\name\endcsname
2537 \ifx\bbl@KVP@maparabic\@nil\else
2538 \ifx\bbl@latinarabic\@undefined
2539 \expandafter\let\expandafter\@arabic
2540 \csname bbl@counter@\language\name\endcsname
2541 \else % ie, if layout=counters, which redefines \@arabic
2542 \expandafter\let\expandafter\bbl@latinarabic
2543 \csname bbl@counter@\language\name\endcsname
2544 \fi
2545 \fi
2546 \fi}%
2547 \fi
2548 % == Counters: mapdigits ==
2549 % Native digits (lua level).
2550 \ifodd\bbl@engine
2551 \ifx\bbl@KVP@mapdigits\@nil\else
2552 \bbl@ifunset\bbl@dgnat@\language\name}{%
2553 {\RequirePackage{luatexbase}%
2554 \bbl@activate@preotf
2555 \directlua{

```

```

2556     Babel = Babel or {} %% -> presets in luababel
2557     Babel.digits_mapped = true
2558     Babel.digits = Babel.digits or {}
2559     Babel.digits[\the\localeid] =
2560         table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2561     if not Babel.numbers then
2562         function Babel.numbers(head)
2563             local LOCALE = Babel.attr_locale
2564             local GLYPH = node.id'glyph'
2565             local inmath = false
2566             for item in node.traverse(head) do
2567                 if not inmath and item.id == GLYPH then
2568                     local temp = node.get_attribute(item, LOCALE)
2569                     if Babel.digits[temp] then
2570                         local chr = item.char
2571                         if chr > 47 and chr < 58 then
2572                             item.char = Babel.digits[temp][chr-47]
2573                         end
2574                     end
2575                     elseif item.id == node.id'math' then
2576                         inmath = (item.subtype == 0)
2577                     end
2578                 end
2579             return head
2580         end
2581     end
2582     }}%
2583 \fi
2584 \fi
2585 % == Counters: alph, Alph ==
2586 % What if extras<lang> contains a \babel@save\@alph? It won't be
2587 % restored correctly when exiting the language, so we ignore
2588 % this change with the \bbl@alph@saved trick.
2589 \ifx\bbl@KVP@alph\@nil\else
2590     \bbl@extras@wrap{\bbl@alph@saved}%
2591     {\let\bbl@alph@saved\@alph}%
2592     {\let\@alph\bbl@alph@saved
2593     \babel@save\@alph}%
2594     \bbl@exp{%
2595         \bbl@add\<extras\language\>%
2596         \let\@alph<bbl@cntr@bbl@KVP@alph @\language\>}}%
2597 \fi
2598 \ifx\bbl@KVP@Alph\@nil\else
2599     \bbl@extras@wrap{\bbl@Alph@saved}%
2600     {\let\bbl@Alph@saved\@Alph}%
2601     {\let\@Alph\bbl@Alph@saved
2602     \babel@save\@Alph}%
2603     \bbl@exp{%
2604         \bbl@add\<extras\language\>%
2605         \let\@Alph<bbl@cntr@bbl@KVP@Alph @\language\>}}%
2606 \fi
2607 % == require.babel in ini ==
2608 % To load or reload the babel-*.tex, if require.babel in ini
2609 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2610     \bbl@ifunset{bbl@rqtex\@language\>}%
2611     {\expandafter\ifx\csname bbl@rqtex\@language\endcsname\@empty\else
2612         \let\BabelBeforeIni@gobbletwo
2613         \chardef\atcatcode=\catcode`\@
2614         \catcode`\@=11\relax

```

```

2615         \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2616         \catcode`\@=\atcatcode
2617         \let\atcatcode\relax
2618         \global\bbl@csarg\let{rqtex@\language}\relax
2619     \fi}%
2620 \fi
2621 % == frenchspacing ==
2622 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2623 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2624 \ifin@
2625     \bbl@extras@wrap{\bbl@pre@fs}%
2626     {\bbl@pre@fs}%
2627     {\bbl@post@fs}%
2628 \fi
2629 % == Release saved transforms ==
2630 \bbl@release@transforms\relax % \relax closes the last item.
2631 % == main ==
2632 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
2633     \let\language\bbl@savelangname
2634     \chardef\localeid\bbl@savelocaleid\relax
2635 \fi}

```

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

```

2636 \def\bbl@provide@new#1{%
2637     \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2638     \@namedef{extras#1}{}%
2639     \@namedef{noextras#1}{}%
2640     \bbl@startcommands*{#1}{captions}%
2641     \ifx\bbl@KVP@captions\@nil % and also if import, implicit
2642         \def\bbl@tempb##1{% elt for \bbl@captionslist
2643             \ifx##1\@empty\else
2644                 \bbl@exp{%
2645                     \\SetString\\##1{%
2646                         \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2647                 \expandafter\bbl@tempb
2648             \fi}%
2649         \expandafter\bbl@tempb\bbl@captionslist\@empty
2650     \else
2651         \ifx\bbl@initoload\relax
2652             \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2653         \else
2654             \bbl@read@ini{\bbl@initoload}2% % Same
2655         \fi
2656     \fi
2657     \StartBabelCommands*{#1}{date}%
2658     \ifx\bbl@KVP@import\@nil
2659         \bbl@exp{%
2660             \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2661     \else
2662         \bbl@savetoday
2663         \bbl@savedate
2664     \fi
2665     \bbl@endcommands
2666     \bbl@load@basic{#1}%
2667     % == hyphenmins == (only if new)
2668     \bbl@exp{%
2669         \gdef\<#1hyphenmins>{%
2670             {\bbl@ifunset{\bbl@ifthm@#1}{2}{\bbl@cs{lfthm@#1}}}%

```

```

2671     {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2672 % == hyphenrules (also in renew) ==
2673 \bbl@provide@hyphens{#1}%
2674 \ifx\bbl@KVP@main\@nil\else
2675   \expandafter\main@language\expandafter{#1}%
2676 \fi}
2677 %
2678 \def\bbl@provide@renew#1{%
2679   \ifx\bbl@KVP@captions\@nil\else
2680     \StartBabelCommands*{#1}{captions}%
2681     \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2682     \EndBabelCommands
2683   \fi
2684   \ifx\bbl@KVP@import\@nil\else
2685     \StartBabelCommands*{#1}{date}%
2686     \bbl@savetoday
2687     \bbl@savestate
2688     \EndBabelCommands
2689   \fi
2690 % == hyphenrules (also in new) ==
2691 \ifx\bbl@lbfkflag\@empty
2692   \bbl@provide@hyphens{#1}%
2693 \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.)

```

2694 \def\bbl@load@basic#1{%
2695   \ifcase\bbl@howloaded\or\or
2696     \ifcase\csname bbl@llevel@\language\endcsname
2697       \bbl@csarg\let{lname@\language}\relax
2698     \fi
2699   \fi
2700   \bbl@ifunset{\bbl@lname@#1}%
2701   {\def\BabelBeforeIni##1##2{%
2702     \begingroup
2703       \let\bbl@ini@captions@aux\@gobbletwo
2704       \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
2705       \bbl@read@ini{##1}1%
2706       \ifx\bbl@initoload\relax\endinput\fi
2707     \endgroup}%
2708   \begingroup      % boxed, to avoid extra spaces:
2709     \ifx\bbl@initoload\relax
2710       \bbl@input@texini{#1}%
2711     \else
2712       \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2713     \fi
2714   \endgroup}%
2715   {}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2716 \def\bbl@provide@hyphens#1{%
2717   \let\bbl@tempa\relax
2718   \ifx\bbl@KVP@hyphenrules\@nil\else
2719     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2720     \bbl@foreach\bbl@KVP@hyphenrules{%
2721       \ifx\bbl@tempa\relax      % if not yet found
2722         \bbl@ifsamestring{##1}{+}%
2723         {\bbl@exp{\addlanguage\<1@##1>}}}%

```

```

2724     {}%
2725     \bbl@ifunset{l@##1}%
2726     {}%
2727     {\bbl@exp{\let\bbl@tempa<l@##1>}}%
2728     \fi}%
2729 \fi
2730 \ifx\bbl@tempa\relax %           if no opt or no language in opt found
2731     \ifx\bbl@KVP@import\@nil
2732         \ifx\bbl@initoload\relax\else
2733             \bbl@exp{%           and hyphenrules is not empty
2734                 \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2735                 {}%
2736                 {\let\\bbl@tempa<l@bbl@cl{hyphr}>}}%
2737             \fi
2738         \else % if importing
2739             \bbl@exp{%           and hyphenrules is not empty
2740                 \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2741                 {}%
2742                 {\let\\bbl@tempa<l@bbl@cl{hyphr}>}}%
2743             \fi
2744         \fi
2745     \bbl@ifunset{bbl@tempa}%     ie, relax or undefined
2746     {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
2747         {\bbl@exp{\\adddialect<l@#1>\language}}%
2748         {}}%                     so, l@<lang> is ok - nothing to do
2749     {\bbl@exp{\\adddialect<l@#1>\bbl@tempa}}% found in opt list or ini

```

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

```

2750 \def\bbl@input@texini#1{%
2751     \bbl@bsphack
2752     \bbl@exp{%
2753         \catcode`\\%=14 \catcode`\\=0
2754         \catcode`\\{=1 \catcode`\\}=2
2755         \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2756         \catcode`\\%=\\the\catcode`\%\\relax
2757         \catcode`\\=\\the\catcode`\\%\\relax
2758         \catcode`\\{=\\the\catcode`\%\\relax
2759         \catcode`\\}=\\the\catcode`\%\\relax}%
2760     \bbl@esphack}

```

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

```

2761 \def\bbl@inline#1\bbl@inline{%
2762     \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2763 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2764 \def\bbl@iniskip#1\@@{%           if starts with ;
2765 \def\bbl@inistore#1=#2\@@{%       full (default)
2766     \bbl@trim@def\bbl@tempa{#1}%
2767     \bbl@trim\toks@{#2}%
2768     \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2769     \ifin@else
2770         \bbl@exp{%
2771             \\g@addto@macro\\bbl@inidata{%
2772                 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}%
2773             \fi}
2774 \def\bbl@inistore@min#1=#2\@@{%   minimal (maybe set in \bbl@read@ini)
2775     \bbl@trim@def\bbl@tempa{#1}%
2776     \bbl@trim\toks@{#2}%

```

```

2777 \bbl@xin@{.identification.}{.\bbl@section.}%
2778 \ifin@
2779 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2780 \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2781 \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.

```

2782 \ifx\bbl@readstream\undefined
2783 \csname newread\endcsname\bbl@readstream
2784 \fi
2785 \def\bbl@read@ini#1#2{%
2786 \global\let\bbl@extend@ini\@gobble
2787 \openin\bbl@readstream=babel-#1.ini
2788 \ifeof\bbl@readstream
2789 \bbl@error
2790 {There is no ini file for the requested language\\%
2791 (#1). Perhaps you misspelled it or your installation\\%
2792 is not complete.}%
2793 {Fix the name or reinstall babel.}%
2794 \else
2795 % == Store ini data in \bbl@inidata ==
2796 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2797 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2798 \bbl@info{Importing
2799 \ifcase#2font and identification \or basic \fi
2800 data for \language\\%
2801 from babel-#1.ini. Reported}%
2802 \ifnum#2=\z@
2803 \global\let\bbl@inidata\@empty
2804 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2805 \fi
2806 \def\bbl@section{identification}%
2807 \bbl@exp{\\bbl@inistore tag.ini=#1\\@}%
2808 \bbl@inistore load.level=#2@@
2809 \loop
2810 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2811 \endlinechar\m@ne
2812 \read\bbl@readstream to \bbl@line
2813 \endlinechar\^^M
2814 \ifx\bbl@line\empty\else
2815 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2816 \fi
2817 \repeat
2818 % == Process stored data ==
2819 \bbl@csarg\xdef{lini@language}{#1}%
2820 \bbl@read@ini@aux
2821 % == 'Export' data ==
2822 \bbl@ini@exports{#2}%
2823 \global\bbl@csarg\let{inidata@language}\bbl@inidata
2824 \global\let\bbl@inidata\@empty
2825 \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{language}}%
2826 \bbl@tglobal\bbl@ini@loaded
2827 \fi}
2828 \def\bbl@read@ini@aux{%

```



```

2829 \let\bbl@savestrings\@empty
2830 \let\bbl@savetoday\@empty
2831 \let\bbl@savedate\@empty
2832 \def\bbl@elt##1##2##3{%
2833   \def\bbl@section{##1}%
2834   \in@{=date.}{=##1}% Find a better place
2835   \ifin@
2836     \bbl@ini@calendar{##1}%
2837   \fi
2838   \bbl@ifunset{bbl@inikv@##1}{}%
2839   {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2840 \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.

```

2841 \def\bbl@extend@ini@aux#1{%
2842   \bbl@startcommands*{#1}{captions}%
2843   % Activate captions/... and modify exports
2844   \bbl@csarg\def{inikv@captions.licr}##1##2{%
2845     \setlocalecaption{#1}{##1}{##2}}%
2846   \def\bbl@inikv@captions##1##2{%
2847     \bbl@ini@captions@aux{##1}{##2}}%
2848   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2849   \def\bbl@exportkey##1##2##3{%
2850     \bbl@ifunset{bbl@kv@##2}{}%
2851     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2852       \bbl@exp{\global\let<bbl@##1@<language>\<bbl@kv@##2>}%
2853       \fi}}%
2854   % As with \bbl@read@ini, but with some changes
2855   \bbl@read@ini@aux
2856   \bbl@ini@exports\tw@
2857   % Update inidata@lang by pretending the ini is read.
2858   \def\bbl@elt##1##2##3{%
2859     \def\bbl@section{##1}%
2860     \bbl@iniline##2=##3\bbl@iniline}%
2861     \csname bbl@inidata@#1\endcsname
2862     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2863     \StartBabelCommands*{#1}{date}% And from the import stuff
2864     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2865     \bbl@savetoday
2866     \bbl@savedate
2867     \bbl@endcommands}

```

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

```

2868 \def\bbl@ini@calendar#1{%
2869   \lowercase{\def\bbl@tempa{=##1=}}%
2870   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2871   \bbl@replace\bbl@tempa{=date.}{}%
2872   \in@{.licr=}{#1=}%
2873   \ifin@
2874     \ifcase\bbl@engine
2875       \bbl@replace\bbl@tempa{.licr=}{}%
2876     \else
2877       \let\bbl@tempa\relax
2878     \fi
2879   \fi
2880   \ifx\bbl@tempa\relax\else
2881     \bbl@replace\bbl@tempa{=}{}%
2882     \bbl@exp{%

```

```

2883 \def\<bbl@inikv@#1>###1###2{%
2884   \<bbl@inidate###1...\relax{###2}\<bbl@tempa}\}%
2885 \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).

```

2886 \def\bbl@renewinikey#1/#2\@#3{%
2887   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2888   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2889   \bbl@trim\toks@{#3}% value
2890   \bbl@exp{%
2891     \edef\<bbl@key@list\<bbl@key@list \bbl@tempa/\bbl@tempb;\}%
2892     \<g@addto@macro\<bbl@inidata{%
2893       \<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.

```

2894 \def\bbl@exportkey#1#2#3{%
2895   \bbl@ifunset{bbl@kv@#2}%
2896   {\bbl@csarg\gdef{#1\<language\>\<#3\>\>%
2897    {\<expandafter\<ifx\<csname \bbl@kv@#2\>\<endcsname\>\<empty
2898     \bbl@csarg\gdef{#1\<language\>\<#3\>\>%
2899     \else
2900     \bbl@exp{\<global\<let\<bbl@#1\<language\>\<bbl@kv@#2\>\>%
2901     \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.

```

2902 \def\bbl@iniwarning#1{%
2903   \bbl@ifunset{bbl@kv@identification.warning#1}\>%
2904   {\bbl@warning{%
2905     From babel-\<bbl@cs\<lini@language\>\<ini\>\<%
2906     \bbl@cs\<kv@identification.warning#1\>\<%
2907     Reported }}%
2908 %
2909 \let\bbl@release@transforms\<empty
2910 %
2911 \def\bbl@ini@exports#1{%
2912   % Identification always exported
2913   \bbl@iniwarning}%
2914   \ifcase\bbl@engine
2915     \bbl@iniwarning{\<pdf\<latex}%
2916   \or
2917     \bbl@iniwarning{\<lual\<tex}%
2918   \or
2919     \bbl@iniwarning{\<xel\<tex}%
2920   \fi%
2921   \bbl@exportkey{lllevel}\<identification.load.level\>\>%
2922   \bbl@exportkey{elname}\<identification.name.english\>\>%
2923   \bbl@exp{\<bbl@exportkey\<lname\>\<identification.name.opentype\>%
2924     {\<csname \bbl@elname\<language\>\<endcsname\>\>%
2925     \bbl@exportkey{tbcp}\<identification.tag.bcp47\>\>%
2926     \bbl@exportkey{lbcpl\<identification.language.tag.bcp47\>\>%
2927     \bbl@exportkey{lotf\>\<identification.tag.opentype\>\<dflt\>\>%
2928     \bbl@exportkey{esname\>\<identification.script.name\>\>%
2929     \bbl@exp{\<bbl@exportkey\<sname\>\<identification.script.name.opentype\>%

```

```

2930   {\csname bbl@esname@\language\endcsname}}%
2931   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2932   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2933   % Also maps bcp47 -> language
2934   \ifbbl@bcptoname
2935     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
2936   \fi
2937   % Conditional
2938   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
2939     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2940     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2941     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2942     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2943     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2944     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2945     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2946     \bbl@exportkey{intsp}{typography.intraspace}{}%
2947     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2948     \bbl@exportkey{chrng}{characters.ranges}{}%
2949     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2950     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2951     \ifnum#1=\tw@       % only (re)new
2952       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2953       \bbl@tglobal\bbl@savetoday
2954       \bbl@tglobal\bbl@savestate
2955       \bbl@savestrings
2956     \fi
2957   \fi}

```

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

```

2958 \def\bbl@inikv#1#2{%      key=value
2959   \toks@{#2}%              This hides #'s from ini values
2960   \bbl@csarg\xdef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2961 \let\bbl@inikv@identification\bbl@inikv
2962 \let\bbl@inikv@typography\bbl@inikv
2963 \let\bbl@inikv@characters\bbl@inikv
2964 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

2965 \def\bbl@inikv@counters#1#2{%
2966   \bbl@ifsamestring{#1}{digits}%
2967   {\bbl@error{The counter name 'digits' is reserved for mapping\\
2968     decimal digits}%
2969     {Use another name.}}%
2970   }%
2971   \def\bbl@tempc{#1}%
2972   \bbl@trim@def{\bbl@tempb*}{#2}%
2973   \in@{.1$}{#1$}%
2974   \ifin@
2975     \bbl@replace\bbl@tempc{.1}{}%
2976     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
2977       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2978   \fi
2979   \in@{.F.}{#1}%
2980   \ifin@else\in@{.S.}{#1}\fi

```

```

2981 \ifin@
2982   \bbl@csarg\protected@xdef{cntr@#1@\language}\bbl@tempb*}%
2983 \else
2984   \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2985   \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2986   \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
2987 \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.

```

2988 \ifcase\bbl@engine
2989   \bbl@csarg\def{inikv@captions.licr}#1#2{%
2990     \bbl@ini@captions@aux{#1}{#2}}
2991 \else
2992   \def\bbl@inikv@captions#1#2{%
2993     \bbl@ini@captions@aux{#1}{#2}}
2994 \fi

```

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

```

2995 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2996   \bbl@replace\bbl@tempa{.template}{}%
2997   \def\bbl@toreplace{#1}{}%
2998   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
2999   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3000   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3001   \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3002   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3003   \bbl@xin@{, \bbl@tempa,}{, chapter, appendix, part,}%
3004 \ifin@
3005   \@nameuse{\bbl@patch\bbl@tempa}%
3006   \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3007 \fi
3008 \bbl@xin@{, \bbl@tempa,}{, figure, table,}%
3009 \ifin@
3010   \toks@\expandafter{\bbl@toreplace}%
3011   \bbl@exp{\gdef\<fnun@\bbl@tempa>{\the\toks@}}%
3012 \fi}
3013 \def\bbl@ini@captions@aux#1#2{%
3014   \bbl@trim@def\bbl@tempa{#1}%
3015   \bbl@xin@{.template}{\bbl@tempa}%
3016 \ifin@
3017   \bbl@ini@captions@template{#2}\language
3018 \else
3019   \bbl@ifblank{#2}%
3020     {\bbl@exp{%
3021       \toks@{\ \ \bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3022     {\bbl@trim\toks@{#2}}}%
3023   \bbl@exp{%
3024     \ \bbl@add\ \bbl@savestrings{%
3025       \ \SetString\<\bbl@tempa name>{\the\toks@}}}%
3026   \toks@\expandafter{\bbl@captionslist}%
3027   \bbl@exp{\ \in@{\<\bbl@tempa name>}{\the\toks@}}%
3028 \ifin@ \else
3029   \bbl@exp{%
3030     \ \bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3031     \ \bbl@toglobal\<\bbl@extracaps@\language>}%
3032 \fi
3033 \fi}

```

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

```

3034 \def\bbl@list@the{%
3035   part,chapter,section,subsection,subsubsection,paragraph,%
3036   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3037   table,page,footnote,mpfootnote,mpfn}
3038 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3039   \bbl@ifunset{bbl@map@#1@\languagename}%
3040     {\@nameuse{#1}}%
3041     {\@nameuse{bbl@map@#1@\languagename}}}%
3042 \def\bbl@inikv@labels#1#2{%
3043   \in@{.map}{#1}%
3044   \ifin@
3045     \ifx\bbl@KVP@labels\@nil\else
3046       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3047       \ifin@
3048         \def\bbl@tempc{#1}%
3049         \bbl@replace\bbl@tempc{.map}{}%
3050         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3051         \bbl@exp{%
3052           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3053             {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3054         \bbl@foreach\bbl@list@the{%
3055           \bbl@ifunset{the##1}{}%
3056           {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3057             \bbl@exp{%
3058               \\bbl@sreplace\<the##1>%
3059               {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3060               \\bbl@sreplace\<the##1>%
3061               {\<\empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3062             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3063               \toks@\expandafter\expandafter\expandafter{%
3064                 \csname the##1\endcsname}%
3065               \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3066               \fi}}}%
3067   \fi
3068   \fi
3069   %
3070   \else
3071     %
3072     % The following code is still under study. You can test it and make
3073     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3074     % language dependent.
3075     \in@{enumerate.}{#1}%
3076     \ifin@
3077       \def\bbl@tempa{#1}%
3078       \bbl@replace\bbl@tempa{enumerate.}{}%
3079       \def\bbl@toreplace{#2}%
3080       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3081       \bbl@replace\bbl@toreplace{{}}{\csname the}%
3082       \bbl@replace\bbl@toreplace{}}{\endcsname{}}}%
3083       \toks@\expandafter{\bbl@toreplace}%
3084       % TODO. Execute only once:
3085       \bbl@exp{%
3086         \\bbl@add\<extras\languagename>%
3087         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3088         \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3089         \\bbl@tglobal\<extras\languagename>%
3090     \fi

```

```
3091 \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.

