

Babel

Version 3.66.2551
2021/11/09

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	35
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	42
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	47
1.27	Languages supported by babel with ldf files	48
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	50
1.31	Current and future work	51
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	52
2.1	Format	52
3	The interface between the core of babel and the language definition files	53
3.1	Guidelines for contributed languages	54
3.2	Basic macros	54
3.3	Skeleton	56
3.4	Support for active characters	57
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	58
3.8	Encoding-dependent strings	58
4	Changes	62
4.1	Changes in babel version 3.9	62

II	Source code	62
5	Identification and loading of required files	62
6	locale directory	63
7	Tools	63
7.1	Multiple languages	68
7.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	68
7.3	<code>base</code>	70
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	74
8.1	Selecting the language	76
8.2	Errors	85
8.3	Hooks	87
8.4	Setting up language files	89
8.5	Shorthands	91
8.6	Language attributes	101
8.7	Support for saving macro definitions	103
8.8	Short tags	104
8.9	Hyphens	104
8.10	Multiencoding strings	106
8.11	Macros common to a number of languages	112
8.12	Making glyphs available	113
8.12.1	Quotation marks	113
8.12.2	Letters	114
8.12.3	Shorthands for quotation marks	115
8.12.4	Umlauts and tremas	116
8.13	Layout	117
8.14	Load engine specific macros	118
8.15	Creating and modifying languages	118
9	Adjusting the Babel behavior	139
9.1	Cross referencing macros	142
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>	146
9.3.3	<code>hhline</code>	146
9.4	Encoding and fonts	147
9.5	Basic bidi support	149
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	157
12	Font handling with <code>fontspec</code>	161

13	Hooks for XeTeX and LuaTeX	165
13.1	XeTeX	165
13.2	Layout	167
13.3	LuaTeX	169
13.4	Southeast Asian scripts	175
13.5	CJK line breaking	176
13.6	Arabic justification	178
13.7	Common stuff	182
13.8	Automatic fonts and ids switching	183
13.9	Bidi	187
13.10	Layout	189
13.11	Lua: transforms	193
13.12	Lua: Auto bidi with basic and basic-r	201
14	Data for CJK	212
15	The ‘nil’ language	212
16	Support for Plain T_EX (plain.def)	213
16.1	Not renaming hyphen.tex	213
16.2	Emulating some L ^A T _E X features	214
16.3	General tools	214
16.4	Encoding related macros	218
17	Acknowledgements	221

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	6
You are loading directly a language style	8
Unknown language ‘LANG’	9
Argument of \language@active@arg” has an extra }	13
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 is 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 l_{UATEX}, the following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and `\today` in Danish and Vietnamese. No additional packages are required.

L_{UATEX}/X_{ETEX}

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

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

```

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

```

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

Spaces after the environment are ignored.

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

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

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

1.9 More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

```

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

```

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

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.

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

2. If on a certain level (system, language, user, language user) there is a one-char shorthand, two-char ones starting with that char and on the same level are ignored.
3. Since they are active, a shorthand cannot contain the same character in its definition (except if deactivated with, eg, `\string`).

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

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

New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add

`\languageshorthands{⟨lang⟩}` to the corresponding `\extras⟨lang⟩`, as explained below). By default, user shorthands are (re)defined. User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

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

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

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

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

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

`\languageshorthands` {⟨language⟩}

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` {⟨shorthand⟩}

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.)

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.

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

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

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

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

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

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

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

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

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

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the 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

⁶Thanks to Enrico Gregorio

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


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

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

1.11 Package options

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

- KeepShorthandsActive** Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
- activeacute** For some languages babel supports this options to set `'` as a shorthand in case it is not done by default.
- activegrave** Same for ```.
- shorthands=** `<char><char>... | off`
The only language shorthands activated are those given, like, eg:

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

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

- safe=** `none | ref | bib`
Some \TeX macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from `varioref` and `ifthen`). With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in $\epsilon\TeX$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).
- math=** `active | normal`
Shorthands are mainly intended for text, not for math. By setting this option with the value `normal` they are deactivated in math mode (default is `active`) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.

- config=** `<file>`
Load `<file>.cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).
- main=** `<language>`
Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.
- headfoot=** `<language>`
By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** New 3.9l Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** New 3.9l No warnings and no *infos* are written to the log file.⁸
- strings=** `generic | unicode | encoded | <label> | `
Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional \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

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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.

1.12 The base option

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

\AfterBabelLanguage `{⟨option-name⟩}{⟨code⟩}`

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

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

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

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

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

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 `...name` strings).

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.

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

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

LUATEX/XETEX

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

Or also:

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

NOTE The `ini` files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow (which could no longer be valid when you read this manual, if the packages involved had 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. Khemer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{ໂຄ ລຸ ລອ ລງ ລຸ ລາ} % Random
```

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

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

Latin, Greek, Cyrillic Combining chars with the default luatex font renderer might be wrong; on then 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}	as	Assamese
agq	Aghem	asa	Asu
ak	Akan	ast	Asturian ^{ul}
am	Amharic ^{ul}	az-Cyrl	Azerbaijani
ar	Arabic ^{ul}	az-Latn	Azerbaijani
ar-DZ	Arabic ^{ul}	az	Azerbaijani ^{ul}
ar-MA	Arabic ^{ul}	bas	Basaa
ar-SY	Arabic ^{ul}	be	Belarusian ^{ul}

bem	Bemba	fr-CA	French ^{ul}
bez	Bena	fr-CH	French ^{ul}
bg	Bulgarian ^{ul}	fr-LU	French ^{ul}
bm	Bambara	fur	Friulian ^{ul}
bn	Bangla ^{ul}	fy	Western Frisian
bo	Tibetan ^u	ga	Irish ^{ul}
brx	Bodo	gd	Scottish Gaelic ^{ul}
bs-Cyrl	Bosnian	gl	Galician ^{ul}
bs-Latn	Bosnian ^{ul}	grc	Ancient Greek ^{ul}
bs	Bosnian ^{ul}	gsw	Swiss German
ca	Catalan ^{ul}	gu	Gujarati
ce	Chechen	guz	Gusii
cgg	Chiga	gv	Manx
chr	Cherokee	ha-GH	Hausa
ckb	Central Kurdish	ha-NE	Hausa ^l
cop	Coptic	ha	Hausa
cs	Czech ^{ul}	haw	Hawaiian
cu	Church Slavic	he	Hebrew ^{ul}
cu-Cyrs	Church Slavic	hi	Hindi ^u
cu-Glag	Church Slavic	hr	Croatian ^{ul}
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

lb	Luxembourgish	rof	Rombo
lg	Ganda	ru	Russian ^{ul}
lkt	Lakota	rw	Kinyarwanda
ln	Lingala	rwk	Rwa
lo	Lao ^{ul}	sa-Beng	Sanskrit
lrc	Northern Luri	sa-Deva	Sanskrit
lt	Lithuanian ^{ul}	sa-Gujr	Sanskrit
lu	Luba-Katanga	sa-Knda	Sanskrit
luo	Luo	sa-Mlym	Sanskrit
luy	Luyia	sa-Telu	Sanskrit
lv	Latvian ^{ul}	sa	Sanskrit
mas	Masai	sah	Sakha
mer	Meru	saq	Samburu
mfe	Morisyen	sbp	Sangu
mg	Malagasy	se	Northern Sami ^{ul}
mgf	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian ^{ul}	sg	Sango
ml	Malayalam ^{ul}	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
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 ^l
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

uz-Latn	Uzbek	yue	Cantonese
uz	Uzbek	zgh	Standard Moroccan Tamazight
vai-Latn	Vai		
vai-Vaii	Vai	zh-Hans-HK	Chinese
vai	Vai	zh-Hans-MO	Chinese
vi	Vietnamese ^{ul}	zh-Hans-SG	Chinese
vun	Vunjo	zh-Hans	Chinese
wae	Walser	zh-Hant-HK	Chinese
xog	Soga	zh-Hant-MO	Chinese
yav	Yangben	zh-Hant	Chinese
yi	Yiddish	zh	Chinese
yo	Yoruba	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	bosnian-cyrillic
akan	bosnian-cyrl
albanian	bosnian-latin
american	bosnian-latn
amharic	bosnian
ancientgreek	brazilian
arabic	breton
arabic-algeria	british
arabic-DZ	bulgarian
arabic-morocco	burmese
arabic-MA	canadian
arabic-syria	cantonese
arabic-SY	catalan
armenian	centralatlastamazight
assamese	centralkurdish
asturian	chechen
asu	cherokee
australian	chiga
austrian	chinese-hans-hk
azerbaijani-cyrillic	chinese-hans-mo
azerbaijani-cyrl	chinese-hans-sg
azerbaijani-latin	chinese-hans
azerbaijani-latn	chinese-hant-hk
azerbaijani	chinese-hant-mo
bafia	chinese-hant
bambara	chinese-simplified-hongkongsarchina
basaa	chinese-simplified-macausarchina
basque	chinese-simplified-singapore
belarusian	chinese-simplified
bemba	chinese-traditional-hongkongsarchina
bena	chinese-traditional-macausarchina
bengali	chinese-traditional
bodo	chinese

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{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.

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

```
\addto\extrarussian{\mymacro}
```

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

The main language is not changed (danish in this example). So, you must add `\selectlanguage{arhinish}` or other selectors where necessary.

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

`import=` *⟨language-tag⟩*

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

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

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

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

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

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

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

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

captions= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

A special value is `+`, which allocates a new language (in the $\text{T}_{\text{E}}\text{X}$ 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= $\langle script-name \rangle$

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

language= $\langle language-name \rangle$

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

alph= $\langle counter-name \rangle$

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

Alph= $\langle counter-name \rangle$

Same for `\Alph`.

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

onchar= ids | fonts

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

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

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

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

intrapenalty= $\langle penalty \rangle$

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

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	Central Kurdish	Khmer	Northern Luri	Nepali
Assamese	Dzongkha	Kannada	Malayalam	Odia
Bangla	Persian	Konkani	Marathi	Punjabi
Tibetar	Gujarati	Kashmiri	Burmese	Pashto
Bodo	Hindi	Lao	Mazanderani	Tamil

Telugu	Uyghur	Uzbek	Cantonese
Thai	Urdu	Vai	Chinese

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

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

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):

- `\localnumeral{<style>}{<number>}`, like `\localnumeral{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
Belarusian, 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 an ini file may define strings for other calendars (currently ar, ar-*, he, fa, hi.) In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).

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

1.19 Accessing language info

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

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

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

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

\localeinfo {*<field>*}

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

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

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

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

`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 פרק.

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

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 store 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 \TeX : (1) the character used is that set for the current font, while in \TeX it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in \TeX , but it can be changed to another value by redefining `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). 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 `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 \TeX -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> .

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

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

Czech, Polish, Slovak	oneletter.nobreak	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	diaeresis.hyphen	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	transliteration.omega	Although he provided combinations are not exactly the same, this transform follows the syntax of Omega: = for the circumflex, v for digamma, and so on. For better compatibility with Levy's system, ~ (as 'string') is an alternative to =. ' is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.
Greek	sigma.final	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms does it. To prevent the conversion (an abbreviation, for example), write "s.
Hindi, Sanskrit	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` `{\langle locale-name \rangle}{\langle lua-pattern \rangle}{\langle replacement \rangle}`

New 3.44-3-52 It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences: (1) the first argument is the locale instead of the name of the hyphenation patterns; (2) in the search patterns = has no special meaning, while | stands for an ordinary space; (3) in the replacement, discretionaries are not accepted.

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

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

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

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

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

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

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

1.22 Selection based on BCP 47 tags

New 3.43 The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of

each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

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

`autoload.bcp47` with values on and off.

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

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

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

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

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

1.23 Selecting scripts

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

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

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

New 3.9i This macro makes sure $\langle text \rangle$ is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine `\TeX` and `\LaTeX` so that they are correctly typeset even with 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:

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.

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

    وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الاجريقي) بـ
    Arabia أو Aravia (بالاغريقية 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 `\texthe` must be defined to select the main language):

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the bidi package, which provides its own mechanism to control these elements). You may use several options with a 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 `TeX` primitive) which makes lists to be horizontally misplaced if they are inside a `\vbox` (like `minipage`) and the current direction is different from the main one. A workaround is to restore the main language before the box and then set the local one inside.

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr `{\lr-text}`

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

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

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

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

\BabelPatchSection `{\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` `{\langle cmd\rangle}{\langle local-language\rangle}{\langle before\rangle}{\langle after\rangle}`

New 3.17 Something like:

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

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

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

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

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

(which also redefine `\footnotetext` and define `\localfootnotetext` and `\mainfootnotetext`). If the language argument is empty, then no language is selected inside the argument of the footnote. Note this command is available always in bidi documents, even without `layout=footnotes`.

EXAMPLE If you want to preserve directionality in footnotes and there are many footnotes entirely in English, you can define:

```
\BabelFootnote{\enfootnote}{english}\{.}
```

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

1.25 Language attributes

`\languageattribute`

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

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

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

1.26 Hooks

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

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

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

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

\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

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

Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppsorbian
Welsh welsh

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchle, ethiop and friulan. Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension .dn:

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

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

1.28 Unicode character properties in luatex

New 3.32 Part of the babel job is to apply Unicode rules to some script-specific features based on some properties. Currently, they are 3, namely, direction (ie, bidi class), mirroring glyphs, and line breaking for CJK scripts. These properties are stored in lua tables, which you can modify with the following macro (for example, to set them for glyphs in the PUA).

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

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 $\backslash\text{babelprovide}$, or, if the last argument is empty, removes them. The last argument is the locale name:

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

1.29 Tweaking some features

`\babeladjust` `{\key-value-list}`

New 3.36 Sometimes you might need to disable some babel features. Currently this macro understands the following keys (and only for luatex), with values on or off: `bidi.text`, `bidi.mirroring`, `bidi.mapdigits`, `layout.lists`, `layout.tabular`, `linebreak.sea`, `linebreak.cjk`, `justify.arabic`. For example, you can set `\babeladjust{bidi.text=off}` if you are using an alternative algorithm or with large sections not requiring it. Use with care, because these options do not deactivate other related options (like paragraph direction with `bidi.text`).

1.30 Tips, workarounds, known issues and notes

- If you use the document class *book* and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), \TeX will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the safe option to `none` or `bib`.
- Both `ltxdoc` and `babel` use `\AtBeginDocument` to change some catcodes, and `babel` reloads `hline` to make sure `:` has the right one, so if you want to change the catcode of `|` it has to be done using the same method at the proper place, with

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

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

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

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

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

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

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

csquotes Logical markup for quotes.
iflang Tests correctly the current language.
hyphsubst Selects a different set of patterns for a language.
translator An open platform for packages that need to be localized.
siunitx Typesetting of numbers and physical quantities.
biblatex Programmable bibliographies and citations.
bicaption Bilingual captions.
babelbib Multilingual bibliographies.
microtype Adjusts the typesetting according to some languages (kerning and spacing). Ligatures can be disabled.
substitutefont Combines fonts in several encodings.
mkpattern Generates hyphenation patterns.
tracklang Tracks which languages have been requested.
ucharclasses (xetex) Switches fonts when you switch from one Unicode block to another.
zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better). Useful additions would be, for example, time, currency, addresses and personal names.²¹ But that is the easy part, because they don't require modifying the \LaTeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study. Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ből”, in Spanish an item labelled “3.^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 and 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.

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

2 Loading languages with `language.dat`

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

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

2.1 Format

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

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

```
% 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
```

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

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

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

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

```
rebuild the format. Now I will use the patterns
preloaded for english instead}}
```

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

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

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

The following assumptions are made:

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

Some recommendations:

- The preferred shorthand is `"`, which is not used in \LaTeX (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.

- Avoid adding things to `\noextras<lang>` except for `umlauth` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras<lang>`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁶
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by `babel` and it can lead to incompatibilities.

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

3.1 Guidelines for contributed languages

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

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

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

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

The following page provides a starting point for `ldf` files:

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

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

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

3.2 Basic macros

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

<code>\addlanguage</code>	The macro <code>\addlanguage</code> is a non-outer version of the macro <code>\newlanguage</code> , defined in <code>plain.tex</code> version 3.x. Here “language” is used in the T _E X sense of set of hyphenation patterns.
<code>\adddialect</code>	The macro <code>\adddialect</code> 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 <code>\language0</code> . Here “language” is used in the T _E X sense of set of hyphenation patterns.
<code>\<lang>hyphenmins</code>	The macro <code>\<lang>hyphenmins</code> is used to store the values of the <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> . Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:
<pre>\renewcommand\spanishhyphenmins{34}</pre>	
	(Assigning <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> directly in <code>\extras<lang></code> has no effect.)
<code>\providehyphenmins</code>	The macro <code>\providehyphenmins</code> should be used in the language definition files to set <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> . This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do <i>not</i> set them).
<code>\captions<lang></code>	The macro <code>\captions<lang></code> defines the macros that hold the texts to replace the original hard-wired texts.
<code>\date<lang></code>	The macro <code>\date<lang></code> defines <code>\today</code> .
<code>\extras<lang></code>	The macro <code>\extras<lang></code> contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
<code>\noextras<lang></code>	Because we want to let the user switch between languages, but we do not know what state T _E X might be in after the execution of <code>\extras<lang></code> , a macro that brings T _E X into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras<lang></code> .
<code>\bbl@declare@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the L ^A T _E X 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 @-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 @-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 @-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, L ^A T _E X 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> .

²⁶But not removed, for backward compatibility.

`\substitutefontfamily` (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct L^AT_EX 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>

\bb{<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:

<code>\AtEndOfPackage{%</code>	
<code>\RequirePackage{dingbat}%</code>	Delay package
<code>\savebox{\myeye}{\eye}%</code>	And direct usage
<code>\newsavebox{\myeye}</code>	
<code>\newcommand\myanchor{\anchor}%</code>	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 \LaTeX to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

`\bbl@activate`

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

`\bbl@deactivate`

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

`\declare@shorthand`

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

`\bbl@add@special`

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

`\bbl@remove@special`

It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \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

²⁷This mechanism was introduced by Bernd Raichle.

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

<code>\bbl@allowhyphens</code>	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 <code>\bbl@allowhyphens</code> can be used.
<code>\allowhyphens</code>	Same as <code>\bbl@allowhyphens</code> , but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with <code>\accent</code> in OT1. Note the previous command (<code>\bbl@allowhyphens</code>) has different applications (hyphens and discretionaries) than this one (composite chars). Note also prior to version 3.7, <code>\allowhyphens</code> had the behavior of <code>\bbl@allowhyphens</code> .
<code>\set@low@box</code>	For some languages, quotes need to be lowered to the baseline. For this purpose the macro <code>\set@low@box</code> is available. It takes one argument and puts that argument in an <code>\hbox</code> , at the baseline. The result is available in <code>\box0</code> for further processing.
<code>\save@sf@q</code>	Sometimes it is necessary to preserve the <code>\spacefactor</code> . For this purpose the macro <code>\save@sf@q</code> is available. It takes one argument, saves the current <code>\spacefactor</code> , executes the argument, and restores the <code>\spacefactor</code> .
<code>\bbl@frenchspacing</code> <code>\bbl@nonfrenchspacing</code>	The commands <code>\bbl@frenchspacing</code> and <code>\bbl@nonfrenchspacing</code> can be used to properly switch French spacing on and off.

3.8 Encoding-dependent strings

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

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

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

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

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

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

Encoding info is `charset=` followed by a `charset`, which if given sets how the strings should be translated to the internal representation used by the engine, typically `utf8`, which is the

only value supported currently (default is no translations). Note `charset` is applied by `luatex` and `xetex` when reading the file, not when the macro or string is used in the document.

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiname{M}{a}rz}
\SetString\monthivname{April}
\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
\SetString\monthviiname{Juli}
\SetString\monthviiname{August}
```

²⁸In future releases further categories may be added.

```

\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

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

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

$\backslash EndBabelCommands$ Marks the end of the series of blocks.

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

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

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

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

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

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

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

```

\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.