```
3092 \def\bbl@chapttype{chapter}
3093 \ifx\@makechapterhead\undefined
3094 \let\bbl@patchchapter\relax
3095 \else\ifx\thechapter\undefined
3096 \let\bbl@patchchapter\relax
3097 \else\ifx\ps@headings\undefined
3098 \let\bbl@patchchapter\relax
3099 \else
3100 \def\bbl@patchchapter{%
3101 \global\let\bbl@patchchapter\relax
3102 \gdef\bbl@chfmt{%
3103 \bbl@ifunset{bbl@\bbl@chapttype fmt@\languagename}%
3104 {\@chapapp\space\thechapter}
3105 {\@nameuse{bbl@\bbl@chapttype fmt@\languagename}}}%
3106 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3107 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3108 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3109 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3110 \bbl@tglobal\appendix
3111 \bbl@tglobal\ps@headings
3112 \bbl@tglobal\chaptermark
3113 \bbl@tglobal\@makechapterhead}
3114 \let\bbl@patchappendix\bbl@patchchapter
3115 \fi\fi\fi
3116 \ifx\@part\undefined
3117 \let\bbl@patchpart\relax
3118 \else
3119 \def\bbl@patchpart{%
3120 \global\let\bbl@patchpart\relax
3121 \gdef\bbl@partformat{%
3122 \bbl@ifunset{bbl@partfmt@\languagename}%
3123 {\partname\nobreakspace\thepart}
3124 {\@nameuse{bbl@partfmt@\languagename}}}%
3125 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3126 \bbl@tglobal\@part}
3127 \fi
```

Date. TODO. Document

```
3128% Arguments are _not_ protected.
3129 \let\bbl@calendar\@empty
3130 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3131 \def\bbl@localedate#1#2#3#4{%
3132 \begingroup
3133 \ifx\@empty#1\@empty\else
3134 \let\bbl@ld@calendar\@empty
3135 \let\bbl@ld@variant\@empty
3136 \edef\bbl@tempa{\zap@space#1 \@empty}%
3137 \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld##1}{##2}}%
3138 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3139 \edef\bbl@calendar{%
3140 \bbl@ld@calendar
3141 \ifx\bbl@ld@variant\@empty\else
3142 .\bbl@ld@variant
```

```

3143     \fi}%
3144     \bbl@replace\bbl@calendar{gregorian}{}%
3145     \fi
3146     \bbl@cased
3147     {\@nameuse\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3148 \endgroup}
3149 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3150 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3151 \bbl@trim@def\bbl@tempa{#1.#2}%
3152 \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3153 {\bbl@trim@def\bbl@tempa{#3}%
3154 \bbl@trim\toks@{#5}%
3155 \@temptokena\expandafter{\bbl@savestate}%
3156 \bbl@exp{% Reverse order - in ini last wins
3157 \def\\bbl@savestate{%
3158 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3159 \the\@temptokena}}}%
3160 {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3161 {\lowercase{\def\bbl@tempb{#6}}}%
3162 \bbl@trim@def\bbl@toreplace{#5}%
3163 \bbl@TG@@date
3164 \bbl@ifunset\bbl@date@\language @}%
3165 {\bbl@exp{% TODO. Move to a better place.
3166 \gdef\<\language date>{\protect\<\language date >}%
3167 \gdef\<\language date >####1####2####3{%
3168 \\bbl@usedategroupttrue
3169 \<bbl@ensure@\language >{%
3170 \\localedate{####1}{####2}{####3}}}%
3171 \\bbl@add\\bbl@savetoday{%
3172 \\SetString\\today{%
3173 \<\language date>%
3174 {\the\year}{\the\month}{\the\day}}}%
3175 {}}%
3176 \global\bbl@csarg\let{date@\language @}\bbl@toreplace
3177 \ifx\bbl@tempb\@empty\else
3178 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3179 \fi}%
3180 {}}}}

```

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).

```

3181 \let\bbl@calendar\@empty
3182 \newcommand\BabelDateSpace{\nobreakspace}
3183 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3184 \newcommand\BabelDated[1]{\number#1}
3185 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3186 \newcommand\BabelDateM[1]{\number#1}
3187 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3188 \newcommand\BabelDateMMM[1]{%
3189 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3190 \newcommand\BabelDatey[1]{\number#1}%
3191 \newcommand\BabelDateyy[1]{%
3192 \ifnum#1<10 0\number#1 %
3193 \else\ifnum#1<100 \number#1 %
3194 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3195 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %

```

```

3196 \else
3197 \bbl@error
3198 {Currently two-digit years are restricted to the\
3199 range 0-9999.}%
3200 {There is little you can do. Sorry.}%
3201 \fi\fi\fi\fi}
3202 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3203 \def\bbl@replace@finish@iii#1{%
3204 \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3205 \def\bbl@TG@date{%
3206 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3207 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3208 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3209 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3210 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3211 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3212 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3213 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3214 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3215 \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3216 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[###1|]}%
3217 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[###2|]}%
3218 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[###3|]}%
3219 \bbl@replace@finish@iii\bbl@toreplace}
3220 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3221 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3222 \let\bbl@release@transforms\@empty
3223 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3224 \bbl@transforms\babelprehyphenation}
3225 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3226 \bbl@transforms\babelposthyphenation}
3227 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3228 \begingroup % A hack. TODO. Don't require an specific order
3229 \catcode\%=12
3230 \catcode\&=14
3231 \gdef\bbl@transforms#1#2#3{&%
3232 \ifx\bbl@KVP@transforms\@nil\else
3233 \directlua{
3234 str = [==[#2]==]
3235 str = str:gsub('%.%d+%.%d+$', '')
3236 tex.print([[ \def\string\babeltempa{]] .. str .. [ ]]])
3237 }&%
3238 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3239 \ifin@
3240 \in@{.0$}{#2$}&%
3241 \ifin@
3242 \g@addto@macro\bbl@release@transforms{&%
3243 \relax\bbl@transforms@aux#1{\languagenam}{#3}}&%
3244 \else
3245 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3246 \fi
3247 \fi
3248 \fi}
3249 \endgroup

```

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


```

3250 \def\bbl@provide@lsys#1{%
3251   \bbl@ifunset{bbl@lname@#1}%
3252     {\bbl@load@info{#1}}%
3253   }%
3254   \bbl@csarg\let{lsys@#1}\@empty
3255   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3256   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3257   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3258   \bbl@ifunset{bbl@lname@#1}{}%
3259     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3260   \ifcase\bbl@engine\or\or
3261     \bbl@ifunset{bbl@prehc@#1}{}%
3262     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3263       }%
3264     {\ifx\bbl@xenoxyph\@undefined
3265       \let\bbl@xenoxyph\bbl@xenoxyph@d
3266       \ifx\AtBeginDocument\@notprerr
3267         \expandafter\@secondoftwo % to execute right now
3268         \fi
3269         \AtBeginDocument{%
3270           \bbl@patchfont{\bbl@xenoxyph}%
3271           \expandafter\selectlanguage\expandafter{\language}%
3272         \fi}%
3273   \fi
3274   \bbl@csarg\bbl@tglobal{lsys@#1}}
3275 \def\bbl@xenoxyph@d{%
3276   \bbl@ifset{bbl@prehc@language}%
3277     {\ifnum\hyphenchar\font=\defaultshyphenchar
3278       \iffontchar\font\bbl@c1{prehc}\relax
3279       \hyphenchar\font\bbl@c1{prehc}\relax
3280       \else\iffontchar\font"200B
3281       \hyphenchar\font"200B
3282       \else
3283         \bbl@warning
3284           {Neither 0 nor ZERO WIDTH SPACE are available\\%
3285            in the current font, and therefore the hyphen\\%
3286            will be printed. Try changing the fontspec's\\%
3287            'HyphenChar' to another value, but be aware\\%
3288            this setting is not safe (see the manual)}%
3289         \hyphenchar\font\defaultshyphenchar
3290       \fi\fi
3291     \fi}%
3292   {\hyphenchar\font\defaultshyphenchar}}
3293 % \fi}

```

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

```

3294 \def\bbl@load@info#1{%
3295   \def\BabelBeforeIni##1##2{%
3296     \begingroup
3297       \bbl@read@ini{##1}0%
3298       \endinput          % babel- .tex may contain onlypreamble's
3299       \endgroup}%       boxed, to avoid extra spaces:
3300   {\bbl@input@texini{#1}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T_EX. Non-digits characters are kept.

[illegible]

```

3332 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3333   \ifx\\#1%           % \\ before, in case #1 is multiletter
3334     \bbl@exp{%
3335       \def\\ \bbl@tempa####1{%
3336         \<ifcase>####1\space\the\toks@\<else>\\ \ctrerr\<fi>}}%
3337   \else
3338     \toks@\expandafter{\the\toks@\or #1}%
3339     \expandafter\bbl@buildifcase
3340   \fi}

```

```

3341 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1\@language}{#2}}
3342 \def\bbl@localecntr#1#2{\@localenumeral{#2}{#1}}
3343 \newcommand\localecounter[2]{%
3344   \expandafter\bbl@localecntr
3345   \expandafter{\number\csname c@#2\endcsname}{#1}}
3346 \def\bbl@alphnumeral#1#2{%
3347   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3348 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3349   \ifcase\car#8\nilor % Currently <10000, but prepared for bigger
3350     \bbl@alphnumeral@ii{#9}000000#1\or

```

```

3351 \bbl@alphanumeric@ii{#9}0000#1#2\or
3352 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3353 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3354 \bbl@alphnum@invalid{>9999}%
3355 \fi}
3356 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3357 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3358 {\bbl@cs{cntr@#1.4@\language}#5%
3359 \bbl@cs{cntr@#1.3@\language}#6%
3360 \bbl@cs{cntr@#1.2@\language}#7%
3361 \bbl@cs{cntr@#1.1@\language}#8%
3362 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3363 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3364 {\bbl@cs{cntr@#1.S.321@\language}}%
3365 \fi}%
3366 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
3367 \def\bbl@alphnum@invalid#1{%
3368 \bbl@error{Alphabetic numeral too large (#1)}%
3369 {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.

```

3370 \newcommand\localeinfo[1]{%
3371 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
3372 {\bbl@error{I've found no info for the current locale.\%
3373 The corresponding ini file has not been loaded\%
3374 Perhaps it doesn't exist}%
3375 {See the manual for details.}}%
3376 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}%
3377 % \@namedef{bbl@info@name.locale}{lcname}
3378 \@namedef{bbl@info@tag.ini}{lini}
3379 \@namedef{bbl@info@name.english}{elname}
3380 \@namedef{bbl@info@name.opentype}{lname}
3381 \@namedef{bbl@info@tag.bcp47}{tbc}
3382 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3383 \@namedef{bbl@info@tag.opentype}{lotf}
3384 \@namedef{bbl@info@script.name}{esname}
3385 \@namedef{bbl@info@script.name.opentype}{sname}
3386 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3387 \@namedef{bbl@info@script.tag.opentype}{sotf}
3388 \let\bbl@ensureinfo\@gobble
3389 \newcommand\BabelEnsureInfo{%
3390 \ifx\InputIfFileExists\undefined\else
3391 \def\bbl@ensureinfo##1{%
3392 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
3393 \fi
3394 \bbl@foreach\bbl@loaded{%
3395 \def\language{##1}%
3396 \bbl@ensureinfo{##1}}}

```

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

```

3397 \newcommand\getlocaleproperty{%
3398 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3399 \def\bbl@getproperty@s#1#2#3{%
3400 \let#1\relax
3401 \def\bbl@elt##1##2##3{%
3402 \bbl@ifsamestring{##1/##2}{#3}%

```

```

3403      {\providecommand#1{##3}%
3404      \def\bbl@elt####1####2####3{}}%
3405      {}}%
3406      \bbl@cs{inidata@#2}}%
3407 \def\bbl@getproperty@x#1#2#3{%
3408   \bbl@getproperty@s{#1}{#2}{#3}%
3409   \ifx#1\relax
3410     \bbl@error
3411     {Unknown key for locale '#2':\%
3412      #3\%
3413      \string#1 will be set to \relax}%
3414     {Perhaps you misspelled it.}%
3415   \fi}
3416 \let\bbl@ini@loaded\@empty
3417 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

9 Adjusting the Babel behavior

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

```

3418 \newcommand\babeladjust[1]{% TODO. Error handling.
3419   \bbl@forkv{#1}{%
3420     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3421     {\bbl@cs{ADJ@##1}{##2}}%
3422     {\bbl@cs{ADJ@##1@##2}}}
3423 %
3424 \def\bbl@adjust@lua#1#2{%
3425   \ifvmode
3426     \ifnum\currentgrouplevel=\z@
3427       \directlua{ Babel.#2 }%
3428       \expandafter\expandafter\expandafter\@gobble
3429     \fi
3430   \fi
3431   {\bbl@error % The error is gobbled if everything went ok.
3432    {Currently, #1 related features can be adjusted only\%
3433     in the main vertical list.%
3434     {Maybe things change in the future, but this is what it is.}}}
3435 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3436   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3437 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3438   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3439 \@namedef{\bbl@ADJ@bidi.text@on}{%
3440   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3441 \@namedef{\bbl@ADJ@bidi.text@off}{%
3442   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3443 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3444   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3445 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3446   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3447 %
3448 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3449   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3450 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3451   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3452 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
3453   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3454 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
3455   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3456 \@namedef{\bbl@ADJ@justify.arabic@on}{%

```

```

3457 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3458 \@namedef{bbl@ADJ@justify.arabic@off}{%
3459 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3460 %
3461 \def\bbl@adjust@layout#1{%
3462 \ifvmode
3463 #1%
3464 \expandafter\@gobble
3465 \fi
3466 {\bbl@error % The error is gobbled if everything went ok.
3467 {Currently, layout related features can be adjusted only\%
3468 in vertical mode.}%
3469 {Maybe things change in the future, but this is what it is.}}}
3470 \@namedef{bbl@ADJ@layout.tabular@on}{%
3471 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3472 \@namedef{bbl@ADJ@layout.tabular@off}{%
3473 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3474 \@namedef{bbl@ADJ@layout.lists@on}{%
3475 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3476 \@namedef{bbl@ADJ@layout.lists@off}{%
3477 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3478 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3479 \bbl@activateposthyphen}
3480 %
3481 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3482 \bbl@bcppallowedtrue}
3483 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3484 \bbl@bcppallowedfalse}
3485 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3486 \def\bbl@bcp@prefix{#1}}
3487 \def\bbl@bcp@prefix{bcp47-}
3488 \@namedef{bbl@ADJ@autoload.options}#1{%
3489 \def\bbl@autoload@options{#1}}
3490 \let\bbl@autoload@bcptoptions\@empty
3491 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3492 \def\bbl@autoload@bcptoptions{#1}}
3493 \newif\ifbbl@bcptname
3494 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3495 \bbl@bcptnametrue}
3496 \BabelEnsureInfo}
3497 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3498 \bbl@bcptnamefalse}
3499 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3500 \directlua{ Babel.ignore_pre_char = function(node)
3501 return (node.lang == \the\csname l@nohyphenation\endcsname)
3502 end }}
3503 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3504 \directlua{ Babel.ignore_pre_char = function(node)
3505 return false
3506 end }}
3507 \@namedef{bbl@ADJ@select.write@shift}{%
3508 \let\bbl@restorelastskip\relax
3509 \def\bbl@savelastskip{%
3510 \let\bbl@restorelastskip\relax
3511 \ifvmode
3512 \ifdim\lastskip=\z@
3513 \let\bbl@restorelastskip\nobreak
3514 \else
3515 \bbl@exp{%

```

```

3516         \def\\bbl@restorelastskip{%
3517             \skip@=\the\lastskip
3518             \\nobreak \vskip-\skip@ \vskip\skip@}%
3519         \fi
3520     \fi}}
3521 \@namedef{bbl@ADJ@select.write@keep}{%
3522     \let\bbl@restorelastskip\relax
3523     \let\bbl@savelastskip\relax}
3524 \@namedef{bbl@ADJ@select.write@omit}{%
3525     \let\bbl@restorelastskip\relax
3526     \def\bbl@savelastskip##1\bbl@restorelastskip{}}

```

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

```

3527 \ifx\directlua\@undefined\else
3528     \ifx\bbl@luapatterns\@undefined
3529         \input luababel.def
3530     \fi
3531 \fi

```

Continue with \LaTeX .

```

3532 </package | core>
3533 <*package>

```

9.1 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.

```

3534 <<*More package options>> ≡
3535 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3536 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3537 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3538 <</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.

```

3539 \bbl@trace{Cross referencing macros}
3540 \ifx\bbl@opt@safe\@empty\else
3541     \def\@newl@bel#1#2#3{%
3542         {\@safe@activestrue
3543         \bbl@ifunset{#1@#2}%
3544             \relax
3545             {\gdef\@multiplelabels{%
3546                 \@latex@warning@no@line{There were multiply-defined labels}}%
3547                 \@latex@warning@no@line{Label `#2' multiply defined}}%
3548             \global\@namedef{#1@#2}{#3}}}

```

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

```

3549 \CheckCommand*\@testdef[3]{%

```

```

3550 \def\reserved@a{#3}%
3551 \expandafter\ifx\cename#1@#2\endcsname\reserved@a
3552 \else
3553 \@tempswatrue
3554 \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.

```

3555 \def\@testdef#1#2#3{% TODO. With @samestring?
3556 \@safe@activetrue
3557 \expandafter\let\expandafter\bbl@tempa\cename #1@#2\endcsname
3558 \def\bbl@tempb{#3}%
3559 \@safe@activetrue
3560 \ifx\bbl@tempa\relax
3561 \else
3562 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3563 \fi
3564 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3565 \ifx\bbl@tempa\bbl@tempb
3566 \else
3567 \@tempswatrue
3568 \fi}
3569 \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.

```

3570 \bbl@xin@{R}\bbl@opt@safe
3571 \ifin@
3572 \bbl@redefineroobust\ref#1{%
3573 \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3574 \bbl@redefineroobust\pageref#1{%
3575 \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3576 \else
3577 \let\org@ref\ref
3578 \let\org@pageref\pageref
3579 \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.

```

3580 \bbl@xin@{B}\bbl@opt@safe
3581 \ifin@
3582 \bbl@redefine\@citex[#1]#2{%
3583 \@safe@activetrue\edef\@tempa{#2}\@safe@activetrue}
3584 \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.

```

3585 \AtBeginDocument{%
3586 \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.)

```
3587 \def\@citex[#1][#2]#3{%
3588   \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3589   \org@@citex[#1][#2]{\@tempa}}%
3590 }{}}
```

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

```
3591 \AtBeginDocument{%
3592   \@ifpackageloaded{cite}{%
3593     \def\@citex[#1]#2{%
3594       \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
3595     }{}}
```

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

```
3596 \bbl@redefine\nocite#1{%
3597   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3598 \bbl@redefine\bibcite{%
3599   \bbl@cite@choice
3600   \bibcite}
```

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

```
3601 \def\bbl@bibcite#1#2{%
3602   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3603 \def\bbl@cite@choice{%
3604   \global\let\bibcite\bbl@bibcite
3605   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3606   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3607   \global\let\bbl@cite@choice\relax}
```

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

```
3608 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3609 \bbl@redefine\@bibitem#1{%
3610   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3611 \else
3612   \let\org@nocite\nocite
3613   \let\org@@citex\@citex
3614   \let\org@bibcite\bibcite
3615   \let\org@@bibitem\@bibitem
3616 \fi
```


9.2 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used.

We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

```

3617 \bbl@trace{Marks}
3618 \IfBabelLayout{sectioning}
3619   {\ifx\bbl@opt@headfoot\@nnil
3620     \g@addto@macro\@resetactivechars{%
3621       \set@typeset@protect
3622       \expandafter\select@language@\expandafter{\bbl@main@language}%
3623       \let\protect\noexpand
3624       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3625         \edef\thepage{%
3626           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3627       \fi}%
3628   \fi}
3629 {\ifbbl@single\else
3630   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3631   \markright#1{%
3632     \bbl@ifblank{#1}%
3633     {\org@markright{}}%
3634     {\toks@{#1}%
3635       \bbl@exp{%
3636         \\org@markright{\protect\\foreignlanguage{\language}%
3637           {\protect\\bbl@restore@actives\the\toks@}}}%

```

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

`\@mkboth`

```

3638   \ifx\@mkboth\markboth
3639     \def\bbl@tempc{\let\@mkboth\markboth}
3640   \else
3641     \def\bbl@tempc{}
3642   \fi
3643   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3644   \markboth#1#2{%
3645     \protected@edef\bbl@tempb##1{%
3646       \protect\foreignlanguage
3647       {\language}{\protect\bbl@restore@actives##1}%
3648     \bbl@ifblank{#1}%
3649     {\toks@{}}%
3650     {\toks@\expandafter{\bbl@tempb{#1}}}%
3651     \bbl@ifblank{#2}%
3652     {\@temptokena{}}%
3653     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3654     \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
3655     \bbl@tempc
3656   \fi} % end ifbbl@single, end \IfBabelLayout

```

9.3 Preventing clashes with other packages

9.3.1 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.

```
3657 \bbl@trace{Preventing clashes with other packages}
3658 \bbl@xin@{R}\bbl@opt@safe
3659 \ifin@
3660 \AtBeginDocument{%
3661   \@ifpackageloaded{ifthen}{%
3662     \bbl@redefine@long\ifthenelse#1#2#3{%
3663       \let\bbl@temp@pref\pageref
3664       \let\pageref\org@pageref
3665       \let\bbl@temp@ref\ref
3666       \let\ref\org@ref
3667       \@safe@activestrue
3668       \org@ifthenelse{#1}%
3669         {\let\pageref\bbl@temp@pref
3670          \let\ref\bbl@temp@ref
3671          \@safe@activesfalse
3672          #2}%
3673         {\let\pageref\bbl@temp@pref
3674          \let\ref\bbl@temp@ref
3675          \@safe@activesfalse
3676          #3}%
3677     }%
3678   }{}%
3679 }
```

9.3.2 varioref

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

```
3680 \AtBeginDocument{%
3681   \@ifpackageloaded{varioref}{%
3682     \bbl@redefine\@@vpageref#1[#2]#3{%
3683       \@safe@activestrue
3684       \org@@@vpageref{#1}[#2]{#3}%
3685       \@safe@activesfalse}%
3686     \bbl@redefine\vrefpagemum#1#2{%
3687       \@safe@activestrue
3688       \org@vrefpagemum{#1}{#2}%
3689       \@safe@activesfalse}%
3690   }
```

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.

```

3690 \expandafter\def\csname Ref \endcsname#1{%
3691 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3692 }{}%
3693 }
3694 \fi

```

9.3.3 `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.

```

3695 \AtEndOfPackage{%
3696 \AtBeginDocument{%
3697 \ifpackageloaded{hhline}%
3698 {\expandafter\ifx\csname normal@char\string\endcsname\relax
3699 \else
3700 \makeatletter
3701 \def\@currname{hhline}\input{hhline.sty}\makeatother
3702 \fi}%
3703 {}}}

```

`\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.

```

3704 \def\substitutefontfamily#1#2#3{%
3705 \lowercase{\immediate\openout15=#1#2.fd\relax}%
3706 \immediate\write15{%
3707 \string\ProvidesFile{#1#2.fd}%
3708 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3709 \space generated font description file]^^J
3710 \string\DeclareFontFamily{#1}{#2}{^^J
3711 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3712 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3713 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3714 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3715 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3716 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3717 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3718 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3719 }%
3720 \closeout15
3721 }
3722 \@onlypreamble\substitutefontfamily

```

9.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings are currently stored in `\@fontenc@load@list`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensureascii

```
3723 \bbl@trace{Encoding and fonts}
3724 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3725 \newcommand\BabelNonText{TS1,T3,TS3}
3726 \let\org@TeX\TeX
3727 \let\org@LaTeX\LaTeX
3728 \let\ensureascii\@firstofone
3729 \AtBeginDocument{%
3730   \def\@elt#1{,#1,}%
3731   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3732   \let\@elt\relax
3733   \let\bbl@tempb\@empty
3734   \def\bbl@tempc{OT1}%
3735   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3736     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3737   \bbl@foreach\bbl@tempa{%
3738     \bbl@xin@{#1}{\BabelNonASCII}%
3739     \ifin@
3740       \def\bbl@tempb{#1}% Store last non-ascii
3741     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3742     \ifin@
3743       \def\bbl@tempc{#1}% Store last ascii
3744     \fi
3745   \fi}%
3746   \ifx\bbl@tempb\@empty\else
3747     \bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
3748   \ifin@
3749     \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3750   \fi
3751   \edef\ensureascii#1{%
3752     {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3753   \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3754   \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3755   \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.

```
3756 \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.

```
3757 \AtBeginDocument{%
3758   \@ifpackageloaded{fontspec}%
3759   {\xdef\latinencoding{%
3760     \ifx\UTFencname\undefined
3761       EU\ifcase\bbl@engine\or2\or1\fi
3762     \else
3763       \UTFencname
3764     \fi}}%
3765   {\gdef\latinencoding{OT1}%
3766     \ifx\cf@encoding\bbl@t@one
3767       \xdef\latinencoding{\bbl@t@one}%
3768     \else
```

```

3769      \def\@elt#1{, #1,}%
3770      \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3771      \let\@elt\relax
3772      \bbl@xin@{, T1, }\bbl@tempa
3773      \ifin@
3774          \xdef\latinencoding{\bbl@t@one}%
3775      \fi
3776      \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.

```

3777 \DeclareRobustCommand{\latintext}{%
3778   \fontencoding{\latinencoding}\selectfont
3779   \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.

```

3780 \ifx\@undefined\DeclareTextFontCommand
3781   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3782 \else
3783   \DeclareTextFontCommand{\textlatin}{\latintext}
3784 \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).

```

3785 \bbl@ifformatlater{2021-06-01}%
3786   {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
3787   {\def\bbl@patchfont#1{%
3788     \expandafter\bbl@add\csname selectfont \endcsname{#1}%
3789     \expandafter\bbl@tglobal\csname selectfont \endcsname}}

```

9.5 Basic bidi support

Work in progress. This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This `babel` module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at `ARABI` (by Youssef Jabri), which is compatible with `babel`.

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like `rlbabel` did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua \TeX -ja` shows, vertical typesetting is possible, too.

```

3790 \bbl@trace{Loading basic (internal) bidi support}
3791 \ifodd\bbl@engine
3792 \else % TODO. Move to txtbabel
3793   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200

```

```

3794 \bbl@error
3795 {The bidi method 'basic' is available only in\%
3796 luatex. I'll continue with 'bidi=default', so\%
3797 expect wrong results}%
3798 {See the manual for further details.}%
3799 \let\bbl@beforeforeign\leavevmode
3800 \AtEndOfPackage{%
3801 \EnableBabelHook{babel-bidi}%
3802 \bbl@xebidipar}
3803 \fi\fi
3804 \def\bbl@loadxebidi#1{%
3805 \ifx\RTLfootnotetext\undefined
3806 \AtEndOfPackage{%
3807 \EnableBabelHook{babel-bidi}%
3808 \ifx\fontspec\undefined
3809 \bbl@loadfontspec % bidi needs fontspec
3810 \fi
3811 \usepackage#1{bidi}}%
3812 \fi}
3813 \ifnum\bbl@bidimode>200
3814 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3815 \bbl@tentative{bidi=bidi}
3816 \bbl@loadxebidi{}
3817 \or
3818 \bbl@loadxebidi{[rldocument]}
3819 \or
3820 \bbl@loadxebidi{}
3821 \fi
3822 \fi
3823 \fi
3824 % TODO? Separate:
3825 \ifnum\bbl@bidimode=\@ne
3826 \let\bbl@beforeforeign\leavevmode
3827 \ifodd\bbl@engine
3828 \newattribute\bbl@attr@dir
3829 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3830 \bbl@exp{\output{\bodydir\pagedir\the\output}}
3831 \fi
3832 \AtEndOfPackage{%
3833 \EnableBabelHook{babel-bidi}%
3834 \ifodd\bbl@engine\else
3835 \bbl@xebidipar
3836 \fi}
3837 \fi

```

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

```

3838 \bbl@trace{Macros to switch the text direction}
3839 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3840 \def\bbl@rscripts{% TODO. Base on codes ??
3841 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3842 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
3843 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
3844 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3845 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3846 Old South Arabian,}%
3847 \def\bbl@provide@dirs#1{%
3848 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3849 \ifin@

```

```

3850 \global\bbbl@csarg\chardef{wdir@#1}\@ne
3851 \bbbl@xin@\csname bbl@sname@#1\endcsname\{\bbbl@alscripts}%
3852 \ifin@
3853 \global\bbbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3854 \fi
3855 \else
3856 \global\bbbl@csarg\chardef{wdir@#1}\z@
3857 \fi
3858 \ifodd\bbbl@engine
3859 \bbbl@csarg\ifcase{wdir@#1}%
3860 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
3861 \or
3862 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
3863 \or
3864 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
3865 \fi
3866 \fi}
3867 \def\bbbl@switchdir{%
3868 \bbbl@ifunset{bbbl@sys@\languagename}\{\bbbl@provide@sys{\languagename}}{}%
3869 \bbbl@ifunset{bbbl@wdir@\languagename}\{\bbbl@provide@dirs{\languagename}}{}%
3870 \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}%
3871 \def\bbbl@setdirs#1{% TODO - math
3872 \ifcase\bbbl@select@type % TODO - strictly, not the right test
3873 \bbbl@bodydir{#1}%
3874 \bbbl@pardir{#1}%
3875 \fi
3876 \bbbl@texmdir{#1}}
3877 % TODO. Only if \bbbl@bidimode > 0?:
3878 \AddBabelHook{babel-bidi}{afterextras}\{\bbbl@switchdir}
3879 \DisableBabelHook{babel-bidi}

```

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

```

3880 \ifodd\bbbl@engine % luatex=1
3881 \else % pdftex=0, xetex=2
3882 \newcount\bbbl@dirlevel
3883 \chardef\bbbl@thetexmdir\z@
3884 \chardef\bbbl@thepardir\z@
3885 \def\bbbl@texmdir#1{%
3886 \ifcase#1\relax
3887 \chardef\bbbl@thetexmdir\z@
3888 \bbbl@texmdir@i\beginL\endL
3889 \else
3890 \chardef\bbbl@thetexmdir\@ne
3891 \bbbl@texmdir@i\beginR\endR
3892 \fi}
3893 \def\bbbl@texmdir@i#1#2{%
3894 \ifhmode
3895 \ifnum\currentgrouplevel>\z@
3896 \ifnum\currentgrouplevel=\bbbl@dirlevel
3897 \bbbl@error{Multiple bidi settings inside a group}%
3898 {I'll insert a new group, but expect wrong results.}%
3899 \bgroup\aftergroup#2\aftergroup\egroup
3900 \else
3901 \ifcase\currentgroup\or % 0 bottom
3902 \aftergroup#2% 1 simple {}
3903 \or
3904 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3905 \or
3906 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox

```

```

3907      \or\or\or % vbox vtop align
3908      \or
3909      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3910      \or\or\or\or\or\or % output math disc insert vcent mathchoice
3911      \or
3912      \aftergroup#2% 14 \begingroup
3913      \else
3914      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3915      \fi
3916      \fi
3917      \bbl@dirlevel\currentgrouplevel
3918      \fi
3919      #1%
3920      \fi}
3921      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
3922      \let\bbl@bodydir\@gobble
3923      \let\bbl@pagedir\@gobble
3924      \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).

```

3925      \def\bbl@xebidipar{%
3926      \let\bbl@xebidipar\relax
3927      \TeXeTstate\@ne
3928      \def\bbl@xeverypar{%
3929      \ifcase\bbl@thepardir
3930      \ifcase\bbl@thetextdir\else\beginR\fi
3931      \else
3932      {\setbox\z@\lastbox\beginR\box\z@}%
3933      \fi}%
3934      \let\bbl@severypar\everypar
3935      \newtoks\everypar
3936      \everypar=\bbl@severypar
3937      \bbl@severypar{\bbl@xeverypar\the\everypar}}
3938      \ifnum\bbl@bidimode>200
3939      \let\bbl@textdir\i\@gobbletwo
3940      \let\bbl@xebidipar\@empty
3941      \AddBabelHook{bidi}{foreign}{%
3942      \def\bbl@tempa{\def\BabelText###1}%
3943      \ifcase\bbl@thetextdir
3944      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
3945      \else
3946      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
3947      \fi}
3948      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
3949      \fi
3950      \fi

```

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

```

3951      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
3952      \AtBeginDocument{%
3953      \ifx\pdfstringdefDisableCommands\@undefined\else
3954      \ifx\pdfstringdefDisableCommands\relax\else
3955      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
3956      \fi
3957      \fi}

```


9.6 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`.

```
3958 \bbl@trace{Local Language Configuration}
3959 \ifx\loadlocalcfg\undefined
3960   \@ifpackagewith{babel}{noconfigs}%
3961   {\let\loadlocalcfg@gobble}%
3962   {\def\loadlocalcfg#1{%
3963     \InputIfFileExists{#1.cfg}%
3964     {\typeout{*****^J%
3965               * Local config file #1.cfg used^^J%
3966               *}}}%
3967   \@empty}}
3968 \fi
```

9.7 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).

```
3969 \bbl@trace{Language options}
3970 \let\bbl@afterlang\relax
3971 \let\BabelModifiers\relax
3972 \let\bbl@loaded\@empty
3973 \def\bbl@load@language#1{%
3974   \InputIfFileExists{#1.ldf}%
3975   {\edef\bbl@loaded{\CurrentOption
3976     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
3977     \expandafter\let\expandafter\bbl@afterlang
3978     \csname\CurrentOption.ldf-h@k\endcsname
3979     \expandafter\let\expandafter\BabelModifiers
3980     \csname bbl@mod@\CurrentOption\endcsname}%
3981   {\bbl@error{%
3982     Unknown option '\CurrentOption'. Either you misspelled it\\%
3983     or the language definition file \CurrentOption.ldf was not found}}%
3984     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
3985     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
3986     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.

```
3987 \def\bbl@try@load@lang#1#2#3{%
3988   \IfFileExists{\CurrentOption.ldf}%
3989   {\bbl@load@language{\CurrentOption}}%
3990   {#1\bbl@load@language{#2}#3}}
3991 %
3992 \DeclareOption{hebrew}{%
3993   \input{rlbabel.def}%
3994   \bbl@load@language{hebrew}}
3995 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
3996 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
3997 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
3998 \DeclareOption{polutonikogreek}{%
```

```

3999 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}
4000 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4001 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4002 \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.

```

4003 \ifx\bbl@opt@config\@nnil
4004 \@ifpackagewith{babel}{noconfigs}{}%
4005   {\InputIfFileExists{bblopts.cfg}%
4006     {\typeout{*****^J%
4007               * Local config file bblopts.cfg used^^J%
4008               *}}%
4009   }%
4010 \else
4011 \InputIfFileExists{\bbl@opt@config.cfg}%
4012   {\typeout{*****^J%
4013             * Local config file \bbl@opt@config.cfg used^^J%
4014             *}}%
4015   {\bbl@error{%
4016     Local config file '\bbl@opt@config.cfg' not found}%
4017     Perhaps you misspelled it.}%
4018 \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages (note this list also contains the language given with `main`). If not declared above, the names of the option and the file are the same.

```

4019 \let\bbl@tempc\relax
4020 \bbl@foreach\bbl@language@opts{%
4021   \ifcase\bbl@iniflag % Default
4022     \bbl@ifunset{ds@#1}%
4023     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4024     {}%
4025   \or % provide=*
4026     \@gobble % case 2 same as 1
4027   \or % provide+=*
4028     \bbl@ifunset{ds@#1}%
4029     {\IfFileExists{#1.ldf}{}%
4030      {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}%
4031      {}%
4032     \bbl@ifunset{ds@#1}%
4033     {\def\bbl@tempc{#1}%
4034      \DeclareOption{#1}{%
4035        \ifnum\bbl@iniflag>\@ne
4036          \bbl@ldfinit
4037          \babelprovide[import]{#1}%
4038          \bbl@afterldf}%
4039        \else
4040          \bbl@load@language{#1}%
4041        \fi}%
4042     }%
4043   \or % provide*=*
4044     \def\bbl@tempc{#1}%
4045     \bbl@ifunset{ds@#1}%
4046     {\DeclareOption{#1}{%
4047       \bbl@ldfinit

```

```

4048      \babelprovide[import]{#1}%
4049      \bbl@afterldf{}}}%
4050    {}%
4051  \fi}

```

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

```

4052 \let\bbl@tempb\@nnil
4053 \let\bbl@clsoptlst\@classoptionslist
4054 \bbl@foreach\@classoptionslist{%
4055   \bbl@ifunset{ds@#1}%
4056   {\IfFileExists{#1.ldf}%
4057    {\def\bbl@tempb{#1}%
4058     \DeclareOption{#1}{%
4059       \ifnum\bbl@iniflag>\@ne
4060         \bbl@ldfinit
4061         \babelprovide[import]{#1}%
4062         \bbl@afterldf{}}%
4063       \else
4064         \bbl@load@language{#1}%
4065       \fi}}%
4066   {\IfFileExists{babel-#1.tex}%
4067    {\def\bbl@tempb{#1}%
4068     \ifnum\bbl@iniflag>\z@
4069     \DeclareOption{#1}{%
4070       \ifnum\bbl@iniflag>\@ne
4071         \bbl@ldfinit
4072         \babelprovide[import]{#1}%
4073         \bbl@afterldf{}}%
4074       \fi}%
4075     \fi}%
4076   {}}}%
4077 {}

```

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

```

4078 \ifnum\bbl@iniflag=\z@\else
4079   \ifx\bbl@opt@main\@nnil
4080     \ifx\bbl@tempc\relax
4081       \let\bbl@opt@main\bbl@tempb
4082     \else
4083       \let\bbl@opt@main\bbl@tempc
4084     \fi
4085   \fi
4086 \fi
4087 \ifx\bbl@opt@main\@nnil\else
4088   \expandafter
4089   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
4090   \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
4091 \fi

```

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

The options have to be processed in the order in which the user specified them (except, of course, global options, which \TeX processes before):

```

4092 \def\AfterBabelLanguage#1{%
4093   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}%
4094   \DeclareOption*{}
4095   \ProcessOptions*

```

This finished the second pass. Now the third one begins, which loads the main language set with the key main. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. Then execute directly the option (because it could be used only in main). After loading all languages, we deactivate \AfterBabelLanguage.

```

4096 \bbl@trace{Option 'main'}
4097 \ifx\bbl@opt@main\@nnil
4098   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4099   \let\bbl@tempc\@empty
4100   \bbl@for\bbl@tempb\bbl@tempa{%
4101     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4102     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4103   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4104   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4105   \ifx\bbl@tempb\bbl@tempc\else
4106     \bbl@warning{%
4107       Last declared language option is '\bbl@tempc',\%
4108       but the last processed one was '\bbl@tempb'.\%
4109       The main language can't be set as both a global\%
4110       and a package option. Use 'main=\bbl@tempc' as\%
4111       option. Reported}%
4112   \fi
4113 \else
4114   \ifodd\bbl@iniflag % case 1,3
4115     \bbl@ldfinit
4116     \let\CurrentOption\bbl@opt@main
4117     \ifx\bbl@opt@provide\@nnil
4118       \bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}%
4119     \else
4120       \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
4121         \bbl@xin@{,provide,}{, #1,}%
4122         \ifin@
4123           \def\bbl@opt@provide{#2}%
4124           \bbl@replace\bbl@opt@provide{;}{,}%
4125         \fi}%
4126       \bbl@exp{%
4127         \bbl@babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4128       \fi
4129       \bbl@afterldf}%
4130   \else % case 0,2
4131     \chardef\bbl@iniflag\z@ % Force ldf
4132     \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
4133     \ExecuteOptions{\bbl@opt@main}
4134     \DeclareOption*{%
4135       \ProcessOptions*
4136     \fi
4137   \fi
4138 \def\AfterBabelLanguage{%
4139   \bbl@error
4140   {Too late for \string\AfterBabelLanguage}%
4141   {Languages have been loaded, so I can do nothing}}

4142 \ifx\bbl@main@language\@undefined
4143   \bbl@info{%
4144     You haven't specified a language. I'll use 'nil'\%
4145     as the main language. Reported}
4146   \bbl@load@language{nil}

```

```

4147 \fi
4148 \</package>

```

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

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

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

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

A proxy file for switch.def

```

4149 <*kernel>
4150 \let\bbl@onlyswitch\@empty
4151 \input babel.def
4152 \let\bbl@onlyswitch\@undefined
4153 </kernel>
4154 <*patterns>

```

11 Loading hyphenation patterns

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

```

4155 <<Make sure ProvidesFile is defined>>
4156 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
4157 \xdef\bbl@format{\jobname}
4158 \def\bbl@version{\<version>}
4159 \def\bbl@date{\<date>}
4160 \ifx\AtBeginDocument\@undefined
4161   \def\@empty{}
4162 \fi
4163 <<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.

```

4164 \def\process@line#1#2 #3 #4 {%
4165   \ifx=#1%
4166     \process@synonym{#2}%
4167   \else
4168     \process@language{#1#2}{#3}{#4}%
4169   \fi
4170   \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.

```

4171 \toks@{}
4172 \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.

```

4173 \def\process@synonym#1{%
4174   \ifnum\last@language=\m@ne
4175     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4176   \else
4177     \expandafter\chardef\csname l@#1\endcsname\last@language
4178     \wlog{\string\l@#1=\string\language\the\last@language}%
4179     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4180       \csname\language\hyphenmins\endcsname
4181     \let\bbl@elt\relax
4182     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
4183   \fi}

```

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

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

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

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

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

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

`\bbl@elt{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

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

```

4184 \def\process@language#1#2#3{%
4185   \expandafter\addlanguage\csname l@#1\endcsname
4186   \expandafter\language\csname l@#1\endcsname
4187   \edef\language#1}%
4188   \bbl@hook@everylanguage{#1}%
4189   % > luatex
4190   \bbl@get@enc#1::\@@@
4191   \begingroup
4192     \lefthyphenmin\m@ne
4193     \bbl@hook@loadpatterns{#2}%
4194     % > luatex
4195     \ifnum\lefthyphenmin=\m@ne
4196     \else
4197       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4198         \the\lefthyphenmin\the\righthyphenmin}%
4199     \fi
4200   \endgroup

```

```

4201 \def\bbl@tempa{#3}%
4202 \ifx\bbl@tempa\@empty\else
4203   \bbl@hook@loadexceptions{#3}%
4204   % > luatex
4205 \fi
4206 \let\bbl@elt\relax
4207 \edef\bbl@languages{%
4208   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4209 \ifnum\the\language=\z@
4210   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4211     \set@hyphenmins\tw@\thr@@\relax
4212   \else
4213     \expandafter\expandafter\expandafter\set@hyphenmins
4214     \csname #1hyphenmins\endcsname
4215   \fi
4216   \the\toks@
4217   \toks@{}%
4218 \fi}

```

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

```

4219 \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.

```

4220 \def\bbl@hook@everylanguage#1{}
4221 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4222 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4223 \def\bbl@hook@loadkernel#1{%
4224   \def\addlanguage{\csname newlanguage\endcsname}%
4225   \def\adddialect##1##2{%
4226     \global\chardef##1##2\relax
4227     \wlog{\string##1 = a dialect from \string\language##2}}%
4228   \def\iflanguage##1{%
4229     \expandafter\ifx\csname l@##1\endcsname\relax
4230       \@nolanerr{##1}%
4231     \else
4232       \ifnum\csname l@##1\endcsname=\language
4233         \expandafter\expandafter\expandafter\@firstoftwo
4234       \else
4235         \expandafter\expandafter\expandafter\@secondoftwo
4236       \fi
4237     \fi}%
4238   \def\providehyphenmins##1##2{%
4239     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4240       \@namedef{##1hyphenmins}{##2}%
4241     \fi}%
4242   \def\set@hyphenmins##1##2{%
4243     \lefthyphenmin##1\relax
4244     \righthyphenmin##2\relax}%
4245   \def\selectlanguage{%
4246     \errhelp{Selecting a language requires a package supporting it}%
4247     \errmessage{Not loaded}}%
4248   \let\foreignlanguage\selectlanguage
4249   \let\otherlanguage\selectlanguage
4250   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4251   \def\bbl@usehooks##1##2{% TODO. Temporary!!

```

```

4252 \def\setlocale{%
4253   \errhelp{Find an armchair, sit down and wait}%
4254   \errmessage{Not yet available}}%
4255 \let\uselocale\setlocale
4256 \let\locale\setlocale
4257 \let\selectlocale\setlocale
4258 \let\localename\setlocale
4259 \let\textlocale\setlocale
4260 \let\textlanguage\setlocale
4261 \let\language\setlocale
4262 \begingroup
4263 \def\AddBabelHook#1#2{%
4264   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4265     \def\next{\toks1}%
4266     \else
4267       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4268     \fi
4269     \next}
4270 \ifx\directlua\undefined
4271   \ifx\XeTeXinputencoding\undefined\else
4272     \input xebabel.def
4273   \fi
4274 \else
4275   \input luababel.def
4276 \fi
4277 \openin1 = babel-\bbl@format.cfg
4278 \ifeof1
4279 \else
4280   \input babel-\bbl@format.cfg\relax
4281 \fi
4282 \closein1
4283 \endgroup
4284 \bbl@hook@loadkernel{switch.def}

```

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

```

4285 \openin1 = language.dat

```

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

```

4286 \def\language{english}%
4287 \ifeof1
4288 \message{I couldn't find the file language.dat,\space
4289         I will try the file hyphen.tex}
4290 \input hyphen.tex\relax
4291 \chardef\l@english\z@
4292 \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`.

```

4293 \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.