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

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

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would typically be things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A *map-list* is a series of macros using the internal format of `@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 L^AT_EX, we can set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

`\SetHyphenMap` *{(to-lower-macros)}*

New 3.9g Case mapping serves in 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 though the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{<uccode-from>}{<uccode-to>}{<step>}{<lccode>}` loops though the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for *many-to-one*).

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

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=3.66.2551>>
2 <<date=2021/11/09>>
```

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 \TeX is executed twice, but we need them when defining options and `babel.def` cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```
3 <<(*Basic macros)>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1#2}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
```



```

13 \def\bbl@cl#1{\csname bbl@#1\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19   \fi}
20 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}

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

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

\bbl@afterelse 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##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}}

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

97 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
98   \toks@{}}%
99   \def\bbl@replace@aux##1#2##2#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 checking 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_{TEX}, 1 is luatex, and 2 is xetex. You may use the latter 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 \LaTeX macro. The following code is placed before them to define (and then undefine) if not in \LaTeX .

```

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 \g@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 \TeX . After it, we will resume the \TeX -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 `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```
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}{\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{\edef#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}}%
489   \ifx\@empty#2%
490     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
491   \else\ifx\@empty#3%
492     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
493     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
494     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
495     }%
496     \ifx\bbl@bcp\relax
497       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
498     \fi
499   \else
500     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
501     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
502     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
503     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
504     }%
505     \ifx\bbl@bcp\relax
506       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
507       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
508     }%
509     \fi
510     \ifx\bbl@bcp\relax
511       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
512       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
513     }%
514     \fi
515     \ifx\bbl@bcp\relax
516       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
517     \fi
518   \fi\fi}
519 \let\bbl@initoload\relax
520 \def\bbl@provide@locale{%
521   \ifx\babelprovide\@undefined
522     \bbl@error{For a language to be defined on the fly 'base'\\%
523               is not enough, and the whole package must be\\%
524               loaded. Either delete the 'base' option or\\%
525               request the languages explicitly}%
526     {See the manual for further details.}%
527   \fi
528 % TODO. Option to search if loaded, with \LocaleForEach
529 \let\bbl@auxname\languagename % Still necessary. TODO
530 \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
531 {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%

```

```

532 \ifbbl@bcpallowed
533   \expandafter\ifx\csname date\language\endcsname\relax
534     \expandafter
535     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@
536     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
537       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
538       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
539       \expandafter\ifx\csname date\language\endcsname\relax
540         \let\bbl@initoload\bbl@bcp
541         \bbl@exp{\\\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
542         \let\bbl@initoload\relax
543       \fi
544       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
545     \fi
546   \fi
547 \fi
548 \expandafter\ifx\csname date\language\endcsname\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\csname l@#1\endcsname=\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\csname selectlanguage \endcsname}

```

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

```

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 \ifx\bbl@selectorname\@empty
608 \def\bbl@selectorname{select}%
609 \fi
610 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
611 \bbl@push@language
612 \aftergroup\bbl@pop@language
613 \bbl@set@language{#1}}

```

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

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

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

```

614 \def\BabelContentsFiles{toc,lof,lot}
615 \def\bbl@set@language#1{% from selectlanguage, pop@
616 % The old buggy way. Preserved for compatibility.
617 \edef\language{%
618 \ifnum\escapechar=\expandafter`\string#1\@empty
619 \else\string#1\@empty\fi}%
620 \ifcat\relax\noexpand#1%
621 \expandafter\ifx\csname date\language\endcsname\relax
622 \edef\language{#1}%
623 \let\localename\language
624 \else
625 \bbl@info{Using '\string\language' instead of 'language' is\\%
626 deprecated. If what you want is to use a\\%
627 macro containing the actual locale, make\\%
628 sure it does not not match any language.\\%
629 Reported}%
630 \ifx\scantokens\undefined
631 \def\localename{??}%
632 \else
633 \scantokens\expandafter{\expandafter
634 \def\expandafter\localename\expandafter{\language}}%
635 \fi
636 \fi
637 \else
638 \def\localename{#1}% This one has the correct catcodes
639 \fi
640 \select@language{\language}%

```

```

641 % write to auxs
642 \expandafter\ifx\csname date\language\endcsname\relax\else
643   \if@files
644     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
645       \bbl@savelastskip
646       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
647       \bbl@restorelastskip
648     \fi
649     \bbl@usehooks{write}}}%
650   \fi
651 \fi}
652 %
653 \let\bbl@restorelastskip\relax
654 \let\bbl@savelastskip\relax
655 %
656 \newif\ifbbl@bcpallowed
657 \bbl@bcpallowedfalse
658 \def\select@language#1{% from set@, babel@aux
659   \ifx\bbl@select@name\@empty
660     \def\bbl@select@name{select}%
661   % set hymap
662   \fi
663   \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
664   % set name
665   \edef\language{#1}%
666   \bbl@fixname\language
667   % TODO. name@map must be here?
668   \bbl@provide@locale
669   \bbl@iflanguage\language{%
670     \expandafter\ifx\csname date\language\endcsname\relax
671       \bbl@error
672       {Unknown language '\language'. Either you have\\%
673         misspelled its name, it has not been installed,\\%
674         or you requested it in a previous run. Fix its name,\\%
675         install it or just rerun the file, respectively. In\\%
676         some cases, you may need to remove the aux file}%
677       {You may proceed, but expect wrong results}%
678     \else
679       % set type
680       \let\bbl@select@type\z@
681       \expandafter\bbl@switch\expandafter{\language}%
682     \fi}}
683 \def\babel@aux#1#2{%
684   \select@language{#1}%
685   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
686     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
687 \def\babel@toc#1#2{%
688   \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 `\leftthyphenmin` and `\rightthyphenmin` 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.

```

689 \newif\ifbbl@usedategroup
690 \def\bbl@switch#1{% from select@, foreign@
691 % make sure there is info for the language if so requested
692 \bbl@ensureinfo{#1}%
693 % restore
694 \originalTeX
695 \expandafter\def\expandafter\originalTeX\expandafter{%
696 \csname noextras#1\endcsname
697 \let\originalTeX\@empty
698 \babel@beginsave}%
699 \bbl@usehooks{afterreset}{}%
700 \languageshorthands{none}%
701 % set the locale id
702 \bbl@id@assign
703 % switch captions, date
704 % No text is supposed to be added here, so we remove any
705 % spurious spaces.
706 \bbl@bsphack
707 \ifcase\bbl@select@type
708 \csname captions#1\endcsname\relax
709 \csname date#1\endcsname\relax
710 \else
711 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
712 \ifin@
713 \csname captions#1\endcsname\relax
714 \fi
715 \bbl@xin@{,date,}{, \bbl@select@opts,}%
716 \ifin@ % if \foreign... within \<lang>date
717 \csname date#1\endcsname\relax
718 \fi
719 \fi
720 \bbl@esphack
721 % switch extras
722 \bbl@usehooks{beforeextras}{}%
723 \csname extras#1\endcsname\relax
724 \bbl@usehooks{afterextras}{}%
725 % > babel-ensure
726 % > babel-sh-<short>
727 % > babel-bidi
728 % > babel-fontspec
729 % hyphenation - case mapping
730 \ifcase\bbl@opt@hyphenmap\or
731 \def\BabelLower##1##2{\lcode##1=##2\relax}%
732 \ifnum\bbl@hymapsel>4\else
733 \csname\language @bbl@hyphenmap\endcsname
734 \fi
735 \chardef\bbl@opt@hyphenmap\z@
736 \else
737 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
738 \csname\language @bbl@hyphenmap\endcsname
739 \fi
740 \fi
741 \let\bbl@hymapsel\@cclv
742 % hyphenation - select rules
743 \ifnum\csname l@\language\endcsname=\l@unhyphenated
744 \edef\bbl@tempa{u}%