```

4294 \loop
4295   \endlinechar\m@ne
4296   \read1 to \bbl@line
4297   \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.

```
4298 \if T\ifeof1F\fi T\relax
4299 \ifx\bbl@line\@empty\else
4300 \edef\bbl@line{\bbl@line\space\space\space}%
4301 \expandafter\process@line\bbl@line\relax
4302 \fi
4303 \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.

```
4304 \begingroup
4305 \def\bbl@elt#1#2#3#4{%
4306 \global\language=#2\relax
4307 \gdef\language#1}%
4308 \def\bbl@elt##1##2##3##4{}}%
4309 \bbl@languages
4310 \endgroup
4311 \fi
4312 \closein1
```

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

```
4313 \if/\the\toks@/\else
4314 \errhelp{language.dat loads no language, only synonyms}
4315 \errmessage{Orphan language synonym}
4316 \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.

```
4317 \let\bbl@line\@undefined
4318 \let\process@line\@undefined
4319 \let\process@synonym\@undefined
4320 \let\process@language\@undefined
4321 \let\bbl@get@enc\@undefined
4322 \let\bbl@hyph@enc\@undefined
4323 \let\bbl@tempa\@undefined
4324 \let\bbl@hook@loadkernel\@undefined
4325 \let\bbl@hook@everylanguage\@undefined
4326 \let\bbl@hook@loadpatterns\@undefined
4327 \let\bbl@hook@loadexceptions\@undefined
4328 \patterns)
```

Here the code for `iniTeX` ends.

12 Font handling with fontspec

Add the bidi handler just before `luaotfload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
4329 <<{*More package options}>> ≡
4330 \chardef\bbl@bidimode\z@
4331 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4332 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4333 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4334 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4335 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4336 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4337 <</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.

```

4338 <<{*Font selection}>> ≡
4339 \bbl@trace{Font handling with fontspec}
4340 \ifx\ExplSyntaxOn\undefined\else
4341   \ExplSyntaxOn
4342   \catcode`\ =10
4343   \def\bbl@loadfontspec{%
4344     \usepackage{fontspec}% TODO. Apply patch always
4345     \expandafter
4346     \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4347       Font '\l_fontspec_fontname_tl' is using the\\%
4348       default features for language '##1'.\\%
4349       That's usually fine, because many languages\\%
4350       require no specific features, but if the output is\\%
4351       not as expected, consider selecting another font.}
4352     \expandafter
4353     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%
4354       Font '\l_fontspec_fontname_tl' is using the\\%
4355       default features for script '##2'.\\%
4356       That's not always wrong, but if the output is\\%
4357       not as expected, consider selecting another font.}}
4358   \ExplSyntaxOff
4359 \fi
4360 \@onlypreamble\babelfont
4361 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4362   \bbl@foreach{#1}{%
4363     \expandafter\ifx\csname date##1\endcsname\relax
4364       \IfFileExists{babel-##1.tex}%
4365         {\babelprovide{##1}}%
4366         {}%
4367     \fi}%
4368   \edef\bbl@tempa{#1}%
4369   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4370   \ifx\fontspec\undefined
4371     \bbl@loadfontspec
4372   \fi
4373   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4374   \bbl@bblfont}
4375 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4376   \bbl@ifunset{\bbl@tempb family}%
4377     {\bbl@providfam{\bbl@tempb}}%
4378     {}%
4379   % For the default font, just in case:
4380   \bbl@ifunset{\bbl@lsys@languagename}{\bbl@provide@lsys{languagename}}{}%
4381   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4382     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save \bbl@rmdflt@
4383     \bbl@exp{%
4384       \let<\bbl@tempb dflt@languagename>\<\bbl@tempb dflt@>%
4385       \\\bbl@font@set<\bbl@tempb dflt@languagename>%
4386         \<\bbl@tempb default>\<\bbl@tempb family>}}%
4387     {\bbl@foreach\bbl@tempa{% ie \bbl@rmdflt@lang / *scrt
4388       \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:

```

4389 \def\bbl@providefam#1{%
4390   \bbl@exp{%
4391     \\\newcommand\<#1default>{}% Just define it
4392     \\\bbl@add@list\\bbl@font@fams{#1}%
4393     \\\DeclareRobustCommand\<#1family>{%
4394       \\\not@math@alphabet\<#1family>\relax
4395       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4396       \\\fontfamily\<#1default>%
4397       \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4398       \\\selectfont}%
4399       \\\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.

```

4400 \def\bbl@nostdfont#1{%
4401   \bbl@ifunset{bbl@WFF@f@family}%
4402   {\bbl@csarg\gdef{WFF@f@family}}{% Flag, to avoid dupl warns
4403     \bbl@infowarn{The current font is not a babel standard family:\%
4404       #1%
4405       \fontname\font\\%
4406       There is nothing intrinsically wrong with this warning, and\\%
4407       you can ignore it altogether if you do not need these\\%
4408       families. But if they are used in the document, you should be\\%
4409       aware 'babel' will no set Script and Language for them, so\\%
4410       you may consider defining a new family with \string\babelfont.\\%
4411       See the manual for further details about \string\babelfont.\\%
4412       Reported}}
4413   {}}%
4414 \gdef\bbl@switchfont{%
4415   \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}{%
4416     \bbl@exp{% eg Arabic -> arabic
4417       \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4418     \bbl@foreach\bbl@font@fams{%
4419       \bbl@ifunset{bbl@##1dflt@language}% (1) language?
4420       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4421         {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4422           {}% 123=F - nothing!
4423           {\bbl@exp{% 3=T - from generic
4424             \global\let\<bbl@##1dflt@language>%
4425             \<bbl@##1dflt@>}}}%
4426           {\bbl@exp{% 2=T - from script
4427             \global\let\<bbl@##1dflt@language>%
4428             \<bbl@##1dflt@*\bbl@tempa>}}}%
4429           {}}% 1=T - language, already defined
4430       \def\bbl@tempa{\bbl@nostdfont}}}%
4431     \bbl@foreach\bbl@font@fams{% don't gather with prev for
4432       \bbl@ifunset{bbl@##1dflt@language}%
4433       {\bbl@cs{famrst@##1}%
4434         \global\bbl@csarg\let{famrst@##1}\relax}%
4435       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4436         \\\bbl@add\\originalTeX{%
4437           \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4438           \<##1default>\<##1family>{##1}}%
4439           \\\bbl@font@set\<bbl@##1dflt@language>% the main part!
4440           \<##1default>\<##1family>}}}%
4441       \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.

```

4442 \ifx\family\undefined\else % if latex
4443 \ifcase\bbbl@engine % if pdftex
4444 \let\bbbl@ckeckstdfonts\relax
4445 \else
4446 \def\bbbl@ckeckstdfonts{%
4447 \begingroup
4448 \global\let\bbbl@ckeckstdfonts\relax
4449 \let\bbbl@tempa\@empty
4450 \bbbl@foreach\bbbl@font@fams{%
4451 \bbbl@ifunset{\bbbl@##1dflt@}%
4452 {\nameuse{##1family}%
4453 \bbbl@csarg\gdef{WFF@\family}}}% Flag
4454 \bbbl@exp{\bbbl@add\bbbl@tempa{* \<##1family>= \family\\}%
4455 \space\space\fontname\font\\}%
4456 \bbbl@csarg\xdef{##1dflt@}{\family}%
4457 \expandafter\xdef\csname ##1default\endcsname{\family}}%
4458 {}}%
4459 \ifx\bbbl@tempa\@empty\else
4460 \bbbl@infowarn{The following font families will use the default\\%
4461 settings for all or some languages:\\%
4462 \bbbl@tempa
4463 There is nothing intrinsically wrong with it, but\\%
4464 'babel' will no set Script and Language, which could\\%
4465 be relevant in some languages. If your document uses\\%
4466 these families, consider redefining them with \string\babelfont.\\%
4467 Reported}%
4468 \fi
4469 \endgroup}
4470 \fi
4471 \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 \bbbl@mapselect because \selectfont is called internally when a font is defined.

```

4472 \def\bbbl@font@set#1#2#3{% eg \bbbl@rmdflt@lang \rmdefault \rmfamily
4473 \bbbl@xin@{<>}{#1}%
4474 \ifin@
4475 \bbbl@exp{\bbbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4476 \fi
4477 \bbbl@exp{%
4478 \def\#2#1% eg, \rmdefault{\bbbl@rmdflt@lang}
4479 \bbbl@ifsamestring{#2}{\family}%
4480 {\#3%
4481 \bbbl@ifsamestring{\family}{\bfdefault}{\bfseries}}}%
4482 \let\bbbl@tempa\relax}%
4483 {}}}
4484 % TODO - next should be global?, but even local does its job. I'm
4485 % still not sure -- must investigate:
4486 \def\bbbl@fontspec@set#1#2#3#4{% eg \bbbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4487 \let\bbbl@tempa\bbbl@mapselect
4488 \let\bbbl@mapselect\relax
4489 \let\bbbl@tempa@fam#4% eg, '\rmfamily', to be restored below
4490 \let#4\@empty % Make sure \renewfontfamily is valid
4491 \bbbl@exp{%
4492 \let\bbbl@tempa@pfam\<\bbbl@stripslash#4\space>% eg, '\rmfamily '
4493 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbbl@cl{sname}}}%
4494 {\newfontscript{\bbbl@cl{sname}}{\bbbl@cl{sotf}}}%
4495 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbbl@cl{lname}}}%

```

```

4496     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4497     \renewfontfamily\#4%
4498     [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{..}{#3}
4499 \begingroup
4500   #4%
4501   \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4502 \endgroup
4503 \let#4\bbl@temp@fam
4504 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4505 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4506 \def\bbl@font@rst#1#2#3#4{%
4507   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4508 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4509 \newcommand\babelFSstore[2][{%
4510   \bbl@ifblank{#1}%
4511   {\bbl@csarg\def{sname@#2}{Latin}}%
4512   {\bbl@csarg\def{sname@#2}{#1}}%
4513   \bbl@provide@dirs{#2}%
4514   \bbl@csarg\ifnum{wdir@#2}>\z@
4515     \let\bbl@beforeforeign\leavevmode
4516     \EnableBabelHook{babel-bidi}%
4517   \fi
4518   \bbl@foreach{#2}{%
4519     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4520     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4521     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4522 \def\bbl@FSstore#1#2#3#4{%
4523   \bbl@csarg\edef{#2default#1}{#3}%
4524   \expandafter\addto\csname extras#1\endcsname{%
4525     \let#4#3%
4526     \ifx#3\f@family
4527       \edef#3{\csname bbl@#2default#1\endcsname}%
4528       \fontfamily{#3}\selectfont
4529     \else
4530       \edef#3{\csname bbl@#2default#1\endcsname}%
4531     \fi}%
4532   \expandafter\addto\csname noextras#1\endcsname{%
4533     \ifx#3\f@family
4534       \fontfamily{#4}\selectfont
4535     \fi
4536     \let#3#4}}
4537 \let\bbl@langfeatures\@empty
4538 \def\babelFSfeatures{% make sure \fontspec is redefined once
4539   \let\bbl@ori@fontspec\fontspec
4540   \renewcommand\fontspec[1][{%
4541     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4542   \let\babelFSfeatures\bbl@FSfeatures
4543   \babelFSfeatures}
4544 \def\bbl@FSfeatures#1#2{%
4545   \expandafter\addto\csname extras#1\endcsname{%

```

```

4546 \babel@save\bbl@langfeatures
4547 \edef\bbl@langfeatures{#2,}}
4548 <</Font selection>>

```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

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

```

4549 <<{*Footnote changes}>> ≡
4550 \bbl@trace{Bidi footnotes}
4551 \ifnum\bbl@bidimode>\z@
4552 \def\bbl@footnote#1#2#3{%
4553 \ifnextchar[%
4554 {\bbl@footnote@o{#1}{#2}{#3}}%
4555 {\bbl@footnote@x{#1}{#2}{#3}}}
4556 \long\def\bbl@footnote@x#1#2#3#4{%
4557 \bgroup
4558 \select@language@x{\bbl@main@language}%
4559 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4560 \egroup}
4561 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4562 \bgroup
4563 \select@language@x{\bbl@main@language}%
4564 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4565 \egroup}
4566 \def\bbl@footnotetext#1#2#3{%
4567 \ifnextchar[%
4568 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4569 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4570 \long\def\bbl@footnotetext@x#1#2#3#4{%
4571 \bgroup
4572 \select@language@x{\bbl@main@language}%
4573 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4574 \egroup}
4575 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4576 \bgroup
4577 \select@language@x{\bbl@main@language}%
4578 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4579 \egroup}
4580 \def\BabelFootnote#1#2#3#4{%
4581 \ifx\bbl@fn@footnote\@undefined
4582 \let\bbl@fn@footnote\footnote
4583 \fi
4584 \ifx\bbl@fn@footnotetext\@undefined
4585 \let\bbl@fn@footnotetext\footnotetext
4586 \fi
4587 \bbl@ifblank{#2}%
4588 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4589 \namedef{\bbl@stripslash#1text}%
4590 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4591 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4592 \namedef{\bbl@stripslash#1text}%
4593 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4594 \fi
4595 <</Footnote changes>>

```

Now, the code.

```
4596 (*xetex)
4597 \def\BabelStringsDefault{unicode}
4598 \let\xebbl@stop\relax
4599 \AddBabelHook{xetex}{encodedcommands}{%
4600   \def\bbl@tempa{#1}%
4601   \ifx\bbl@tempa@empty
4602     \XeTeXinputencoding"bytes"%
4603   \else
4604     \XeTeXinputencoding"#1"%
4605   \fi
4606   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4607 \AddBabelHook{xetex}{stopcommands}{%
4608   \xebbl@stop
4609   \let\xebbl@stop\relax}
4610 \def\bbl@intraspace#1 #2 #3@@{%
4611   \bbl@csarg\gdef{xeisp@\language}%
4612     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4613 \def\bbl@intrapenalty#1@@{%
4614   \bbl@csarg\gdef{xeipn@\language}%
4615     {\XeTeXlinebreakpenalty #1\relax}}
4616 \def\bbl@provide@intraspace{%
4617   \bbl@xin{/s}{/\bbl@cl{lnbrk}}%
4618   \ifin@else\bbl@xin{/c}{/\bbl@cl{lnbrk}}\fi
4619   \ifin@
4620     \bbl@ifunset{bbl@intsp@\language}{}%
4621     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4622       \ifx\bbl@KVP@intraspace\@nil
4623         \bbl@exp{%
4624           \\bbl@intraspace\bbl@cl{intsp}\\\@}%
4625         \fi
4626         \ifx\bbl@KVP@intrapenalty\@nil
4627           \bbl@intrapenalty0\@@
4628         \fi
4629       \fi
4630       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4631         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4632       \fi
4633       \ifx\bbl@KVP@intrapenalty\@nil\else
4634         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4635       \fi
4636       \bbl@exp{%
4637         % TODO. Execute only once (but redundant):
4638         \\bbl@add\<extras\language>{%
4639           \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4640           \<bbl@xeisp@\language>%
4641           \<bbl@xeipn@\language>%
4642           \\bbl@toglobal\<extras\language>%
4643           \\bbl@add\<noextras\language>{%
4644             \XeTeXlinebreaklocale "en"%
4645             \\bbl@toglobal\<noextras\language>%
4646           \ifx\bbl@ispace\undefined
4647             \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4648             \ifx\AtBeginDocument\@notprerr
4649               \expandafter\@secondoftwo % to execute right now
4650             \fi
4651             \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4652           \fi}%
```

```

4653 \fi}
4654 \ifx\DisableBabelHook\undefined\endinput\fi
4655 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4656 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4657 \DisableBabelHook{babel-fontspec}
4658 <<Font selection>>
4659 \input txtbabel.def
4660 </xetex>

```

13.2 Layout

In progress.

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

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

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

```

4661 (*texxet)
4662 \providecommand\bbl@provide@intraspace{}
4663 \bbl@trace{Redefinitions for bidi layout}
4664 \def\bbl@sspre@caption{%
4665   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4666 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4667 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4668 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4669 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4670   \def\hangfrom#1{%
4671     \setbox\@tempboxa\hbox{#1}}%
4672     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4673     \noindent\box\@tempboxa}
4674 \def\raggedright{%
4675   \let\@centercr
4676   \bbl@startskip\z@skip
4677   \@rightskip\@flushglue
4678   \bbl@endskip\@rightskip
4679   \parindent\z@
4680   \parfillskip\bbl@startskip}
4681 \def\raggedleft{%
4682   \let\@centercr
4683   \bbl@startskip\@flushglue
4684   \bbl@endskip\z@skip
4685   \parindent\z@
4686   \parfillskip\bbl@endskip}
4687 \fi
4688 \IfBabelLayout{lists}
4689   {\bbl@sreplace\list
4690     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4691     \def\bbl@listleftmargin{%
4692       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4693     \ifcase\bbl@engine
4694       \def\labelenumii{}\theenumii{}\% pdfTeX doesn't reverse ()
4695       \def\p@enumiii{\p@enumii}\theenumii{}\%
4696     \fi
4697     \bbl@sreplace\@verbatim
4698     {\leftskip\@totalleftmargin}%
4699     {\bbl@startskip\textwidth
4700       \advance\bbl@startskip-\linewidth}%

```



```

4701 \bbl@sreplace\@verbatim
4702 {\rightskip\z@skip}%
4703 {\bbl@endskip\z@skip}}%
4704 {}
4705 \IfBabelLayout{contents}
4706 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4707 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4708 {}
4709 \IfBabelLayout{columns}
4710 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4711 \def\bbl@outputbox#1{%
4712 \hb@xt@\textwidth{%
4713 \hskip\columnwidth
4714 \hfil
4715 {\normalcolor\vrule \@width\columnseprule}%
4716 \hfil
4717 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4718 \hskip-\textwidth
4719 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4720 \hskip\columnsep
4721 \hskip\columnwidth}}}%
4722 {}
4723 <<Footnote changes>>
4724 \IfBabelLayout{footnotes}%
4725 {\BabelFootnote\footnote\languagename{}}{}%
4726 \BabelFootnote\localfootnote\languagename{}}{}%
4727 \BabelFootnote\mainfootnote{}}{}%
4728 {}

```

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.

```

4729 \IfBabelLayout{counters}%
4730 {\let\bbl@latinarabic=\@arabic
4731 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4732 \let\bbl@asciroman=\@roman
4733 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4734 \let\bbl@asciiRoman=\@Roman
4735 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4736 </texxet>

```

13.3 LuaTeX

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

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

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

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

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

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

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

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

```

4737 < *luatex >
4738 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4739 \bbl@trace{Read language.dat}
4740 \ifx\bbl@readstream\undefined
4741   \csname newread\endcsname\bbl@readstream
4742 \fi
4743 \begingroup
4744   \toks@{}
4745   \count@ \z@ % 0=start, 1=0th, 2=normal
4746   \def\bbl@process@line#1#2 #3 #4 {%
4747     \ifx=#1%
4748       \bbl@process@synonym{#2}%
4749     \else
4750       \bbl@process@language{#1#2}{#3}{#4}%
4751     \fi
4752     \ignorespaces}
4753   \def\bbl@manylang{%
4754     \ifnum\bbl@last>\@ne
4755       \bbl@info{Non-standard hyphenation setup}%
4756     \fi
4757     \let\bbl@manylang\relax}
4758   \def\bbl@process@language#1#2#3{%
4759     \ifcase\count@
4760       \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4761     \or
4762       \count@\tw@
4763     \fi
4764     \ifnum\count@=\tw@
4765       \expandafter\addlanguage\csname l@#1\endcsname
4766       \language\allocationnumber
4767       \chardef\bbl@last\allocationnumber
4768       \bbl@manylang
4769       \let\bbl@elt\relax
4770     \xdef\bbl@languages{%
4771       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4772     \fi
4773     \the\toks@
4774     \toks@{}}
4775   \def\bbl@process@synonym@aux#1#2{%
4776     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4777     \let\bbl@elt\relax
4778     \xdef\bbl@languages{%
4779       \bbl@languages\bbl@elt{#1}{#2}{}}}%

```

```

4780 \def\bbl@process@synonym#1{%
4781   \ifcase\count@
4782     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4783   \or
4784     \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4785   \else
4786     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4787   \fi}
4788 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4789   \chardef\l@english\z@
4790   \chardef\l@USenglish\z@
4791   \chardef\bbl@last\z@
4792   \global\@namedef{\bbl@hyphendata@0}{\hyphen.tex}{}
4793   \gdef\bbl@languages{%
4794     \bbl@elt{english}{0}{\hyphen.tex}{}%
4795     \bbl@elt{USenglish}{0}{\hyphen.tex}{}%
4796   \else
4797     \global\let\bbl@languages@format\bbl@languages
4798     \def\bbl@elt#1#2#3#4{% Remove all except language 0
4799       \ifnum#2>\z@\else
4800         \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4801       \fi}%
4802     \xdef\bbl@languages{\bbl@languages}%
4803   \fi
4804   \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4805   \bbl@languages
4806   \openin\bbl@readstream=language.dat
4807   \ifeof\bbl@readstream
4808     \bbl@warning{I couldn't find language.dat. No additional\\%
4809       patterns loaded. Reported}%
4810   \else
4811     \loop
4812       \endlinechar\m@ne
4813       \read\bbl@readstream to \bbl@line
4814       \endlinechar\^^M
4815       \if T\ifeof\bbl@readstream F\fi T\relax
4816       \ifx\bbl@line\@empty\else
4817         \edef\bbl@line{\bbl@line\space\space\space}%
4818         \expandafter\bbl@process@line\bbl@line\relax
4819       \fi
4820     \repeat
4821   \fi
4822 \endgroup
4823 \bbl@trace{Macros for reading patterns files}
4824 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4825 \ifx\babelcatcodetablenum\@undefined
4826   \ifx\newcatcodetable\@undefined
4827     \def\babelcatcodetablenum{5211}
4828     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4829   \else
4830     \newcatcodetable\babelcatcodetablenum
4831     \newcatcodetable\bbl@pattcodes
4832   \fi
4833 \else
4834   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4835 \fi
4836 \def\bbl@luapatterns#1#2{%
4837   \bbl@get@enc#1::\@@
4838   \setbox\z@\hbox\bgroup

```

```

4839 \begingroup
4840 \savecatcodetable\babelcatcodetablenum\relax
4841 \initcatcodetable\bbl@pattcodes\relax
4842 \catcodetable\bbl@pattcodes\relax
4843 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4844 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
4845 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4846 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4847 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4848 \catcode\`=12 \catcode\`=12 \catcode\"=12
4849 \input #1\relax
4850 \catcodetable\babelcatcodetablenum\relax
4851 \endgroup
4852 \def\bbl@tempa{#2}%
4853 \ifx\bbl@tempa@empty\else
4854 \input #2\relax
4855 \fi
4856 \egroup}%
4857 \def\bbl@patterns@lua#1{%
4858 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4859 \csname l@#1\endcsname
4860 \edef\bbl@tempa{#1}%
4861 \else
4862 \csname l@#1:\f@encoding\endcsname
4863 \edef\bbl@tempa{#1:\f@encoding}%
4864 \fi\relax
4865 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4866 \@ifundefined{bbl@hyphendata@the\language}%
4867 {\def\bbl@elt##1##2##3##4{%
4868 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4869 \def\bbl@tempb{##3}%
4870 \ifx\bbl@tempb@empty\else % if not a synonymous
4871 \def\bbl@tempc{##3}{##4}%
4872 \fi
4873 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4874 \fi}%
4875 \bbl@languages
4876 \@ifundefined{bbl@hyphendata@the\language}%
4877 {\bbl@info{No hyphenation patterns were set for\the
4878 language '\bbl@tempa'. Reported}}%
4879 {\expandafter\expandafter\expandafter\bbl@luapatterns
4880 \csname bbl@hyphendata@the\language\endcsname}}}%
4881 \endinput\fi
4882 % Here ends \ifx\AddBabelHook\@undefined
4883 % A few lines are only read by hyphen.cfg
4884 \ifx\DisableBabelHook\@undefined
4885 \AddBabelHook{luatex}{everylanguage}{%
4886 \def\process@language##1##2##3{%
4887 \def\process@line####1####2 ####3 ####4 {}}}
4888 \AddBabelHook{luatex}{loadpatterns}{%
4889 \input #1\relax
4890 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4891 {{#1}}}%
4892 \AddBabelHook{luatex}{loadexceptions}{%
4893 \input #1\relax
4894 \def\bbl@tempb##1##2{{#1}{#1}}%
4895 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4896 {\expandafter\expandafter\expandafter\bbl@tempb
4897 \csname bbl@hyphendata@the\language\endcsname}}

```

```

4898 \endinput\fi
4899 % Here stops reading code for hyphen.cfg
4900 % The following is read the 2nd time it's loaded
4901 \begingroup % TODO - to a lua file
4902 \catcode`\%=12
4903 \catcode`\'=12
4904 \catcode`\`=12
4905 \catcode`\:=12
4906 \directlua{
4907   Babel = Babel or {}
4908   function Babel.bytes(line)
4909     return line:gsub(".",
4910       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4911   end
4912   function Babel.begin_process_input()
4913     if luatexbase and luatexbase.add_to_callback then
4914       luatexbase.add_to_callback('process_input_buffer',
4915         Babel.bytes, 'Babel.bytes')
4916     else
4917       Babel.callback = callback.find('process_input_buffer')
4918       callback.register('process_input_buffer', Babel.bytes)
4919     end
4920   end
4921   function Babel.end_process_input ()
4922     if luatexbase and luatexbase.remove_from_callback then
4923       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4924     else
4925       callback.register('process_input_buffer', Babel.callback)
4926     end
4927   end
4928   function Babel.addpatterns(pp, lg)
4929     local lg = lang.new(lg)
4930     local pats = lang.patterns(lg) or ''
4931     lang.clear_patterns(lg)
4932     for p in pp:gmatch('[^%s]+') do
4933       ss = ''
4934       for i in string.utfcharacters(p:gsub('%d', '')) do
4935         ss = ss .. '%d?' .. i
4936       end
4937       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4938       ss = ss:gsub('%.%%d%?$', '%%.')
4939       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4940       if n == 0 then
4941         tex.sprint(
4942           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4943           .. p .. [[{}]])
4944         pats = pats .. ' ' .. p
4945       else
4946         tex.sprint(
4947           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4948           .. p .. [[{}]])
4949       end
4950     end
4951     lang.patterns(lg, pats)
4952   end
4953 }
4954 \endgroup
4955 \ifx\newattribute\@undefined\else
4956   \newattribute\bbl@attr@locale

```

```

4957 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
4958 \AddBabelHook{luatex}{beforeextras}{%
4959   \setattribute\bbl@attr@locale\localeid}
4960 \fi
4961 \def\BabelStringsDefault{unicode}
4962 \let\luabbbl@stop\relax
4963 \AddBabelHook{luatex}{encodedcommands}{%
4964   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4965   \ifx\bbl@tempa\bbl@tempb\else
4966     \directlua{Babel.begin_process_input()}%
4967     \def\luabbbl@stop{%
4968       \directlua{Babel.end_process_input()}}%
4969   \fi}%
4970 \AddBabelHook{luatex}{stopcommands}{%
4971   \luabbbl@stop
4972   \let\luabbbl@stop\relax}
4973 \AddBabelHook{luatex}{patterns}{%
4974   \@ifundefined{bbl@hyphendata@the\language}%
4975     {\def\bbl@elt##1##2##3##4{%
4976       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4977       \def\bbl@tempb{##3}%
4978       \ifx\bbl@tempb\empty\else % if not a synonymous
4979         \def\bbl@tempc{##3}{##4}}%
4980       \fi
4981       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4982     \fi}%
4983   \bbl@languages
4984   \@ifundefined{bbl@hyphendata@the\language}%
4985     {\bbl@info{No hyphenation patterns were set for\%
4986       language '#2'. Reported}}%
4987     {\expandafter\expandafter\expandafter\bbl@luapatterns
4988       \csname bbl@hyphendata@the\language\endcsname}}}%
4989   \@ifundefined{bbl@patterns@}{}%
4990   \begingroup
4991     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4992     \ifin@else
4993       \ifx\bbl@patterns@\empty\else
4994         \directlua{ Babel.addpatterns(
4995           [[\bbl@patterns@]], \number\language) }%
4996       \fi
4997       \@ifundefined{bbl@patterns@#1}%
4998       \empty
4999       {\directlua{ Babel.addpatterns(
5000         [[\space\csname bbl@patterns@#1\endcsname]],
5001         \number\language) }}%
5002       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5003     \fi
5004   \endgroup}%
5005 \bbl@exp{%
5006   \bbl@ifunset{bbl@prehc@\languagename}{}%
5007   {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5008   {\prehyphenchar=\bbl@c1{prehc}\relax}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

5009 \@onlypreamble\babelpatterns
5010 \AtEndOfPackage{%
5011   \newcommand\babelpatterns[2][\empty]{%

```

```

5012 \ifx\bbbl@patterns@\relax
5013 \let\bbbl@patterns@\@empty
5014 \fi
5015 \ifx\bbbl@pttnlist\@empty\else
5016 \bbbl@warning{%
5017   You must not intermingle \string\selectlanguage\space and\\%
5018   \string\babelpatterns\space or some patterns will not\\%
5019   be taken into account. Reported}%
5020 \fi
5021 \ifx\@empty#1%
5022 \protected@edef\bbbl@patterns@{\bbbl@patterns@\space#2}%
5023 \else
5024 \edef\bbbl@tempb{\zap@space#1 \@empty}%
5025 \bbbl@for\bbbl@tempa\bbbl@tempb{%
5026 \bbbl@fixname\bbbl@tempa
5027 \bbbl@iflanguage\bbbl@tempa{%
5028 \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
5029 \@ifundefined{bbbl@patterns@\bbbl@tempa}%
5030 \@empty
5031 {\csname bbl@patterns@\bbbl@tempa\endcsname\space}%
5032 #2}}}%
5033 \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```

5034% TODO - to a lua file
5035 \directlua{
5036   Babel = Babel or {}
5037   Babel.linebreaking = Babel.linebreaking or {}
5038   Babel.linebreaking.before = {}
5039   Babel.linebreaking.after = {}
5040   Babel.locale = {} % Free to use, indexed by \localeid
5041   function Babel.linebreaking.add_before(func)
5042     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5043     table.insert(Babel.linebreaking.before, func)
5044   end
5045   function Babel.linebreaking.add_after(func)
5046     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5047     table.insert(Babel.linebreaking.after, func)
5048   end
5049 }
5050 \def\bbbl@intraspace#1 #2 #3\@{#%
5051 \directlua{
5052   Babel = Babel or {}
5053   Babel.intraspaces = Babel.intraspaces or {}
5054   Babel.intraspaces['\csname bbl@sbcpr@\language\endcsname'] = %
5055     {b = #1, p = #2, m = #3}
5056   Babel.locale_props[\the\localeid].intraspace = %
5057     {b = #1, p = #2, m = #3}
5058 }}
5059 \def\bbbl@intrapenalty#1\@{#%
5060 \directlua{
5061   Babel = Babel or {}
5062   Babel.intrapenalties = Babel.intrapenalties or {}

```

```

5063     Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5064     Babel.locale_props[\the\localeid].intrapenalty = #1
5065   }}
5066 \begingroup
5067 \catcode`\%=12
5068 \catcode`\^=14
5069 \catcode`\'=12
5070 \catcode`\~=12
5071 \gdef\bbl@seaintraspace{^
5072   \let\bbl@seaintraspace\relax
5073   \directlua{
5074     Babel = Babel or {}
5075     Babel.sea_enabled = true
5076     Babel.sea_ranges = Babel.sea_ranges or {}
5077     function Babel.set_chranges (script, chrng)
5078       local c = 0
5079       for s, e in string.gmatch(chrng..' ', '(.)%.%.(.)%s') do
5080         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5081         c = c + 1
5082       end
5083     end
5084     function Babel.sea_disc_to_space (head)
5085       local sea_ranges = Babel.sea_ranges
5086       local last_char = nil
5087       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5088       for item in node.traverse(head) do
5089         local i = item.id
5090         if i == node.id'glyph' then
5091           last_char = item
5092         elseif i == 7 and item.subtype == 3 and last_char
5093           and last_char.char > 0x0C99 then
5094           quad = font.getfont(last_char.font).size
5095           for lg, rg in pairs(sea_ranges) do
5096             if last_char.char > rg[1] and last_char.char < rg[2] then
5097               lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5098               local intraspace = Babel.intraspaces[lg]
5099               local intrapenalty = Babel.intrapenalties[lg]
5100               local n
5101               if intrapenalty ~= 0 then
5102                 n = node.new(14, 0) ^% penalty
5103                 n.penalty = intrapenalty
5104                 node.insert_before(head, item, n)
5105               end
5106               n = node.new(12, 13) ^% (glue, spaceskip)
5107               node.setglue(n, intraspace.b * quad,
5108                 intraspace.p * quad,
5109                 intraspace.m * quad)
5110               node.insert_before(head, item, n)
5111               node.remove(head, item)
5112             end
5113           end
5114         end
5115       end
5116     end
5117   }^^
5118   \bbl@luahyphenate}

```


13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

```
5119 \catcode`\%=14
5120 \gdef\bbl@cjkintraspacespace{%
5121   \let\bbl@cjkintraspacespace\relax
5122   \directlua{
5123     Babel = Babel or {}
5124     require('babel-data-cjk.lua')
5125     Babel.cjk_enabled = true
5126     function Babel.cjk_linebreak(head)
5127       local GLYPH = node.id'glyph'
5128       local last_char = nil
5129       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5130       local last_class = nil
5131       local last_lang = nil
5132
5133       for item in node.traverse(head) do
5134         if item.id == GLYPH then
5135
5136           local lang = item.lang
5137
5138           local LOCALE = node.get_attribute(item,
5139             Babel.attr_locale)
5140           local props = Babel.locale_props[LOCALE]
5141
5142           local class = Babel.cjk_class[item.char].c
5143
5144           if props.cjk_quotes and props.cjk_quotes[item.char] then
5145             class = props.cjk_quotes[item.char]
5146           end
5147
5148           if class == 'cp' then class = 'cl' end % ]] as CL
5149           if class == 'id' then class = 'I' end
5150
5151           local br = 0
5152           if class and last_class and Babel.cjk_breaks[last_class][class] then
5153             br = Babel.cjk_breaks[last_class][class]
5154           end
5155
5156           if br == 1 and props.linebreak == 'c' and
5157             lang ~= \the\l@nohyphenation\space and
5158             last_lang ~= \the\l@nohyphenation then
5159             local intrapenalty = props.intrapenalty
5160             if intrapenalty ~= 0 then
5161               local n = node.new(14, 0)      % penalty
5162               n.penalty = intrapenalty
5163               node.insert_before(head, item, n)
5164             end
5165             local intraspacespace = props.intraspacespace
5166             local n = node.new(12, 13)      % (glue, spaceskip)
5167             node.setglue(n, intraspacespace.b * quad,
5168               intraspacespace.p * quad,
5169               intraspacespace.m * quad)
```

```

5170         node.insert_before(head, item, n)
5171     end
5172
5173     if font.getfont(item.font) then
5174         quad = font.getfont(item.font).size
5175     end
5176     last_class = class
5177     last_lang = lang
5178     else % if penalty, glue or anything else
5179         last_class = nil
5180     end
5181 end
5182 lang.hyphenate(head)
5183 end
5184 }%
5185 \bbl@luahyphenate}
5186 \gdef\bbl@luahyphenate{%
5187 \let\bbl@luahyphenate\relax
5188 \directlua{
5189     luatexbase.add_to_callback('hyphenate',
5190     function (head, tail)
5191         if Babel.linebreaking.before then
5192             for k, func in ipairs(Babel.linebreaking.before) do
5193                 func(head)
5194             end
5195         end
5196         if Babel.cjk_enabled then
5197             Babel.cjk_linebreak(head)
5198         end
5199         lang.hyphenate(head)
5200         if Babel.linebreaking.after then
5201             for k, func in ipairs(Babel.linebreaking.after) do
5202                 func(head)
5203             end
5204         end
5205         if Babel.sea_enabled then
5206             Babel.sea_disc_to_space(head)
5207         end
5208     end,
5209     'Babel.hyphenate')
5210 }
5211 }
5212 \endgroup
5213 \def\bbl@provide@intraspace{%
5214 \bbl@ifunset{\bbl@intsp@language}{}%
5215 {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5216 \bbl@xin@{/c}{\bbl@cl{lnbrk}}}%
5217 \ifin@ % cjk
5218 \bbl@cjkintraspace
5219 \directlua{
5220     Babel = Babel or {}
5221     Babel.locale_props = Babel.locale_props or {}
5222     Babel.locale_props[\the\localeid].linebreak = 'c'
5223 }%
5224 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5225 \ifx\bbl@KVP@intrapenalty\@nil
5226 \bbl@intrapenalty0\@
5227 \fi
5228 \else % sea

```

```

5229      \bbl@seaintraspace
5230      \bbl@exp{\bbl@intraspace\bbl@c1{intsp}\bbl@@}%
5231      \directlua{
5232          Babel = Babel or {}
5233          Babel.sea_ranges = Babel.sea_ranges or {}
5234          Babel.set_chranges('\bbl@c1{sbcpr}',
5235                          '\bbl@c1{chrng}')
5236      }%
5237      \ifx\bbl@KVP@intrapenalty\@nil
5238          \bbl@intrapenalty0\@@
5239      \fi
5240  \fi
5241  \fi
5242  \ifx\bbl@KVP@intrapenalty\@nil\else
5243      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5244  \fi}}

```

13.6 Arabic justification

```

5245 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5246 \def\bblar@chars{%
5247   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5248   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5249   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5250 \def\bblar@elongated{%
5251   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5252   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5253   0649,064A}
5254 \begingroup
5255   \catcode\_ =11 \catcode\` =11
5256   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5257 \endgroup
5258 \gdef\bbl@arabicjust{%
5259   \let\bbl@arabicjust\relax
5260   \newattribute\bblar@kashida
5261   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5262   \bblar@kashida=\z@
5263   \bbl@patchfont{\bbl@parsejalt}}%
5264   \directlua{
5265     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5266     Babel.arabic.elong_map[\the\localeid] = {}
5267     luatexbase.add_to_callback('post_linebreak_filter',
5268     Babel.arabic.justify, 'Babel.arabic.justify')
5269     luatexbase.add_to_callback('hpack_filter',
5270     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5271   }}%
5272 % Save both node lists to make replacement. TODO. Save also widths to
5273 % make computations
5274 \def\bblar@fetchjalt#1#2#3#4{%
5275   \bbl@exp{\bbl@foreach{#1}}{%
5276     \bbl@ifunset\bblar@JE@##1}%
5277     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5278     {\setbox\z@\hbox{^^^200d\char"\@nameuse\bblar@JE@##1#2}}}%
5279   \directlua{%
5280     local last = nil
5281     for item in node.traverse(tex.box[0].head) do
5282       if item.id == node.id'glyph' and item.char > 0x600 and
5283       not (item.char == 0x200D) then
5284         last = item

```

```

5285         end
5286     end
5287     Babel.arabic.#3['##1#4'] = last.char
5288 }}}}
5289 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5290 % perhaps other tables (falt?, cswb?). What about kaf? And diacritic
5291 % positioning?
5292 \gdef\bbl@parsejalt{%
5293     \ifx\addfontfeature\undefined\else
5294         \bbl@xin@{/e}{/\bbl@cl{lbrk}}}%
5295     \ifin@
5296         \directlua{%
5297             if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5298                 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5299                 tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5300             end
5301         }%
5302     \fi
5303 \fi}
5304 \gdef\bbl@parsejalti{%
5305     \beginingroup
5306         \let\bbl@parsejalt\relax    % To avoid infinite loop
5307         \edef\bbl@tempb{\fontid\font}%
5308         \bblar@nofswarn
5309         \bblar@fetchjalt\bblar@elongated{}{from}{}%
5310         \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5311         \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5312         \addfontfeature{RawFeature+=jalt}%
5313         % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5314         \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5315         \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5316         \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5317         \directlua{%
5318             for k, v in pairs(Babel.arabic.from) do
5319                 if Babel.arabic.dest[k] and
5320                     not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5321                     Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5322                     [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5323                 end
5324             end
5325         }%
5326     \endgroup}
5327 %
5328 \beginingroup
5329 \catcode`#=11
5330 \catcode`~=11
5331 \directlua{
5332
5333 Babel.arabic = Babel.arabic or {}
5334 Babel.arabic.from = {}
5335 Babel.arabic.dest = {}
5336 Babel.arabic.justify_factor = 0.95
5337 Babel.arabic.justify_enabled = true
5338
5339 function Babel.arabic.justify(head)
5340     if not Babel.arabic.justify_enabled then return head end
5341     for line in node.traverse_id(node.id'hlist', head) do
5342         Babel.arabic.justify_hlist(head, line)
5343     end

```

```

5344 return head
5345 end
5346
5347 function Babel.arabic.justify_hbox(head, gc, size, pack)
5348   local has_inf = false
5349   if Babel.arabic.justify_enabled and pack == 'exactly' then
5350     for n in node.traverse_id(12, head) do
5351       if n.stretch_order > 0 then has_inf = true end
5352     end
5353     if not has_inf then
5354       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5355     end
5356   end
5357   return head
5358 end
5359
5360 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5361   local d, new
5362   local k_list, k_item, pos_inline
5363   local width, width_new, full, k_curr, wt_pos, goal, shift
5364   local subst_done = false
5365   local elong_map = Babel.arabic.elong_map
5366   local last_line
5367   local GLYPH = node.id'glyph'
5368   local KASHIDA = Babel.attr_kashida
5369   local LOCALE = Babel.attr_locale
5370
5371   if line == nil then
5372     line = {}
5373     line.glue_sign = 1
5374     line.glue_order = 0
5375     line.head = head
5376     line.shift = 0
5377     line.width = size
5378   end
5379
5380   % Exclude last line. todo. But-- it discards one-word lines, too!
5381   % ? Look for glue = 12:15
5382   if (line.glue_sign == 1 and line.glue_order == 0) then
5383     elongs = {} % Stores elongated candidates of each line
5384     k_list = {} % And all letters with kashida
5385     pos_inline = 0 % Not yet used
5386
5387     for n in node.traverse_id(GLYPH, line.head) do
5388       pos_inline = pos_inline + 1 % To find where it is. Not used.
5389
5390       % Elongated glyphs
5391       if elong_map then
5392         local locale = node.get_attribute(n, LOCALE)
5393         if elong_map[locale] and elong_map[locale][n.font] and
5394           elong_map[locale][n.font][n.char] then
5395           table.insert(elongs, {node = n, locale = locale} )
5396           node.set_attribute(n.prev, KASHIDA, 0)
5397         end
5398       end
5399
5400       % Tatwil
5401       if Babel.kashida_wts then
5402         local k_wt = node.get_attribute(n, KASHIDA)

```

```

5403         if k_wt > 0 then % todo. parameter for multi inserts
5404             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5405         end
5406     end
5407
5408 end % of node.traverse_id
5409
5410 if #elongs == 0 and #k_list == 0 then goto next_line end
5411 full = line.width
5412 shift = line.shift
5413 goal = full * Babel.arabic.justify_factor % A bit crude
5414 width = node.dimensions(line.head) % The 'natural' width
5415
5416 % == Elongated ==
5417 % Original idea taken from 'chickenize'
5418 while (#elongs > 0 and width < goal) do
5419     subst_done = true
5420     local x = #elongs
5421     local curr = elongs[x].node
5422     local oldchar = curr.char
5423     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5424     width = node.dimensions(line.head) % Check if the line is too wide
5425     % Substitute back if the line would be too wide and break:
5426     if width > goal then
5427         curr.char = oldchar
5428         break
5429     end
5430     % If continue, pop the just substituted node from the list:
5431     table.remove(elongs, x)
5432 end
5433
5434 % == Tatwil ==
5435 if #k_list == 0 then goto next_line end
5436
5437 width = node.dimensions(line.head) % The 'natural' width
5438 k_curr = #k_list
5439 wt_pos = 1
5440
5441 while width < goal do
5442     subst_done = true
5443     k_item = k_list[k_curr].node
5444     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5445         d = node.copy(k_item)
5446         d.char = 0x0640
5447         line.head, new = node.insert_after(line.head, k_item, d)
5448         width_new = node.dimensions(line.head)
5449         if width > goal or width == width_new then
5450             node.remove(line.head, new) % Better compute before
5451             break
5452         end
5453         width = width_new
5454     end
5455     if k_curr == 1 then
5456         k_curr = #k_list
5457         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5458     else
5459         k_curr = k_curr - 1
5460     end
5461 end