```

```

745 \else
746 \edef\bbl@tempa{\bbl@c1{lnbrk}}%
747 \fi
748 % linebreaking - handle u, e, k (v in the future)
749 \bbl@xin@{/u}{/\bbl@tempa}%
750 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
751 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
752 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
753 \ifin@
754 % unhyphenated/kashida/elongated = allow stretching
755 \language\l@unhyphenated
756 \babel@savevariable\emergencystretch
757 \emergencystretch\maxdimen
758 \babel@savevariable\hbadness
759 \hbadness\@M
760 \else
761 % other = select patterns
762 \bbl@patterns{#1}%
763 \fi
764 % hyphenation - mins
765 \babel@savevariable\lefthyphenmin
766 \babel@savevariable\righthyphenmin
767 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
768 \set@hyphenmins\tw@\thr@\relax
769 \else
770 \expandafter\expandafter\expandafter\set@hyphenmins
771 \csname #1hyphenmins\endcsname\relax
772 \fi
773 \let\bbl@selectorname\@empty}

```

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.

```

774 \long\def\otherlanguage#1{%
775 \def\bbl@selectorname{other}%
776 \ifnum\bbl@hymapsel=\@cc1v\let\bbl@hymapsel\thr@\fi
777 \csname selectlanguage \endcsname{#1}%
778 \ignorespaces}

```

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

```

779 \long\def\endotherlanguage{%
780 \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`.

```

781 \expandafter\def\csname otherlanguage*\endcsname{%
782 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}%
783 \def\bbl@otherlanguage@s[#1]#2{%
784 \def\bbl@selectorname{other*}%
785 \ifnum\bbl@hymapsel=\@cc1v\chardef\bbl@hymapsel4\relax\fi
786 \def\bbl@select@opts{#1}%
787 \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”.

```
788 \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.

```
789 \providecommand\bbl@beforeforeign{}
790 \edef\foreignlanguage{%
791   \noexpand\protect
792   \expandafter\noexpand\csname foreignlanguage \endcsname}
793 \expandafter\def\csname foreignlanguage \endcsname{%
794   \@ifstar\bbl@foreign@s\bbl@foreign@x}
795 \providecommand\bbl@foreign@x[3][]{%
796   \begingroup
797     \def\bbl@selectorname{foreign}%
798     \def\bbl@select@opts{#1}%
799     \let\BabelText\@firstofone
800     \bbl@beforeforeign
801     \foreign@language{#2}%
802     \bbl@usehooks{foreign}{}%
803     \BabelText{#3}% Now in horizontal mode!
804   \endgroup}
805 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@par
806   \begingroup
807     {\par}%
808     \def\bbl@selectorname{foreign*}%
809     \let\bbl@select@opts\@empty
810     \let\BabelText\@firstofone
811     \foreign@language{#1}%
812     \bbl@usehooks{foreign*}{}%
813     \bbl@dirparastext
814     \BabelText{#2}% Still in vertical mode!
815     {\par}%
816   \endgroup}
```

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

```
817 \def\foreign@language#1{%
```

```

818 % set name
819 \edef\language{#1}%
820 \ifbbl@usedategroup
821   \bbl@add\bbl@select@opts{,date,}%
822   \bbl@usedategroupfalse
823 \fi
824 \bbl@fixname\language
825 % TODO. name@map here?
826 \bbl@provide@locale
827 \bbl@iflanguage\language{
828   \expandafter\ifx\csname date\language\endcsname\relax
829     \bbl@warning % TODO - why a warning, not an error?
830     {Unknown language '#1'. Either you have\\%
831      misspelled its name, it has not been installed,\\%
832      or you requested it in a previous run. Fix its name,\\%
833      install it or just rerun the file, respectively. In\\%
834      some cases, you may need to remove the aux file.\\%
835      I'll proceed, but expect wrong results.\\%
836      Reported}%
837   \fi
838 % set type
839 \let\bbl@select@type\@ne
840 \expandafter\bbl@switch\expandafter{\language}}

```

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

```

841 \def\IfBabelSelectorTF#1{%
842   \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
843   \ifin@
844     \expandafter\@firstoftwo
845   \else
846     \expandafter\@secondoftwo
847   \fi}

```

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

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

```

848 \let\bbl@hyphlist\@empty
849 \let\bbl@hyphenation@\relax
850 \let\bbl@pttnlist\@empty
851 \let\bbl@patterns@\relax
852 \let\bbl@hymapsel=\@cclv
853 \def\bbl@patterns#1{%
854   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
855     \csname l@#1\endcsname
856     \edef\bbl@tempa{#1}%
857   \else
858     \csname l@#1:\f@encoding\endcsname
859     \edef\bbl@tempa{#1:\f@encoding}%
860   \fi
861   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
862   % > luatex
863   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
864     \begingroup

```

```

865 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
866 \ifin@else
867 \expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
868 \hyphenation{%
869 \bbl@hyphenation@
870 \@ifundefined{bbl@hyphenation@#1}%
871 \empty
872 {\space\csname bbl@hyphenation@#1\endcsname}}%
873 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
874 \fi
875 \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*`.

```

876 \def\hyphenrules#1{%
877 \edef\bbl@tempf{\#1}%
878 \bbl@fixname\bbl@tempf
879 \bbl@iflanguage\bbl@tempf{%
880 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
881 \ifx\languageshortands\undefined\else
882 \languageshortands{none}%
883 \fi
884 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
885 \set@hyphenmins\tw@\thr@@\relax
886 \else
887 \expandafter\expandafter\expandafter\set@hyphenmins
888 \csname\bbl@tempf hyphenmins\endcsname\relax
889 \fi}}
890 \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.

```

891 \def\providehyphenmins#1#2{%
892 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
893 \@namedef{#1hyphenmins}{#2}%
894 \fi}

```

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

```

895 \def\set@hyphenmins#1#2{%
896 \lefthyphenmin#1\relax
897 \righthyphenmin#2\relax}

```

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

```

898 \ifx\ProvidesFile\undefined
899 \def\ProvidesLanguage#1[#2 #3 #4]{%
900 \wlog{Language: #1 #4 #3 <#2>}%
901 }
902 \else
903 \def\ProvidesLanguage#1{%
904 \begingroup
905 \catcode`\ 10 %

```

```

906     \@makeother\/%
907     \@ifnextchar[%]
908         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
909 \def\@provideslanguage#1[#2]{%
910     \wlog{Language: #1 #2}%
911     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
912     \endgroup}
913 \fi

```

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

```
914 \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.

```
915 \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’:

```

916 \providecommand\setlocale{%
917     \bbl@error
918     {Not yet available}%
919     {Find an armchair, sit down and wait}}
920 \let\uselocale\setlocale
921 \let\locale\setlocale
922 \let\selectlocale\setlocale
923 \let\textlocale\setlocale
924 \let\textlanguage\setlocale
925 \let\languagegetext\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.

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

```

926 \edef\bbl@nulllanguage{\string\language=0}
927 \def\bbl@nocaption{\protect\bbl@nocaption@i}
928 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
929     \global\@namedef{#2}{\textbf{?#1?}}}%
930     \@nameuse{#2}%
931 \edef\bbl@tempa{#1}%
932 \bbl@sreplace\bbl@tempa{name}{}%
933 \bbl@warning{% TODO.
934     \@backslashchar#1 not set for '\language'. Please,\\%
935     define it after the language has been loaded\\%
936     (typically in the preamble) with:\\%
937     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
938     Reported}}
939 \def\bbl@tentative{\protect\bbl@tentative@i}
940 \def\bbl@tentative@i#1{%
941     \bbl@warning{%

```

```

942   Some functions for '#1' are tentative.\\%
943   They might not work as expected and their behavior\\%
944   could change in the future.\\%
945   Reported}}
946 \def\nolanerr#1{%
947   \bbl@error
948   {You haven't defined the language '#1' yet.\\%
949     Perhaps you misspelled it or your installation\\%
950     is not complete}%
951   {Your command will be ignored, type <return> to proceed}}
952 \def\nopatterns#1{%
953   \bbl@warning
954   {No hyphenation patterns were preloaded for\\%
955     the language '#1' into the format.\\%
956     Please, configure your TeX system to add them and\\%
957     rebuild the format. Now I will use the patterns\\%
958     preloaded for \bbl@nulllanguage\space instead}}
959 \let\bbl@usehooks\@gobbletwo
960 \ifx\bbl@onlyswitch\@empty\endinput\fi
961 % Here ended switch.def

```

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

```

962 \ifx\directlua\@undefined\else
963   \ifx\bbl@luapatterns\@undefined
964     \input luababel.def
965   \fi
966 \fi
967 <<Basic macros>>
968 \bbl@trace{Compatibility with language.def}
969 \ifx\bbl@languages\@undefined
970   \ifx\directlua\@undefined
971     \openin1 = language.def % TODO. Remove hardcoded number
972     \ifeof1
973       \closein1
974       \message{I couldn't find the file language.def}
975     \else
976       \closein1
977       \begingroup
978         \def\addlanguage#1#2#3#4#5{%
979           \expandafter\ifx\csname lang@#1\endcsname\relax\else
980             \global\expandafter\let\csname l@#1\endcsname
981               \csname lang@#1\endcsname
982           \fi}%
983         \def\uselanguage#1{%
984           \input language.def
985         \endgroup
986       \fi
987     \fi
988     \chardef\l@english\z@
989 \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.

```

990 \def\addto#1#2{%
991   \ifx#1\@undefined
992     \def#1{#2}%
993   \else

```

```

994 \ifx#1\relax
995 \def#1{#2}%
996 \else
997 {\toks@\expandafter{#1#2}%
998 \xdef#1{the\toks@}}%
999 \fi
1000 \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?

```

1001 \def\bbl@withactive#1#2{%
1002 \begingroup
1003 \lccode`~=`#2\relax
1004 \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`.

```

1005 \def\bbl@redefine#1{%
1006 \edef\bbl@tempa{\bbl@stripslash#1}%
1007 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1008 \expandafter\def\csname\bbl@tempa\endcsname{
1009 \@onlypreamble\bbl@redefine

```

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

```

1010 \def\bbl@redefine@long#1{%
1011 \edef\bbl@tempa{\bbl@stripslash#1}%
1012 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1013 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1014 \@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_`.

```

1015 \def\bbl@redefineroobust#1{%
1016 \edef\bbl@tempa{\bbl@stripslash#1}%
1017 \bbl@ifunset{\bbl@tempa\space}%
1018 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1019 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1020 {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1021 \@namedef{\bbl@tempa\space}}
1022 \@onlypreamble\bbl@redefineroobust

```

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.

```

1023 \bbl@trace{Hooks}
1024 \newcommand\AddBabelHook[3][{}]{%
1025 \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}}%
1026 \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1027 \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1028 \bbl@ifunset{bbl@ev@#2@#3@#1}%

```



```

1029   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1030   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1031   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1032 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1033 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1034 \def\bbl@usehooks#1#2{%
1035   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1036   \def\bbl@elth##1{%
1037     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}}%
1038   \bbl@cs{ev@#1@}%
1039   \ifx\language\@undefined\else % Test required for Plain (?)
1040     \ifx\UseHook\@undefined\else\UseHook{babel/\language/#1}\fi
1041     \def\bbl@elth##1{%
1042       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}}%
1043     \bbl@cl{ev@#1}%
1044   \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).