```

```

5462
5463 ::next_line::
5464
5465 % Must take into account marks and ins, see luatex manual.
5466 % Have to be executed only if there are changes. Investigate
5467 % what's going on exactly.
5468 if subst_done and not gc then
5469     d = node.hpack(line.head, full, 'exactly')
5470     d.shift = shift
5471     node.insert_before(head, line, d)
5472     node.remove(head, line)
5473 end
5474 end % if process line
5475 end
5476 }
5477 \endgroup
5478 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5479 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5480 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5481 \DisableBabelHook{babel-fontspec}
5482 <<Font selection>>

```

13.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5483 % TODO - to a lua file
5484 \directlua{
5485 Babel.script_blocks = {
5486     ['dflt'] = {},
5487     ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5488                {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5489     ['Armn'] = {{0x0530, 0x058F}},
5490     ['Beng'] = {{0x0980, 0x09FF}},
5491     ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5492     ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5493     ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5494                {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5495     ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5496     ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5497                {0xAB00, 0xAB2F}},
5498     ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5499     % Don't follow strictly Unicode, which places some Coptic letters in
5500     % the 'Greek and Coptic' block
5501     ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5502     ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5503                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5504                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5505                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5506                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5507                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5508     ['Hebr'] = {{0x0590, 0x05FF}},

```

```

5509 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5510           {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5511 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5512 ['Knda'] = {{0x0C80, 0x0CFF}},
5513 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5514           {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5515           {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5516 ['Lao'] = {{0x0E80, 0x0EFF}},
5517 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5518           {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5519           {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5520 ['Mahj'] = {{0x11150, 0x1117F}},
5521 ['Mlym'] = {{0x0D00, 0x0D7F}},
5522 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5523 ['Orya'] = {{0x0B00, 0x0B7F}},
5524 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5525 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5526 ['Taml'] = {{0x0B80, 0x0BFF}},
5527 ['Telu'] = {{0x0C00, 0x0C7F}},
5528 ['Tfng'] = {{0x2D30, 0x2D7F}},
5529 ['Thai'] = {{0x0E00, 0x0E7F}},
5530 ['Tibt'] = {{0x0F00, 0x0FFF}},
5531 ['Vaii'] = {{0xA500, 0xA63F}},
5532 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5533 }
5534
5535 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5536 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5537 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5538
5539 function Babel.locale_map(head)
5540   if not Babel.locale_mapped then return head end
5541
5542   local LOCALE = Babel.attr_locale
5543   local GLYPH = node.id('glyph')
5544   local inmath = false
5545   local toloc_save
5546   for item in node.traverse(head) do
5547     local toloc
5548     if not inmath and item.id == GLYPH then
5549       % Optimization: build a table with the chars found
5550       if Babel.chr_to_loc[item.char] then
5551         toloc = Babel.chr_to_loc[item.char]
5552       else
5553         for lc, maps in pairs(Babel.loc_to_scr) do
5554           for _, rg in pairs(maps) do
5555             if item.char >= rg[1] and item.char <= rg[2] then
5556               Babel.chr_to_loc[item.char] = lc
5557               toloc = lc
5558               break
5559             end
5560           end
5561         end
5562       end
5563       % Now, take action, but treat composite chars in a different
5564       % fashion, because they 'inherit' the previous locale. Not yet
5565       % optimized.
5566       if not toloc and
5567         (item.char >= 0x0300 and item.char <= 0x036F) or

```



```

5568         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5569         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5570         toloc = toloc_save
5571     end
5572     if toloc and toloc > -1 then
5573         if Babel.locale_props[toloc].lg then
5574             item.lang = Babel.locale_props[toloc].lg
5575             node.set_attribute(item, LOCALE, toloc)
5576         end
5577         if Babel.locale_props[toloc]['/'..item.font] then
5578             item.font = Babel.locale_props[toloc]['/'..item.font]
5579         end
5580         toloc_save = toloc
5581     end
5582     elseif not inmath and item.id == 7 then
5583         item.replace = item.replace and Babel.locale_map(item.replace)
5584         item.pre      = item.pre and Babel.locale_map(item.pre)
5585         item.post      = item.post and Babel.locale_map(item.post)
5586     elseif item.id == node.id'math' then
5587         inmath = (item.subtype == 0)
5588     end
5589 end
5590 return head
5591 end
5592 }

```

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

```

5593 \newcommand\babelcharproperty[1]{%
5594   \count@=#1\relax
5595   \ifvmode
5596     \expandafter\bbl@chprop
5597   \else
5598     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5599               vertical mode (preamble or between paragraphs)}%
5600     {See the manual for futher info}%
5601   \fi}
5602 \newcommand\bbl@chprop[3][\the\count@]{%
5603   \@tempcnta=#1\relax
5604   \bbl@ifunset{\bbl@chprop@#2}%
5605   {\bbl@error{No property named '#2'. Allowed values are\\%
5606               direction (bc), mirror (bmg), and linebreak (lb)}%
5607    {See the manual for futher info}}%
5608   {}%
5609   \loop
5610     \bbl@cs{chprop@#2}{#3}%
5611     \ifnum\count@<\@tempcnta
5612       \advance\count@\@ne
5613     \repeat}
5614 \def\bbl@chprop@direction#1{%
5615   \directlua{
5616     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5617     Babel.characters[\the\count@]['d'] = '#1'
5618   }}
5619 \let\bbl@chprop@bc\bbl@chprop@direction
5620 \def\bbl@chprop@mirror#1{%
5621   \directlua{
5622     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5623     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

5624 }}
5625 \let\bbl@chprop@bmg\bbl@chprop@mirror
5626 \def\bbl@chprop@linebreak#1{%
5627   \directlua{
5628     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5629     Babel.cjk_characters[\the\count@]['c'] = '#1'
5630   }}
5631 \let\bbl@chprop@lb\bbl@chprop@linebreak
5632 \def\bbl@chprop@locale#1{%
5633   \directlua{
5634     Babel.chr_to_loc = Babel.chr_to_loc or {}
5635     Babel.chr_to_loc[\the\count@] =
5636       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5637   }}

```

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.

```

5638 \directlua{
5639   Babel.nohyphenation = \the\l@nohyphenation
5640 }

```

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

```

5641 \begingroup
5642 \catcode`\~ = 12
5643 \catcode`\% = 12
5644 \catcode`\& = 14
5645 \gdef\babelposthyphenation#1#2#3{&%
5646   \bbl@activateposthyphen
5647   \begingroup
5648     \def\babeltempa{\bbl@add@list\babeltempb}&%
5649     \let\babeltempb\@empty
5650     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5651     \bbl@replace\bbl@tempa{,}{ ,}&%
5652     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5653       \bbl@ifsamestring{##1}{remove}&%
5654       {\bbl@add@list\babeltempb{nil}}&%
5655       {\directlua{
5656         local rep = {[##1]=}
5657         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5658         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5659         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5660         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5661         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5662         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5663         tex.print([[\\string\babeltempa{}}] .. rep .. [[{}]])
5664       }}&%
5665     \directlua{
5666       local lbkr = Babel.linebreaking.replacements[1]
5667       local u = unicode.utf8
5668       local id = \the\csname l@#1\endcsname
5669       &% Convert pattern:

```

```

5670     local patt = string.gsub(#[#2]==, '%s', '')
5671     if not u.find(patt, '()', nil, true) then
5672         patt = '()' .. patt .. '()'
5673     end
5674     patt = string.gsub(patt, '%(%)%^(', '^()')
5675     patt = string.gsub(patt, '%$(%)%', '()$')
5676     patt = u.gsub(patt, '{(.)}',
5677         function (n)
5678             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5679         end)
5680     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5681         function (n)
5682             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5683         end)
5684     lbkr[id] = lbkr[id] or {}
5685     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5686 }&%
5687 \endgroup}
5688 % TODO. Copypaste pattern.
5689 \gdef\babelprehyphenation#1#2#3{&%
5690 \bbl@activateprehyphen
5691 \begin{group}
5692 \def\babeltempa{\bbl@add@list\babeltempb}&%
5693 \let\babeltempb\@empty
5694 \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5695 \bbl@replace\bbl@tempa{,}{ ,}&%
5696 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5697 \bbl@ifsamestring{##1}{remove}&%
5698 {\bbl@add@list\babeltempb{nil}}&%
5699 {\directlua{
5700     local rep = [=[#1]=]
5701     rep = rep.gsub('^%s*(remove)%s*$', 'remove = true')
5702     rep = rep.gsub('^%s*(insert)%s*', 'insert = true, ')
5703     rep = rep.gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5704     rep = rep.gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5705         'space = { ' .. '%2, %3, %4' .. ' }')
5706     rep = rep.gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5707         'spacefactor = { ' .. '%2, %3, %4' .. ' }')
5708     rep = rep.gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5709     tex.print([[string\babeltempa{[]] .. rep .. [{}]])
5710 }}&%
5711 \directlua{
5712     local lbkr = Babel.linebreaking.replacements[0]
5713     local u = unicode.utf8
5714     local id = \the\csname bbl@id@#1\endcsname
5715     &% Convert pattern:
5716     local patt = string.gsub(#[#2]==, '%s', '')
5717     local patt = string.gsub(patt, '|', ' ')
5718     if not u.find(patt, '()', nil, true) then
5719         patt = '()' .. patt .. '()'
5720     end
5721     &% patt = string.gsub(patt, '%(%)%^(', '^()')
5722     &% patt = string.gsub(patt, '([^\s])%$(%)', '%1()$')
5723     patt = u.gsub(patt, '{(.)}',
5724         function (n)
5725             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5726         end)
5727     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5728         function (n)

```

```

5729             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5730         end)
5731         lbr[id] = lbr[id] or {}
5732         table.insert(lbr[id], { pattern = patt, replace = { \babeltempb } })
5733     }&%
5734 \endgroup}
5735 \endgroup
5736 \def\bbl@activateposthyphen{%
5737 \let\bbl@activateposthyphen\relax
5738 \directlua{
5739     require('babel-transforms.lua')
5740     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5741 }}
5742 \def\bbl@activateprehyphen{%
5743 \let\bbl@activateprehyphen\relax
5744 \directlua{
5745     require('babel-transforms.lua')
5746     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5747 }}

```

13.9 Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before luaotfload is applied, which is loaded by default by \LaTeX . Just in case, consider the possibility it has not been loaded.

```

5748 \def\bbl@activate@preotf{%
5749 \let\bbl@activate@preotf\relax % only once
5750 \directlua{
5751     Babel = Babel or {}
5752     %
5753     function Babel.pre_otfload_v(head)
5754         if Babel.numbers and Babel.digits_mapped then
5755             head = Babel.numbers(head)
5756         end
5757         if Babel.bidi_enabled then
5758             head = Babel.bidi(head, false, dir)
5759         end
5760         return head
5761     end
5762     %
5763     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5764         if Babel.numbers and Babel.digits_mapped then
5765             head = Babel.numbers(head)
5766         end
5767         if Babel.bidi_enabled then
5768             head = Babel.bidi(head, false, dir)
5769         end
5770         return head
5771     end
5772     %
5773     luatexbase.add_to_callback('pre_linebreak_filter',
5774         Babel.pre_otfload_v,
5775         'Babel.pre_otfload_v',
5776         luatexbase.priority_in_callback('pre_linebreak_filter',
5777             'luaotfload.node_processor') or nil)
5778     %
5779     luatexbase.add_to_callback('hpack_filter',
5780         Babel.pre_otfload_h,

```

```

5781     'Babel.pre_otfload_h',
5782     luatexbase.priority_in_callback('hpack_filter',
5783     'luaotfload.node_processor') or nil)
5784 }}

```

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=.

```

5785 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5786   \let\bbl@beforeforeign\leavevmode
5787   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5788   \RequirePackage{luatexbase}
5789   \bbl@activate@preotf
5790   \directlua{
5791     require('babel-data-bidi.lua')
5792     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5793       require('babel-bidi-basic.lua')
5794     \or
5795       require('babel-bidi-basic-r.lua')
5796     \fi}
5797   % TODO - to locale_props, not as separate attribute
5798   \newattribute\bbl@attr@dir
5799   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5800   % TODO. I don't like it, hackish:
5801   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5802   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5803 \fi\fi
5804 \chardef\bbl@thetextdir\z@
5805 \chardef\bbl@thepardir\z@
5806 \def\bbl@getluadir#1{%
5807   \directlua{
5808     if tex.#1dir == 'TLT' then
5809       tex.sprint('0')
5810     elseif tex.#1dir == 'TRT' then
5811       tex.sprint('1')
5812     end}}
5813 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5814   \ifcase#3\relax
5815     \ifcase\bbl@getluadir{#1}\relax\else
5816       #2 TLT\relax
5817     \fi
5818   \else
5819     \ifcase\bbl@getluadir{#1}\relax
5820       #2 TRT\relax
5821     \fi
5822   \fi}
5823 \def\bbl@textdir#1{%
5824   \bbl@setluadir{text}\textdir{#1}%
5825   \chardef\bbl@thetextdir#1\relax
5826   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5827 \def\bbl@pardir#1{%
5828   \bbl@setluadir{par}\pardir{#1}%
5829   \chardef\bbl@thepardir#1\relax}
5830 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5831 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5832 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
5833 %
5834 \ifnum\bbl@bidimode>\z@
5835   \def\bbl@mathboxdir{%

```

```

5836 \ifcase\bbl@thetextdir\relax
5837 \everyhbox{\bbl@mathboxdir@aux L}%
5838 \else
5839 \everyhbox{\bbl@mathboxdir@aux R}%
5840 \fi}
5841 \def\bbl@mathboxdir@aux#1{%
5842 \ifnextchar\egroup{}\{\textdir T#1T\relax}}
5843 \frozen@everymath\expandafter{%
5844 \expandafter\bbl@mathboxdir\the\frozen@everymath}
5845 \frozen@everydisplay\expandafter{%
5846 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5847 \fi

```

13.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5848 \bbl@trace{Redefinitions for bidi layout}
5849 \ifx\@eqnnum\undefined\else
5850 \ifx\bbl@attr@dir\undefined\else
5851 \edef\@eqnnum{%
5852 \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5853 \unexpanded\expandafter{\@eqnnum}}
5854 \fi
5855 \fi
5856 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5857 \ifnum\bbl@bidimode>\z@
5858 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5859 \bbl@exp{%
5860 \mathdir\the\bodydir
5861 #1% Once entered in math, set boxes to restore values
5862 \<ifmmode>%
5863 \everyvbox{%
5864 \the\everyvbox
5865 \bodydir\the\bodydir
5866 \mathdir\the\mathdir
5867 \everyhbox{\the\everyhbox}%
5868 \everyvbox{\the\everyvbox}}%
5869 \everyhbox{%
5870 \the\everyhbox
5871 \bodydir\the\bodydir
5872 \mathdir\the\mathdir
5873 \everyhbox{\the\everyhbox}%
5874 \everyvbox{\the\everyvbox}}%
5875 \<fi>}}%
5876 \def\@hangfrom#1{%
5877 \setbox\@tempboxa\hbox{{#1}}%
5878 \hangindent\wd\@tempboxa

```

```

5879 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5880 \shapemode\@ne
5881 \fi
5882 \noindent\box\@tempboxa}
5883 \fi
5884 \IfBabelLayout{tabular}
5885 {\let\bb1@OL@tabular\@tabular
5886 \bb1@replace\@tabular{$}\bb1@nextfake$}%
5887 \let\bb1@NL@tabular\@tabular
5888 \AtBeginDocument{%
5889 \ifx\bb1@NL@tabular\@tabular\else
5890 \bb1@replace\@tabular{$}\bb1@nextfake$}%
5891 \let\bb1@NL@tabular\@tabular
5892 \fi}}
5893 {}
5894 \IfBabelLayout{lists}
5895 {\let\bb1@OL@list\list
5896 \bb1@sreplace\list{\parshape}\bb1@listparshape}%
5897 \let\bb1@NL@list\list
5898 \def\bb1@listparshape#1#2#3{%
5899 \parshape #1 #2 #3 %
5900 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5901 \shapemode\tw@
5902 \fi}}
5903 {}
5904 \IfBabelLayout{graphics}
5905 {\let\bb1@pictresetdir\relax
5906 \def\bb1@pictsetdir#1{%
5907 \ifcase\bb1@thetextdir
5908 \let\bb1@pictresetdir\relax
5909 \else
5910 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5911 \or\textdir TLT
5912 \else\bodydir TLT \textdir TLT
5913 \fi
5914 % \text\par\dir required in pgf:
5915 \def\bb1@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5916 \fi}%
5917 \ifx\AddToHook\undefined\else
5918 \AddToHook{env/picture/begin}{\bb1@pictsetdir\tw@}%
5919 \directlua{
5920 Babel.get_picture_dir = true
5921 Babel.picture_has_bidi = 0
5922 function Babel.picture_dir (head)
5923 if not Babel.get_picture_dir then return head end
5924 for item in node.traverse(head) do
5925 if item.id == node.id'glyph' then
5926 local itemchar = item.char
5927 % TODO. Copy paste pattern from Babel.bidi (-r)
5928 local chardata = Babel.characters[itemchar]
5929 local dir = chardata and chardata.d or nil
5930 if not dir then
5931 for nn, et in ipairs(Babel.ranges) do
5932 if itemchar < et[1] then
5933 break
5934 elseif itemchar <= et[2] then
5935 dir = et[3]
5936 break
5937 end

```

```

5938         end
5939     end
5940     if dir and (dir == 'al' or dir == 'r') then
5941         Babel.picture_has_bidi = 1
5942     end
5943 end
5944 end
5945 return head
5946 end
5947 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5948     "Babel.picture_dir")
5949 }%
5950 \AtBeginDocument{%
5951     \long\def\put(#1,#2)#3{%
5952         \@killglue
5953         % Try:
5954         \ifx\bbl@pictresetdir\relax
5955             \def\bbl@tempc{0}%
5956         \else
5957             \directlua{
5958                 Babel.get_picture_dir = true
5959                 Babel.picture_has_bidi = 0
5960             }%
5961             \setbox\z@\hb@xt@\z@{%
5962                 \@defaultunitsset\@tempdimc{#1}\unitlength
5963                 \kern\@tempdimc
5964                 #3\hss}%
5965             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
5966         \fi
5967         % Do:
5968         \@defaultunitsset\@tempdimc{#2}\unitlength
5969         \raise\@tempdimc\hb@xt@\z@{%
5970             \@defaultunitsset\@tempdimc{#1}\unitlength
5971             \kern\@tempdimc
5972             {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
5973         \ignorespaces}%
5974         \MakeRobust\put}%
5975 \fi
5976 \AtBeginDocument
5977     {\ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
5978         \ifx\AddToHook\@undefined
5979             \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
5980             {\bbl@pictsetdir\z@\pgfpicturetrue}%
5981         \else
5982             \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
5983         \fi
5984         \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
5985         \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
5986     \fi
5987     \ifx\tikzpicture\@undefined\else
5988         \ifx\AddToHook\@undefined\else
5989             \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
5990         \fi
5991         \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5992         \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
5993     \fi
5994     \ifx\AddToHook\@undefined\else
5995         \ifx\tcolorbox\@undefined\else
5996             \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%

```



```

5997         \bbl@sreplace\tcb@savebox
5998         {\ignorespaces}{\ignorespaces\bbl@pictresetdir}%
5999         \fi
6000     \fi
6001 }}
6002 {}

```

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`.

```

6003 \IfBabelLayout{counters}%
6004 {\let\bbl@OL@@textsuperscript\textsuperscript
6005  \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6006  \let\bbl@latinarabic=\@arabic
6007  \let\bbl@OL@@arabic\@arabic
6008  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6009  \@ifpackagewith{babel}{bidi=default}%
6010  {\let\bbl@asciroman=\@roman
6011   \let\bbl@OL@@roman\@roman
6012   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6013   \let\bbl@asciiRoman=\@Roman
6014   \let\bbl@OL@@roman\@Roman
6015   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6016   \let\bbl@OL@labelenumii\labelenumii
6017   \def\labelenumii{}\theenumii{}%
6018   \let\bbl@OL@p@enumiii\p@enumiii
6019   \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
6020 <<Footnote changes>>
6021 \IfBabelLayout{footnotes}%
6022 {\let\bbl@OL@footnote\footnote
6023  \BabelFootnote\footnote\language\language{}{}%
6024  \BabelFootnote\localfootnote\language\language{}{}%
6025  \BabelFootnote\mainfootnote{}\}\}\}
6026 {}

```

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.

```

6027 \IfBabelLayout{extras}%
6028 {\let\bbl@OL@underline\underline
6029  \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake\$@@underline}%
6030  \let\bbl@OL@LaTeX2e\LaTeX2e
6031  \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6032   \if b\expandafter\car\@series\@nil\boldmath\fi
6033   \babelsublr{%
6034     \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6035 {}
6036 </luatex>

```

13.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into

account the capture position points to the next character. Here word_head points to the starting node of the text to be matched.

```
6037 (*transforms)
6038 Babel.linebreaking.replacements = {}
6039 Babel.linebreaking.replacements[0] = {} -- pre
6040 Babel.linebreaking.replacements[1] = {} -- post
6041
6042 -- Discretionaries contain strings as nodes
6043 function Babel.str_to_nodes(fn, matches, base)
6044   local n, head, last
6045   if fn == nil then return nil end
6046   for s in string.utfvalues(fn(matches)) do
6047     if base.id == 7 then
6048       base = base.replace
6049     end
6050     n = node.copy(base)
6051     n.char = s
6052     if not head then
6053       head = n
6054     else
6055       last.next = n
6056     end
6057     last = n
6058   end
6059   return head
6060 end
6061
6062 Babel.fetch_subtext = {}
6063
6064 Babel.ignore_pre_char = function(node)
6065   return (node.lang == Babel.nohyphenation)
6066 end
6067
6068 -- Merging both functions doesn't seem feasible, because there are too
6069 -- many differences.
6070 Babel.fetch_subtext[0] = function(head)
6071   local word_string = ''
6072   local word_nodes = {}
6073   local lang
6074   local item = head
6075   local inmath = false
6076
6077   while item do
6078     if item.id == 11 then
6079       inmath = (item.subtype == 0)
6080     end
6081
6082     if inmath then
6083       -- pass
6084     end
6085
6086     elseif item.id == 29 then
6087       local locale = node.get_attribute(item, Babel.attr_locale)
6088
6089       if lang == locale or lang == nil then
6090         lang = lang or locale
6091         if Babel.ignore_pre_char(item) then
6092           word_string = word_string .. Babel.us_char
```

```

6093         else
6094             word_string = word_string .. unicode.utf8.char(item.char)
6095         end
6096         word_nodes[#word_nodes+1] = item
6097     else
6098         break
6099     end
6100
6101     elseif item.id == 12 and item.subtype == 13 then
6102         word_string = word_string .. ' '
6103         word_nodes[#word_nodes+1] = item
6104
6105         -- Ignore leading unrecognized nodes, too.
6106         elseif word_string ~= '' then
6107             word_string = word_string .. Babel.us_char
6108             word_nodes[#word_nodes+1] = item -- Will be ignored
6109         end
6110
6111         item = item.next
6112     end
6113
6114     -- Here and above we remove some trailing chars but not the
6115     -- corresponding nodes. But they aren't accessed.
6116     if word_string:sub(-1) == ' ' then
6117         word_string = word_string:sub(1,-2)
6118     end
6119     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6120     return word_string, word_nodes, item, lang
6121 end
6122
6123 Babel.fetch_subtext[1] = function(head)
6124     local word_string = ''
6125     local word_nodes = {}
6126     local lang
6127     local item = head
6128     local inmath = false
6129
6130     while item do
6131
6132         if item.id == 11 then
6133             inmath = (item.subtype == 0)
6134         end
6135
6136         if inmath then
6137             -- pass
6138         end
6139
6140         elseif item.id == 29 then
6141             if item.lang == lang or lang == nil then
6142                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6143                     lang = lang or item.lang
6144                     word_string = word_string .. unicode.utf8.char(item.char)
6145                     word_nodes[#word_nodes+1] = item
6146                 end
6147             else
6148                 break
6149             end
6150
6151         elseif item.id == 7 and item.subtype == 2 then
6152             word_string = word_string .. '='

```

```

6152     word_nodes[#word_nodes+1] = item
6153
6154     elseif item.id == 7 and item.subtype == 3 then
6155         word_string = word_string .. '|'
6156         word_nodes[#word_nodes+1] = item
6157
6158     -- (1) Go to next word if nothing was found, and (2) implicitly
6159     -- remove leading USs.
6160     elseif word_string == '' then
6161         -- pass
6162
6163     -- This is the responsible for splitting by words.
6164     elseif (item.id == 12 and item.subtype == 13) then
6165         break
6166
6167     else
6168         word_string = word_string .. Babel.us_char
6169         word_nodes[#word_nodes+1] = item -- Will be ignored
6170     end
6171
6172     item = item.next
6173 end
6174
6175 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6176 return word_string, word_nodes, item, lang
6177 end
6178
6179 function Babel.pre_hyphenate_replace(head)
6180     Babel.hyphenate_replace(head, 0)
6181 end
6182
6183 function Babel.post_hyphenate_replace(head)
6184     Babel.hyphenate_replace(head, 1)
6185 end
6186
6187 Babel.us_char = string.char(31)
6188
6189 function Babel.hyphenate_replace(head, mode)
6190     local u = unicode.utf8
6191     local lbkr = Babel.linebreaking.replacements[mode]
6192
6193     local word_head = head
6194
6195     while true do -- for each subtext block
6196
6197         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6198
6199         if Babel.debug then
6200             print()
6201             print((mode == 0) and '@@@<' or '@@@>', w)
6202         end
6203
6204         if nw == nil and w == '' then break end
6205
6206         if not lang then goto next end
6207         if not lbkr[lang] then goto next end
6208
6209         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6210         -- loops are nested.

```

```

6211 for k=1, #lbr[lang] do
6212     local p = lbr[lang][k].pattern
6213     local r = lbr[lang][k].replace
6214
6215     if Babel.debug then
6216         print('*****', p, mode)
6217     end
6218
6219     -- This variable is set in some cases below to the first *byte*
6220     -- after the match, either as found by u.match (faster) or the
6221     -- computed position based on sc if w has changed.
6222     local last_match = 0
6223     local step = 0
6224
6225     -- For every match.
6226     while true do
6227         if Babel.debug then
6228             print('====')
6229         end
6230         local new -- used when inserting and removing nodes
6231
6232         local matches = { u.match(w, p, last_match) }
6233
6234         if #matches < 2 then break end
6235
6236         -- Get and remove empty captures (with ()'s, which return a
6237         -- number with the position), and keep actual captures
6238         -- (from (...)), if any, in matches.
6239         local first = table.remove(matches, 1)
6240         local last = table.remove(matches, #matches)
6241         -- Non re-fetched substrings may contain \31, which separates
6242         -- subsubstrings.
6243         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6244
6245         local save_last = last -- with A()BC()D, points to D
6246
6247         -- Fix offsets, from bytes to unicode. Explained above.
6248         first = u.len(w:sub(1, first-1)) + 1
6249         last = u.len(w:sub(1, last-1)) -- now last points to C
6250
6251         -- This loop stores in n small table the nodes
6252         -- corresponding to the pattern. Used by 'data' to provide a
6253         -- predictable behavior with 'insert' (now w_nodes is modified on
6254         -- the fly), and also access to 'remove'd nodes.
6255         local sc = first-1 -- Used below, too
6256         local data_nodes = {}
6257
6258         for q = 1, last-first+1 do
6259             data_nodes[q] = w_nodes[sc+q]
6260         end
6261
6262         -- This loop traverses the matched substring and takes the
6263         -- corresponding action stored in the replacement list.
6264         -- sc = the position in substr nodes / string
6265         -- rc = the replacement table index
6266         local rc = 0
6267
6268         while rc < last-first+1 do -- for each replacement
6269             if Babel.debug then

```

```

6270         print('.....', rc + 1)
6271     end
6272     sc = sc + 1
6273     rc = rc + 1
6274
6275     if Babel.debug then
6276         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6277         local ss = ''
6278         for itt in node.traverse(head) do
6279             if itt.id == 29 then
6280                 ss = ss .. unicode.utf8.char(itt.char)
6281             else
6282                 ss = ss .. '{' .. itt.id .. '}'
6283             end
6284         end
6285         print('*****', ss)
6286     end
6287
6288     local crep = r[rc]
6289     local item = w_nodes[sc]
6290     local item_base = item
6291     local placeholder = Babel.us_char
6292     local d
6293
6294     if crep and crep.data then
6295         item_base = data_nodes[crep.data]
6296     end
6297
6298     if crep then
6299         step = crep.step or 0
6300     end
6301
6302     if crep and next(crep) == nil then -- = {}
6303         last_match = save_last -- Optimization
6304         goto next
6305     end
6306
6307     elseif crep == nil or crep.remove then
6308         node.remove(head, item)
6309         table.remove(w_nodes, sc)
6310         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6311         sc = sc - 1 -- Nothing has been inserted.
6312         last_match = utf8.offset(w, sc+1+step)
6313         goto next
6314     end
6315     elseif crep and crep.kashida then -- Experimental
6316         node.set_attribute(item,
6317             Babel.attr_kashida,
6318             crep.kashida)
6319         last_match = utf8.offset(w, sc+1+step)
6320         goto next
6321     end
6322     elseif crep and crep.string then
6323         local str = crep.string(matches)
6324         if str == '' then -- Gather with nil
6325             node.remove(head, item)
6326             table.remove(w_nodes, sc)
6327             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6328             sc = sc - 1 -- Nothing has been inserted.

```

```

6329     else
6330         local loop_first = true
6331         for s in string.utfvalues(str) do
6332             d = node.copy(item_base)
6333             d.char = s
6334             if loop_first then
6335                 loop_first = false
6336                 head, new = node.insert_before(head, item, d)
6337                 if sc == 1 then
6338                     word_head = head
6339                 end
6340                 w_nodes[sc] = d
6341                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6342             else
6343                 sc = sc + 1
6344                 head, new = node.insert_before(head, item, d)
6345                 table.insert(w_nodes, sc, new)
6346                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6347             end
6348             if Babel.debug then
6349                 print('.....', 'str')
6350                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6351             end
6352             end -- for
6353             node.remove(head, item)
6354         end -- if ''
6355         last_match = utf8.offset(w, sc+1+step)
6356         goto next
6357
6358     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6359         d = node.new(7, 0) -- (disc, discretionary)
6360         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6361         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6362         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6363         d.attr = item_base.attr
6364         if crep.pre == nil then -- TeXbook p96
6365             d.penalty = crep.penalty or tex.hyphenpenalty
6366         else
6367             d.penalty = crep.penalty or tex.exhyphenpenalty
6368         end
6369         placeholder = '|'
6370         head, new = node.insert_before(head, item, d)
6371
6372     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6373         -- ERROR
6374
6375     elseif crep and crep.penalty then
6376         d = node.new(14, 0) -- (penalty, userpenalty)
6377         d.attr = item_base.attr
6378         d.penalty = crep.penalty
6379         head, new = node.insert_before(head, item, d)
6380
6381     elseif crep and crep.space then
6382         -- 655360 = 10 pt = 10 * 65536 sp
6383         d = node.new(12, 13) -- (glue, spaceskip)
6384         local quad = font.getfont(item_base.font).size or 655360
6385         node.setglue(d, crep.space[1] * quad,
6386                     crep.space[2] * quad,
6387                     crep.space[3] * quad)

```

```

6388         if mode == 0 then
6389             placeholder = ' '
6390         end
6391         head, new = node.insert_before(head, item, d)
6392
6393     elseif crep and crep.spacefactor then
6394         d = node.new(12, 13)      -- (glue, spaceskip)
6395         local base_font = font.getfont(item_base.font)
6396         node.setglue(d,
6397             crep.spacefactor[1] * base_font.parameters['space'],
6398             crep.spacefactor[2] * base_font.parameters['space_stretch'],
6399             crep.spacefactor[3] * base_font.parameters['space_shrink'])
6400         if mode == 0 then
6401             placeholder = ' '
6402         end
6403         head, new = node.insert_before(head, item, d)
6404
6405     elseif mode == 0 and crep and crep.space then
6406         -- ERROR
6407
6408     end -- ie replacement cases
6409
6410     -- Shared by disc, space and penalty.
6411     if sc == 1 then
6412         word_head = head
6413     end
6414     if crep.insert then
6415         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6416         table.insert(w_nodes, sc, new)
6417         last = last + 1
6418     else
6419         w_nodes[sc] = d
6420         node.remove(head, item)
6421         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6422     end
6423
6424     last_match = utf8.offset(w, sc+1+step)
6425
6426     ::next::
6427
6428 end -- for each replacement
6429
6430 if Babel.debug then
6431     print('.....', '/')
6432     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6433 end
6434
6435 end -- for match
6436
6437 end -- for patterns
6438
6439 ::next::
6440 word_head = nw
6441 end -- for substring
6442 return head
6443 end
6444
6445 -- This table stores capture maps, numbered consecutively
6446 Babel.capture_maps = {}

```



```

6447
6448 -- The following functions belong to the next macro
6449 function Babel.capture_func(key, cap)
6450   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6451   local cnt
6452   local u = unicode.utf8
6453   ret, cnt = ret:gsub('{{[0-9]}|([^\]|+)|(-)}', Babel.capture_func_map)
6454   if cnt == 0 then
6455     ret = u.gsub(ret, '{{(%%x%%x%%x+}}',
6456                 function (n)
6457                   return u.char(tonumber(n, 16))
6458                 end)
6459   end
6460   ret = ret:gsub("%[%[%]]%.", '')
6461   ret = ret:gsub("%.%[%[%]]%", '')
6462   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6463 end
6464
6465 function Babel.capt_map(from, mapno)
6466   return Babel.capture_maps[mapno][from] or from
6467 end
6468
6469 -- Handle the {n|abc|ABC} syntax in captures
6470 function Babel.capture_func_map(capno, from, to)
6471   local u = unicode.utf8
6472   from = u.gsub(from, '{{(%%x%%x%%x+}}',
6473               function (n)
6474                 return u.char(tonumber(n, 16))
6475               end)
6476   to = u.gsub(to, '{{(%%x%%x%%x+}}',
6477              function (n)
6478                return u.char(tonumber(n, 16))
6479              end)
6480   local froms = {}
6481   for s in string.utfcharacters(from) do
6482     table.insert(froms, s)
6483   end
6484   local cnt = 1
6485   table.insert(Babel.capture_maps, {})
6486   local mlen = table.getn(Babel.capture_maps)
6487   for s in string.utfcharacters(to) do
6488     Babel.capture_maps[mlen][froms[cnt]] = s
6489     cnt = cnt + 1
6490   end
6491   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6492         (mlen) .. ").." .. "["
6493 end
6494
6495 -- Create/Extend reversed sorted list of kashida weights:
6496 function Babel.capture_kashida(key, wt)
6497   wt = tonumber(wt)
6498   if Babel.kashida_wts then
6499     for p, q in ipairs(Babel.kashida_wts) do
6500       if wt == q then
6501         break
6502       elseif wt > q then
6503         table.insert(Babel.kashida_wts, p, wt)
6504         break
6505       elseif table.getn(Babel.kashida_wts) == p then

```

```

6506         table.insert(Babel.kashida_wts, wt)
6507     end
6508 end
6509 else
6510     Babel.kashida_wts = { wt }
6511 end
6512 return 'kashida = ' .. wt
6513 end
6514 </transforms>

```

13.12 Lua: Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```

6515 (*basic-r)
6516 Babel = Babel or {}
6517
6518 Babel.bidi_enabled = true
6519
6520 require('babel-data-bidi.lua')
6521
6522 local characters = Babel.characters
6523 local ranges = Babel.ranges
6524

```

```

6525 local DIR = node.id("dir")
6526
6527 local function dir_mark(head, from, to, outer)
6528   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6529   local d = node.new(DIR)
6530   d.dir = '+' .. dir
6531   node.insert_before(head, from, d)
6532   d = node.new(DIR)
6533   d.dir = '-' .. dir
6534   node.insert_after(head, to, d)
6535 end
6536
6537 function Babel.bidi(head, ispar)
6538   local first_n, last_n          -- first and last char with nums
6539   local last_es                  -- an auxiliary 'last' used with nums
6540   local first_d, last_d          -- first and last char in L/R block
6541   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):

```

6542   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6543   local strong_lr = (strong == 'l') and 'l' or 'r'
6544   local outer = strong
6545
6546   local new_dir = false
6547   local first_dir = false
6548   local inmath = false
6549
6550   local last_lr
6551
6552   local type_n = ''
6553
6554   for item in node.traverse(head) do
6555
6556     -- three cases: glyph, dir, otherwise
6557     if item.id == node.id'glyph'
6558       or (item.id == 7 and item.subtype == 2) then
6559
6560       local itemchar
6561       if item.id == 7 and item.subtype == 2 then
6562         itemchar = item.replace.char
6563       else
6564         itemchar = item.char
6565       end
6566       local chardata = characters[itemchar]
6567       dir = chardata and chardata.d or nil
6568       if not dir then
6569         for nn, et in ipairs(ranges) do
6570           if itemchar < et[1] then
6571             break
6572           elseif itemchar <= et[2] then
6573             dir = et[3]
6574             break
6575           end
6576         end
6577       end
6578       dir = dir or 'l'
6579       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).

```

6580     if new_dir then
6581         attr_dir = 0
6582         for at in node.traverse(item.attr) do
6583             if at.number == Babel.attr_dir then
6584                 attr_dir = at.value % 3
6585             end
6586         end
6587         if attr_dir == 1 then
6588             strong = 'r'
6589         elseif attr_dir == 2 then
6590             strong = 'al'
6591         else
6592             strong = 'l'
6593         end
6594         strong_lr = (strong == 'l') and 'l' or 'r'
6595         outer = strong_lr
6596         new_dir = false
6597     end
6598
6599     if dir == 'nsm' then dir = strong end          -- W1

```

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

```

6600     dir_real = dir          -- We need dir_real to set strong below
6601     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:

```

6602     if strong == 'al' then
6603         if dir == 'en' then dir = 'an' end          -- W2
6604         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6605         strong_lr = 'r'          -- W3
6606     end

```

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

```

6607     elseif item.id == node.id'dir' and not inmath then
6608         new_dir = true
6609         dir = nil
6610     elseif item.id == node.id'math' then
6611         inmath = (item.subtype == 0)
6612     else
6613         dir = nil          -- Not a char
6614     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>.

```

6615     if dir == 'en' or dir == 'an' or dir == 'et' then
6616         if dir ~= 'et' then
6617             type_n = dir
6618         end
6619         first_n = first_n or item
6620         last_n = last_es or item

```

```

6621     last_es = nil
6622   elseif dir == 'es' and last_n then -- W3+W6
6623     last_es = item
6624   elseif dir == 'cs' then           -- it's right - do nothing
6625   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6626     if strong_lr == 'r' and type_n ~= '' then
6627       dir_mark(head, first_n, last_n, 'r')
6628     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6629       dir_mark(head, first_n, last_n, 'r')
6630       dir_mark(head, first_d, last_d, outer)
6631       first_d, last_d = nil, nil
6632     elseif strong_lr == 'l' and type_n ~= '' then
6633       last_d = last_n
6634     end
6635     type_n = ''
6636     first_n, last_n = nil, nil
6637   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:

```

6638   if dir == 'l' or dir == 'r' then
6639     if dir ~= outer then
6640       first_d = first_d or item
6641       last_d = item
6642     elseif first_d and dir ~= strong_lr then
6643       dir_mark(head, first_d, last_d, outer)
6644       first_d, last_d = nil, nil
6645     end
6646   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.

```

6647   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6648     item.char = characters[item.char] and
6649       characters[item.char].m or item.char
6650   elseif (dir or new_dir) and last_lr ~= item then
6651     local mir = outer .. strong_lr .. (dir or outer)
6652     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6653       for ch in node.traverse(node.next(last_lr)) do
6654         if ch == item then break end
6655         if ch.id == node.id'glyph' and characters[ch.char] then
6656           ch.char = characters[ch.char].m or ch.char
6657         end
6658       end
6659     end
6660   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).

```

6661   if dir == 'l' or dir == 'r' then
6662     last_lr = item
6663     strong = dir_real           -- Don't search back - best save now
6664     strong_lr = (strong == 'l') and 'l' or 'r'
6665   elseif new_dir then

```

```

6666     last_lr = nil
6667   end
6668 end

```

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

```

6669 if last_lr and outer == 'r' then
6670   for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6671     if characters[ch.char] then
6672       ch.char = characters[ch.char].m or ch.char
6673     end
6674   end
6675 end
6676 if first_n then
6677   dir_mark(head, first_n, last_n, outer)
6678 end
6679 if first_d then
6680   dir_mark(head, first_d, last_d, outer)
6681 end

```

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

```

6682 return node.prev(head) or head
6683 end
6684 </basic-r>

```

And here the Lua code for bidi=basic:

```

6685 <(*basic)
6686 Babel = Babel or {}
6687
6688 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6689
6690 Babel.fontmap = Babel.fontmap or {}
6691 Babel.fontmap[0] = {}      -- l
6692 Babel.fontmap[1] = {}      -- r
6693 Babel.fontmap[2] = {}      -- al/an
6694
6695 Babel.bidi_enabled = true
6696 Babel.mirroring_enabled = true
6697
6698 require('babel-data-bidi.lua')
6699
6700 local characters = Babel.characters
6701 local ranges = Babel.ranges
6702
6703 local DIR = node.id('dir')
6704 local GLYPH = node.id('glyph')
6705
6706 local function insert_implicit(head, state, outer)
6707   local new_state = state
6708   if state.sim and state.eim and state.sim ~= state.eim then
6709     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6710     local d = node.new(DIR)
6711     d.dir = '+' .. dir
6712     node.insert_before(head, state.sim, d)
6713     local d = node.new(DIR)
6714     d.dir = '-' .. dir
6715     node.insert_after(head, state.eim, d)
6716   end
6717   new_state.sim, new_state.eim = nil, nil

```

```

6718 return head, new_state
6719 end
6720
6721 local function insert_numeric(head, state)
6722     local new
6723     local new_state = state
6724     if state.san and state.ean and state.san ~= state.ean then
6725         local d = node.new(DIR)
6726         d.dir = '+TLT'
6727         _, new = node.insert_before(head, state.san, d)
6728         if state.san == state.sim then state.sim = new end
6729         local d = node.new(DIR)
6730         d.dir = '-TLT'
6731         _, new = node.insert_after(head, state.ean, d)
6732         if state.ean == state.eim then state.eim = new end
6733     end
6734     new_state.san, new_state.ean = nil, nil
6735     return head, new_state
6736 end
6737
6738 -- TODO - \hbox with an explicit dir can lead to wrong results
6739 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6740 -- was s made to improve the situation, but the problem is the 3-dir
6741 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6742 -- well.
6743
6744 function Babel.bidi(head, ispar, hdir)
6745     local d -- d is used mainly for computations in a loop
6746     local prev_d = ''
6747     local new_d = false
6748
6749     local nodes = {}
6750     local outer_first = nil
6751     local inmath = false
6752
6753     local glue_d = nil
6754     local glue_i = nil
6755
6756     local has_en = false
6757     local first_et = nil
6758
6759     local ATDIR = Babel.attr_dir
6760
6761     local save_outer
6762     local temp = node.get_attribute(head, ATDIR)
6763     if temp then
6764         temp = temp % 3
6765         save_outer = (temp == 0 and 'l') or
6766                     (temp == 1 and 'r') or
6767                     (temp == 2 and 'al')
6768     elseif ispar then -- Or error? Shouldn't happen
6769         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6770     else -- Or error? Shouldn't happen
6771         save_outer = ('TRT' == hdir) and 'r' or 'l'
6772     end
6773     -- when the callback is called, we are just _after_ the box,
6774     -- and the textdir is that of the surrounding text
6775     -- if not ispar and hdir ~= tex.textdir then
6776     --     save_outer = ('TRT' == hdir) and 'r' or 'l'