```

1045 \def\bbl@evargs{,% <- don't delete this comma
1046   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1047   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1048   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1049   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1050   beforestart=0,language=2}
1051 \ifx\NewHook\@undefined\else
1052   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1053   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1054 \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.

```

1055 \bbl@trace{Defining babelensure}
1056 \newcommand\babelensure[2][{}% TODO - revise test files
1057   \AddBabelHook{babel-ensure}{afterextras}{%
1058     \ifcase\bbl@select@type
1059       \bbl@cl{e}%
1060     \fi}%
1061 \begingroup
1062   \let\bbl@ens@include\@empty
1063   \let\bbl@ens@exclude\@empty
1064   \def\bbl@ens@fontenc{\relax}%
1065   \def\bbl@tempb##1{%
1066     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1067   \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1068   \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1069   \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1070   \def\bbl@tempc{\bbl@ensure}%
1071   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1072     \expandafter{\bbl@ens@include}}%
1073   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%

```

```

1074 \expandafter{\bbl@ens@exclude}}%
1075 \toks@\expandafter{\bbl@tempc}%
1076 \bbl@exp{%
1077 \endgroup
1078 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1079 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1080 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1081 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1082 \edef##1{\noexpand\bbl@nocaption
1083 {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1084 \fi
1085 \ifx##1\@empty\else
1086 \in@{##1}{#2}%
1087 \ifin@\else
1088 \bbl@ifunset{\bbl@ensure@\language\name}%
1089 {\bbl@exp{%
1090 \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1091 \\\foreignlanguage{\language\name}%
1092 {\ifx\relax#3\else
1093 \\\fontencoding{#3}\selectfont
1094 \fi
1095 #####1}}}%
1096 {}}%
1097 \toks@\expandafter{##1}%
1098 \edef##1{%
1099 \bbl@csarg\noexpand{\ensure@\language\name}%
1100 {\the\toks@}}}%
1101 \fi
1102 \expandafter\bbl@tempb
1103 \fi}%
1104 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1105 \def\bbl@tempa##1{% elt for include list
1106 \ifx##1\@empty\else
1107 \bbl@csarg\in@{\ensure@\language\name\expandafter}\expandafter{##1}%
1108 \ifin@\else
1109 \bbl@tempb##1\@empty
1110 \fi
1111 \expandafter\bbl@tempa
1112 \fi}%
1113 \bbl@tempa#1\@empty}
1114 \def\bbl@captionslist{%
1115 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1116 \contentsname\listfigurename\listtablename\indexname\figurename
1117 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1118 \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`.

```

1119 \bbl@trace{Macros for setting language files up}
1120 \def\bbl@ldfinit{%
1121   \let\bbl@screset\@empty
1122   \let\BabelStrings\bbl@opt@string
1123   \let\BabelOptions\@empty
1124   \let\BabelLanguages\relax
1125   \ifx\originalTeX\@undefined
1126     \let\originalTeX\@empty
1127   \else
1128     \originalTeX
1129   \fi}
1130 \def\LdfInit#1#2{%
1131   \chardef\atcatcode=\catcode`\@
1132   \catcode`\@=11\relax
1133   \chardef\eqcatcode=\catcode`\=
1134   \catcode`\==12\relax
1135   \expandafter\if\expandafter\@backslashchar
1136     \expandafter\@car\string#2\@nil
1137   \ifx#2\@undefined\else
1138     \ldf@quit{#1}%
1139   \fi
1140 \else
1141   \expandafter\ifx\csname#2\endcsname\relax\else
1142     \ldf@quit{#1}%
1143   \fi
1144 \fi
1145 \bbl@ldfinit}

```

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

```

1146 \def\ldf@quit#1{%
1147   \expandafter\main@language\expandafter{#1}%
1148   \catcode`\@=\atcatcode \let\atcatcode\relax
1149   \catcode`\==\eqcatcode \let\eqcatcode\relax
1150   \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.

```

1151 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1152   \bbl@afterlang
1153   \let\bbl@afterlang\relax
1154   \let\BabelModifiers\relax
1155   \let\bbl@screset\relax}%
1156 \def\ldf@finish#1{%
1157   \loadlocalcfg{#1}%
1158   \bbl@afterldf{#1}%
1159   \expandafter\main@language\expandafter{#1}%
1160   \catcode`\@=\atcatcode \let\atcatcode\relax
1161   \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 .

```
1162 \@onlypreamble\LdfInit
1163 \@onlypreamble\ldf@quit
1164 \@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.

```
1165 \def\main@language#1{%
1166   \def\bbl@main@language{#1}%
1167   \let\language\name\bbl@main@language % TODO. Set localename
1168   \bbl@id@assign
1169   \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`.

```
1170 \def\bbl@beforestart{%
1171   \def\@nolanerr##1{%
1172     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1173   \bbl@usehooks{beforestart}}%
1174   \global\let\bbl@beforestart\relax}
1175 \AtBeginDocument{%
1176   {\@nameuse{bbl@beforestart}}% Group!
1177   \if@filesw
1178     \providecommand\babel@aux[2]{}%
1179     \immediate\write\@mainaux{%
1180       \string\providecommand\string\babel@aux[2]{}%
1181       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1182   \fi
1183   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1184   \ifbbl@single % must go after the line above.
1185     \renewcommand\selectlanguage[1]{}%
1186     \renewcommand\foreignlanguage[2]{#2}%
1187     \global\let\babel@aux\@gobbletwo % Also as flag
1188   \fi
1189   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1190 \def\select@language@x#1{%
1191   \ifcase\bbl@select@type
1192     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1193   \else
1194     \select@language{#1}%
1195   \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 \LaTeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```
1196 \bbl@trace{Shorhands}
1197 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1198   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1199   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
```

```

1200 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1201 \begingroup
1202 \catcode`#1\active
1203 \nfss@catcodes
1204 \ifnum\catcode`#1=\active
1205 \endgroup
1206 \bbl@add\nfss@catcodes{\@makeother#1}%
1207 \else
1208 \endgroup
1209 \fi
1210 \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.

```

1211 \def\bbl@remove@special#1{%
1212 \begingroup
1213 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1214 \else\noexpand##1\noexpand##2\fi}%
1215 \def\do{\x\do}%
1216 \def\@makeother{\x\@makeother}%
1217 \edef\x{\endgroup
1218 \def\noexpand\dospecials{\dospecials}%
1219 \expandafter\ifx\curname @sanitize\endcurname\relax\else
1220 \def\noexpand\@sanitize{\@sanitize}%
1221 \fi}%
1222 \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).