```

```

6777 -- end
6778 local outer = save_outer
6779 local last = outer
6780 -- 'al' is only taken into account in the first, current loop
6781 if save_outer == 'al' then save_outer = 'r' end
6782
6783 local fontmap = Babel.fontmap
6784
6785 for item in node.traverse(head) do
6786
6787     -- In what follows, #node is the last (previous) node, because the
6788     -- current one is not added until we start processing the neutrals.
6789
6790     -- three cases: glyph, dir, otherwise
6791     if item.id == GLYPH
6792         or (item.id == 7 and item.subtype == 2) then
6793
6794         local d_font = nil
6795         local item_r
6796         if item.id == 7 and item.subtype == 2 then
6797             item_r = item.replace -- automatic discs have just 1 glyph
6798         else
6799             item_r = item
6800         end
6801         local chardata = characters[item_r.char]
6802         d = chardata and chardata.d or nil
6803         if not d or d == 'nsm' then
6804             for nn, et in ipairs(ranges) do
6805                 if item_r.char < et[1] then
6806                     break
6807                 elseif item_r.char <= et[2] then
6808                     if not d then d = et[3]
6809                     elseif d == 'nsm' then d_font = et[3]
6810                     end
6811                     break
6812                 end
6813             end
6814         end
6815         d = d or 'l'
6816
6817         -- A short 'pause' in bidi for mapfont
6818         d_font = d_font or d
6819         d_font = (d_font == 'l' and 0) or
6820             (d_font == 'nsm' and 0) or
6821             (d_font == 'r' and 1) or
6822             (d_font == 'al' and 2) or
6823             (d_font == 'an' and 2) or nil
6824         if d_font and fontmap and fontmap[d_font][item_r.font] then
6825             item_r.font = fontmap[d_font][item_r.font]
6826         end
6827
6828         if new_d then
6829             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6830             if inmath then
6831                 attr_d = 0
6832             else
6833                 attr_d = node.get_attribute(item, ATDIR)
6834                 attr_d = attr_d % 3
6835             end

```



```

6836         if attr_d == 1 then
6837             outer_first = 'r'
6838             last = 'r'
6839         elseif attr_d == 2 then
6840             outer_first = 'r'
6841             last = 'al'
6842         else
6843             outer_first = 'l'
6844             last = 'l'
6845         end
6846         outer = last
6847         has_en = false
6848         first_et = nil
6849         new_d = false
6850     end
6851
6852     if glue_d then
6853         if (d == 'l' and 'l' or 'r') ~= glue_d then
6854             table.insert(nodes, {glue_i, 'on', nil})
6855         end
6856         glue_d = nil
6857         glue_i = nil
6858     end
6859
6860     elseif item.id == DIR then
6861         d = nil
6862         new_d = true
6863
6864     elseif item.id == node.id'glue' and item.subtype == 13 then
6865         glue_d = d
6866         glue_i = item
6867         d = nil
6868
6869     elseif item.id == node.id'math' then
6870         inmath = (item.subtype == 0)
6871
6872     else
6873         d = nil
6874     end
6875
6876     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6877     if last == 'al' and d == 'en' then
6878         d = 'an'          -- W3
6879     elseif last == 'al' and (d == 'et' or d == 'es') then
6880         d = 'on'          -- W6
6881     end
6882
6883     -- EN + CS/ES + EN      -- W4
6884     if d == 'en' and #nodes >= 2 then
6885         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6886             and nodes[#nodes-1][2] == 'en' then
6887             nodes[#nodes][2] = 'en'
6888         end
6889     end
6890
6891     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6892     if d == 'an' and #nodes >= 2 then
6893         if (nodes[#nodes][2] == 'cs')
6894             and nodes[#nodes-1][2] == 'an' then

```

```

6895         nodes[#nodes][2] = 'an'
6896     end
6897 end
6898
6899 -- ET/EN          -- W5 + W7->l / W6->on
6900 if d == 'et' then
6901     first_et = first_et or (#nodes + 1)
6902 elseif d == 'en' then
6903     has_en = true
6904     first_et = first_et or (#nodes + 1)
6905 elseif first_et then -- d may be nil here !
6906     if has_en then
6907         if last == 'l' then
6908             temp = 'l' -- W7
6909         else
6910             temp = 'en' -- W5
6911         end
6912     else
6913         temp = 'on' -- W6
6914     end
6915     for e = first_et, #nodes do
6916         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6917     end
6918     first_et = nil
6919     has_en = false
6920 end
6921
6922 -- Force mathdir in math if ON (currently works as expected only
6923 -- with 'l')
6924 if inmath and d == 'on' then
6925     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6926 end
6927
6928 if d then
6929     if d == 'al' then
6930         d = 'r'
6931         last = 'al'
6932     elseif d == 'l' or d == 'r' then
6933         last = d
6934     end
6935     prev_d = d
6936     table.insert(nodes, {item, d, outer_first})
6937 end
6938
6939 outer_first = nil
6940
6941 end
6942
6943 -- TODO -- repeated here in case EN/ET is the last node. Find a
6944 -- better way of doing things:
6945 if first_et then -- dir may be nil here !
6946     if has_en then
6947         if last == 'l' then
6948             temp = 'l' -- W7
6949         else
6950             temp = 'en' -- W5
6951         end
6952     else
6953         temp = 'on' -- W6

```

```

6954     end
6955     for e = first_et, #nodes do
6956         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6957     end
6958 end
6959
6960 -- dummy node, to close things
6961 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6962
6963 ----- NEUTRAL -----
6964
6965 outer = save_outer
6966 last = outer
6967
6968 local first_on = nil
6969
6970 for q = 1, #nodes do
6971     local item
6972
6973     local outer_first = nodes[q][3]
6974     outer = outer_first or outer
6975     last = outer_first or last
6976
6977     local d = nodes[q][2]
6978     if d == 'an' or d == 'en' then d = 'r' end
6979     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6980
6981     if d == 'on' then
6982         first_on = first_on or q
6983     elseif first_on then
6984         if last == d then
6985             temp = d
6986         else
6987             temp = outer
6988         end
6989         for r = first_on, q - 1 do
6990             nodes[r][2] = temp
6991             item = nodes[r][1] -- MIRRORING
6992             if Babel.mirroring_enabled and item.id == GLYPH
6993                 and temp == 'r' and characters[item.char] then
6994                 local font_mode = font.fonts[item.font].properties.mode
6995                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6996                     item.char = characters[item.char].m or item.char
6997                 end
6998             end
6999         end
7000         first_on = nil
7001     end
7002
7003     if d == 'r' or d == 'l' then last = d end
7004 end
7005
7006 ----- IMPLICIT, REORDER -----
7007
7008 outer = save_outer
7009 last = outer
7010
7011 local state = {}
7012 state.has_r = false

```

```

7013
7014 for q = 1, #nodes do
7015
7016     local item = nodes[q][1]
7017
7018     outer = nodes[q][3] or outer
7019
7020     local d = nodes[q][2]
7021
7022     if d == 'nsm' then d = last end          -- W1
7023     if d == 'en' then d = 'an' end
7024     local isdir = (d == 'r' or d == 'l')
7025
7026     if outer == 'l' and d == 'an' then
7027         state.san = state.san or item
7028         state.ean = item
7029     elseif state.san then
7030         head, state = insert_numeric(head, state)
7031     end
7032
7033     if outer == 'l' then
7034         if d == 'an' or d == 'r' then      -- im -> implicit
7035             if d == 'r' then state.has_r = true end
7036             state.sim = state.sim or item
7037             state.eim = item
7038         elseif d == 'l' and state.sim and state.has_r then
7039             head, state = insert_implicit(head, state, outer)
7040         elseif d == 'l' then
7041             state.sim, state.eim, state.has_r = nil, nil, false
7042         end
7043     else
7044         if d == 'an' or d == 'l' then
7045             if nodes[q][3] then -- nil except after an explicit dir
7046                 state.sim = item -- so we move sim 'inside' the group
7047             else
7048                 state.sim = state.sim or item
7049             end
7050             state.eim = item
7051         elseif d == 'r' and state.sim then
7052             head, state = insert_implicit(head, state, outer)
7053         elseif d == 'r' then
7054             state.sim, state.eim = nil, nil
7055         end
7056     end
7057
7058     if isdir then
7059         last = d          -- Don't search back - best save now
7060     elseif d == 'on' and state.san then
7061         state.san = state.san or item
7062         state.ean = item
7063     end
7064
7065 end
7066
7067 return node.prev(head) or head
7068 end
7069 (/basic)

```

14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

For the meaning of these codes, see the Unicode standard.

15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
7070 ⟨*nil⟩
7071 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7072 \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.

```
7073 \ifx\l@nil\undefined
7074   \newlanguage\l@nil
7075   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7076   \let\bbl@elt\relax
7077   \edef\bbl@languages{% Add it to the list of languages
7078     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7079 \fi
```

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

```
7080 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7081 \let\captionnil\@empty
7082 \let\datenil\@empty
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
7083 \ldf@finish{nil}
7084 ⟨/nil⟩
```

16 Support for Plain T_EX (plain.def)

16.1 Not renaming hyphen.tex

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

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).

As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

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).

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.

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

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.

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).

213

```

7109 \closein0
7110 \else
7111 \closein0
7112 {\immediate\write16{*****}%
7113 \immediate\write16{* Local config file #1.cfg used}%
7114 \immediate\write16{*}%
7115 }
7116 \input #1.cfg\relax
7117 \fi
7118 \@endoflfd}

```

16.3 General tools

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

```

7119 \long\def\@firstofone#1{#1}
7120 \long\def\@firstoftwo#1#2{#1}
7121 \long\def\@secondoftwo#1#2{#2}
7122 \def\@nnil{\@nil}
7123 \def\@gobbletwo#1#2{}
7124 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7125 \def\@star@or@long#1{%
7126 \@ifstar
7127 {\let\l@ngrel@x\relax#1}%
7128 {\let\l@ngrel@x\long#1}}
7129 \let\l@ngrel@x\relax
7130 \def\@car#1#2\@nil{#1}
7131 \def\@cdr#1#2\@nil{#2}
7132 \let\@typeset@protect\relax
7133 \let\protected@edef\edef
7134 \long\def\@gobble#1{}
7135 \edef\@backslashchar{\expandafter\@gobble\string\}
7136 \def\strip@prefix#1>{}
7137 \def\g@addto@macro#1#2{%
7138 \toks@\expandafter{#1#2}%
7139 \xdef#1{\the\toks@}}
7140 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7141 \def\@nameuse#1{\csname #1\endcsname}
7142 \def\@ifundefined#1{%
7143 \expandafter\ifx\csname#1\endcsname\relax
7144 \expandafter\@firstoftwo
7145 \else
7146 \expandafter\@secondoftwo
7147 \fi}
7148 \def\@expandtwoargs#1#2#3{%
7149 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7150 \def\zap@space#1 #2{%
7151 #1%
7152 \ifx#2\@empty\else\expandafter\zap@space\fi
7153 #2}
7154 \let\bbl@trace\@gobble
7155 \def\bbl@error#1#2{%
7156 \begingroup
7157 \newlinechar=^^J
7158 \def\{^^J(babel) }%
7159 \errhelp{#2}\errmessage{\{#1}%
7160 \endgroup}
7161 \def\bbl@warning#1{%
7162 \begingroup

```

```

7163 \newlinechar=`^^J
7164 \def\{^^J(babel) }%
7165 \message{\#1}%
7166 \endgroup}
7167 \let\bbl@infowarn\bbl@warning
7168 \def\bbl@info#1{%
7169 \begingroup
7170 \newlinechar=`^^J
7171 \def\{^^J}%
7172 \wlog{#1}%
7173 \endgroup}

```

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

```

7174 \ifx\@preamblecmds\@undefined
7175 \def\@preamblecmds{}
7176 \fi
7177 \def\@onlypreamble#1{%
7178 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7179 \@preamblecmds\do#1}}
7180 \@onlypreamble\@onlypreamble

```

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

```

7181 \def\begindocument{%
7182 \@begindocumenthook
7183 \global\let\@begindocumenthook\@undefined
7184 \def\do##1{\global\let##1\@undefined}%
7185 \@preamblecmds
7186 \global\let\do\noexpand}
7187 \ifx\@begindocumenthook\@undefined
7188 \def\@begindocumenthook{}
7189 \fi
7190 \@onlypreamble\@begindocumenthook
7191 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

7192 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7193 \@onlypreamble\AtEndOfPackage
7194 \def\@endofldf{}
7195 \@onlypreamble\@endofldf
7196 \let\bbl@afterlang\@empty
7197 \chardef\bbl@opt@hyphenmap\z@

```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

7198 \catcode`\&=\z@
7199 \ifx&\if@files\@undefined
7200 \expandafter\let\csname if@files\expandafter\endcsname
7201 \csname iffalse\endcsname
7202 \fi
7203 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7204 \def\newcommand{\@star@or@long\new@command}
7205 \def\new@command#1{%
7206 \@testopt{\@newcommand#1}0}
7207 \def\@newcommand#1[#2]{%

```



```

7208 \@ifnextchar [{\@argdef#1[#2]]%
7209         {\@argdef#1[#2]}}
7210 \long\def\@argdef#1[#2]#3{%
7211     \@argdef#1\@ne{#2}{#3}}
7212 \long\def\@argdef#1[#2][#3]#4{%
7213     \expandafter\def\expandafter#1\expandafter{%
7214         \expandafter\@protected@testopt\expandafter #1%
7215         \csname\string#1\expandafter\endcsname{#3}}}%
7216 \expandafter\@argdef \csname\string#1\endcsname
7217 \tw@{#2}{#4}}
7218 \long\def\@argdef#1#2#3{%
7219     \@tempcnta#3\relax
7220     \advance \@tempcnta \@ne
7221     \let\@hash@\relax
7222     \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7223     \@tempcntb #2%
7224     \@whilenum\@tempcntb <\@tempcnta
7225     \do{%
7226         \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7227         \advance\@tempcntb \@ne}%
7228     \let\@hash@###
7229     \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
7230 \def\providecommand{\@star@or@long\provide@command}
7231 \def\provide@command#1{%
7232     \begingroup
7233     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7234     \endgroup
7235     \expandafter\ifundefined\@gtempa
7236         {\def\reserved@a{\new@command#1}}%
7237         {\let\reserved@a\relax
7238         \def\reserved@a{\new@command\reserved@a}}%
7239     \reserved@a}%
7240 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7241 \def\declare@robustcommand#1{%
7242     \edef\reserved@a{\string#1}%
7243     \def\reserved@b{#1}%
7244     \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7245     \edef#1{%
7246         \ifx\reserved@a\reserved@b
7247             \noexpand\x@protect
7248             \noexpand#1%
7249         \fi
7250         \noexpand\protect
7251         \expandafter\noexpand\csname
7252             \expandafter\@gobble\string#1 \endcsname
7253     }%
7254     \expandafter\new@command\csname
7255         \expandafter\@gobble\string#1 \endcsname
7256 }
7257 \def\x@protect#1{%
7258     \ifx\protect\@typeset@protect\else
7259         \@x@protect#1%
7260     \fi
7261 }
7262 \catcode`\&=\z@ % Trick to hide conditionals
7263 \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`.

```

7264 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7265 \catcode`\&=4
7266 \ifx\in@\@undefined
7267 \def\in@#1#2{%
7268   \def\in@@##1##2##3\in@{%
7269     \ifx\in@@##2\in@false\else\in@true\fi}%
7270   \in@@#2#1\in@\in@@}
7271 \else
7272 \let\bbl@tempa\@empty
7273 \fi
7274 \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).

```

7275 \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.

```

7276 \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_ϵ versions; just enough to make things work in plain \TeX environments.

```

7277 \ifx\@tempcnta\@undefined
7278   \csname newcount\endcsname\@tempcnta\relax
7279 \fi
7280 \ifx\@tempcntb\@undefined
7281   \csname newcount\endcsname\@tempcntb\relax
7282 \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`).

```

7283 \ifx\bye\@undefined
7284   \advance\count10 by -2\relax
7285 \fi
7286 \ifx\@ifnextchar\@undefined
7287   \def\@ifnextchar#1#2#3{%
7288     \let\reserved@d=#1%
7289     \def\reserved@a{#2}\def\reserved@b{#3}%
7290     \futurelet\@let@token\@ifnch}
7291 \def\@ifnch{%
7292   \ifx\@let@token\@sptoken
7293     \let\reserved@c\@xifnch
7294   \else
7295     \ifx\@let@token\reserved@d
7296       \let\reserved@c\reserved@a
7297     \else
7298       \let\reserved@c\reserved@b
7299     \fi
7300   \fi
7301   \reserved@c}
7302 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
7303 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7304 \fi
7305 \def\@testopt#1#2{%

```

```

7306 \@ifnextchar[{#1}{#1[#2]}}
7307 \def\@protected@testopt#1{%
7308   \ifx\protect\@typeset@protect
7309     \expandafter\@testopt
7310   \else
7311     \@x@protect#1%
7312   \fi}
7313 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7314   #2\relax}\fi}
7315 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7316   \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

7317 \def\DeclareTextCommand{%
7318   \@dec@text@cmd\providecommand
7319 }
7320 \def\ProvideTextCommand{%
7321   \@dec@text@cmd\providecommand
7322 }
7323 \def\DeclareTextSymbol#1#2#3{%
7324   \@dec@text@cmd\chardef#1{#2}#3\relax
7325 }
7326 \def\@dec@text@cmd#1#2#3{%
7327   \expandafter\def\expandafter#2%
7328     \expandafter{%
7329       \csname#3-cmd\expandafter\endcsname
7330       \expandafter#2%
7331       \csname#3\string#2\endcsname
7332     }%
7333 %   \let\@ifdefinable\@rc@ifdefinable
7334   \expandafter#1\csname#3\string#2\endcsname
7335 }
7336 \def\@current@cmd#1{%
7337   \ifx\protect\@typeset@protect\else
7338     \noexpand#1\expandafter\@gobble
7339   \fi
7340 }
7341 \def\@changed@cmd#1#2{%
7342   \ifx\protect\@typeset@protect
7343     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7344       \expandafter\ifx\csname ?\string#1\endcsname\relax
7345         \expandafter\def\csname ?\string#1\endcsname{%
7346           \@changed@x@err{#1}%
7347         }%
7348       \fi
7349       \global\expandafter\let
7350         \csname\cf@encoding\string#1\expandafter\endcsname
7351         \csname ?\string#1\endcsname
7352     \fi
7353     \csname\cf@encoding\string#1%
7354       \expandafter\endcsname
7355   \else
7356     \noexpand#1%
7357   \fi
7358 }
7359 \def\@changed@x@err#1{%

```

```

7360 \errhelp{Your command will be ignored, type <return> to proceed}%
7361 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7362 \def\DeclareTextCommandDefault#1{%
7363 \DeclareTextCommand#1?%
7364 }
7365 \def\ProvideTextCommandDefault#1{%
7366 \ProvideTextCommand#1?%
7367 }
7368 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7369 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7370 \def\DeclareTextAccent#1#2#3{%
7371 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7372 }
7373 \def\DeclareTextCompositeCommand#1#2#3#4{%
7374 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7375 \edef\reserved@b{\string##1}%
7376 \edef\reserved@c{%
7377 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7378 \ifx\reserved@b\reserved@c
7379 \expandafter\expandafter\expandafter\ifx
7380 \expandafter\@car\reserved@a\relax\relax\@nil
7381 \@text@composite
7382 \else
7383 \edef\reserved@b##1{%
7384 \def\expandafter\noexpand
7385 \csname#2\string#1\endcsname####1{%
7386 \noexpand\@text@composite
7387 \expandafter\noexpand\csname#2\string#1\endcsname
7388 ####1\noexpand\@empty\noexpand\@text@composite
7389 {##1}%
7390 }%
7391 }%
7392 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7393 \fi
7394 \expandafter\def\csname\expandafter\string\csname
7395 #2\endcsname\string#1-\string#3\endcsname{#4}
7396 \else
7397 \errhelp{Your command will be ignored, type <return> to proceed}%
7398 \errmessage{\string\DeclareTextCompositeCommand\space used on
7399 inappropriate command \protect#1}
7400 \fi
7401 }
7402 \def\@text@composite#1#2#3\@text@composite{%
7403 \expandafter\@text@composite@x
7404 \csname\string#1-\string#2\endcsname
7405 }
7406 \def\@text@composite@x#1#2{%
7407 \ifx#1\relax
7408 #2%
7409 \else
7410 #1%
7411 \fi
7412 }
7413 %
7414 \def\@strip@args#1:#2-#3\@strip@args{#2}
7415 \def\DeclareTextComposite#1#2#3#4{%
7416 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7417 \bgroup
7418 \lcode`\@=#4%

```

```

7419     \lowercase{%
7420     \egroup
7421     \reserved@a @%
7422     }%
7423 }
7424 %
7425 \def\UseTextSymbol#1#2{#2}
7426 \def\UseTextAccent#1#2#3{}
7427 \def\@use@text@encoding#1{}
7428 \def\DeclareTextSymbolDefault#1#2{%
7429     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7430 }
7431 \def\DeclareTextAccentDefault#1#2{%
7432     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7433 }
7434 \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.

```

7435 \DeclareTextAccent{"}{OT1}{127}
7436 \DeclareTextAccent{'}{OT1}{19}
7437 \DeclareTextAccent{^}{OT1}{94}
7438 \DeclareTextAccent`{OT1}{18}
7439 \DeclareTextAccent~{OT1}{126}

```

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

```

7440 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7441 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7442 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
7443 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7444 \DeclareTextSymbol{\i}{OT1}{16}
7445 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain $\text{T}_{\text{E}}\text{X}$ doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```

7446 \ifx\scriptsize\undefined
7447     \let\scriptsize\sevenrm
7448 \fi

```

And a few more “dummy” definitions.

```

7449 \def\language{english}%
7450 \let\bbl@opt@shorthands\@nnil
7451 \def\bbl@ifshorthand#1#2#3{#2}%
7452 \let\bbl@language@opts\@empty
7453 \ifx\babeloptionstrings\undefined
7454     \let\bbl@opt@strings\@nnil
7455 \else
7456     \let\bbl@opt@strings\babeloptionstrings
7457 \fi
7458 \def\BabelStringsDefault{generic}
7459 \def\bbl@tempa{normal}
7460 \ifx\babeloptionmath\bbl@tempa
7461     \def\bbl@mathnormal{\noexpand\textormath}
7462 \fi
7463 \def\AfterBabelLanguage#1#2{}
7464 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
7465 \let\bbl@afterlang\relax
7466 \def\bbl@opt@safe{BR}
7467 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi

```

```

7468 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
7469 \expandafter\newif\csname ifbbl@single\endcsname
7470 \chardef\bbl@bidimode\z@
7471 \</Emulate LaTeX>

```

A proxy file:

```

7472 <*plain>
7473 \input babel.def
7474 </plain>

```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.

During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).