```

1223 \def\bbl@active@def#1#2#3#4{%
1224 \@namedef{#3#1}{%
1225 \expandafter\ifx\curname#2@sh@#1\endcurname\relax
1226 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1227 \else
1228 \bbl@afterfi\curname#2@sh@#1\endcurname
1229 \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.

```

1230 \long\@namedef{#3@arg#1}##1{%
1231 \expandafter\ifx\curname#2@sh@#1\string##1\endcurname\relax
1232 \bbl@afterelse\curname#4#1\endcurname##1%
1233 \else

```

```

1234 \bbl@afterfi\csname#2@sh@#1@\string##1@endcsname
1235 \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.

```

1236 \def\initiate@active@char#1{%
1237 \bbl@ifunset{active@char\string#1}%
1238 {\bbl@withactive
1239 {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1240 {}}

```

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).

```

1241 \def\@initiate@active@char#1#2#3{%
1242 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1243 \ifx#1\@undefined
1244 \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1245 \else
1246 \bbl@csarg\let{oridef@#2}#1%
1247 \bbl@csarg\edef{oridef@#2}{%
1248 \let\noexpand#1%
1249 \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
1250 \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).

```

1251 \ifx#1#3\relax
1252 \expandafter\let\csname normal@char#2@endcsname#3%
1253 \else
1254 \bbl@info{Making #2 an active character}%
1255 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1256 \@namedef{normal@char#2}{%
1257 \textormath{#3}{\csname bbl@oridef@@#2@endcsname}}%
1258 \else
1259 \@namedef{normal@char#2}{#3}%
1260 \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).

```

1261 \bbl@restoreactive{#2}%
1262 \AtBeginDocument{%
1263 \catcode`#2\active
1264 \if@filesw
1265 \immediate\write\@mainaux{\catcode`\string#2\active}%
1266 \fi}%
1267 \expandafter\bbl@add@special\csname#2@endcsname
1268 \catcode`#2\active
1269 \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⟩`).

```

1270 \let\bbl@tempa\@firstoftwo
1271 \if\string^#2%
1272   \def\bbl@tempa{\noexpand\textormath}%
1273 \else
1274   \ifx\bbl@mathnormal\@undefined\else
1275     \let\bbl@tempa\bbl@mathnormal
1276   \fi
1277 \fi
1278 \expandafter\edef\csname active@char#2\endcsname{%
1279   \bbl@tempa
1280     {\noexpand\if@safe@actives
1281       \noexpand\expandafter
1282       \expandafter\noexpand\csname normal@char#2\endcsname
1283     \noexpand\else
1284       \noexpand\expandafter
1285       \expandafter\noexpand\csname bbl@doactive#2\endcsname
1286     \noexpand\fi}%
1287   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1288 \bbl@csarg\edef{doactive#2}{%
1289   \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!).

```

1290 \bbl@csarg\edef{active@#2}{%
1291   \noexpand\active@prefix\noexpand#1%
1292   \expandafter\noexpand\csname active@char#2\endcsname}%
1293 \bbl@csarg\edef{normal@#2}{%
1294   \noexpand\active@prefix\noexpand#1%
1295   \expandafter\noexpand\csname normal@char#2\endcsname}%
1296 \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.

```

1297 \bbl@active@def#2\user@group{user@active}{language@active}%
1298 \bbl@active@def#2\language@group{language@active}{system@active}%
1299 \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.

```

1300 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1301   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1302 \expandafter\edef\csname\user@group @sh#2@string\protect@\endcsname
1303   {\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.

```

1304 \if\string'#2%
1305 \let\prim@s\bbl@prim@s
1306 \let\active@math@prime#1%
1307 \fi
1308 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1309 <<{*More package options}>> ≡
1310 \DeclareOption{math=active}{}
1311 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1312 <</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.

```

1313 \@ifpackagewith{babel}{KeepShorthandsActive}%
1314 {\let\bbl@restoreactive\@gobble}%
1315 {\def\bbl@restoreactive#1{%
1316 \bbl@exp{%
1317 \\\AfterBabelLanguage\\CurrentOption
1318 {\catcode`#1=\the\catcode`#1\relax}%
1319 \\\AtEndOfPackage
1320 {\catcode`#1=\the\catcode`#1\relax}}}%
1321 \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.

```

1322 \def\bbl@sh@select#1#2{%
1323 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1324 \bbl@afterelse\bbl@scndcs
1325 \else
1326 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1327 \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 `\ifincsname` is available. If there is, the expansion will be more robust.

```

1328 \begingroup
1329 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1330 {\gdef\active@prefix#1{%
1331 \ifx\protect\@typeset@protect
1332 \else
1333 \ifx\protect\@unexpandable@protect
1334 \noexpand#1%
1335 \else
1336 \protect#1%
1337 \fi
1338 \expandafter\@gobble
1339 \fi}}
1340 {\gdef\active@prefix#1{%
1341 \ifincsname
1342 \string#1%
1343 \expandafter\@gobble

```



```

1344     \else
1345     \ifx\protect\@typeset@protect
1346     \else
1347     \ifx\protect\@unexpandable@protect
1348     \noexpand#1%
1349     \else
1350     \protect#1%
1351     \fi
1352     \expandafter\expandafter\expandafter\@gobble
1353     \fi
1354     \fi}}
1355 \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>`.

```

1356 \newif\if@safe@actives
1357 \@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.

```

1358 \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`.

```

1359 \chardef\bbl@activated\z@
1360 \def\bbl@activate#1{%
1361   \chardef\bbl@activated\@ne
1362   \bbl@withactive{\expandafter\let\expandafter}#1%
1363   \csname bbl@active@\string#1\endcsname}
1364 \def\bbl@deactivate#1{%
1365   \chardef\bbl@activated\tw@
1366   \bbl@withactive{\expandafter\let\expandafter}#1%
1367   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1368 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1369 \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.

```

1370 \def\babel@texpdf#1#2#3#4{%
1371   \ifx\texorpdfstring\undefined
1372     \textormath{#1}{#3}%
1373   \else
1374     \texorpdfstring{\textormath{#1}{#3}}{#2}%

```

```

1375 % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1376 \fi}
1377 %
1378 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1379 \def\@decl@short#1#2#3\@nil#4{%
1380 \def\bbl@tempa{#3}%
1381 \ifx\bbl@tempa\@empty
1382 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1383 \bbl@ifunset{#1@sh@\string#2@}{}%
1384 {\def\bbl@tempa{#4}%
1385 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1386 \else
1387 \bbl@info
1388 {Redefining #1 shorthand \string#2\\%
1389 in language \CurrentOption}%
1390 \fi}%
1391 \@namedef{#1@sh@\string#2@}{#4}%
1392 \else
1393 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1394 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1395 {\def\bbl@tempa{#4}%
1396 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1397 \else
1398 \bbl@info
1399 {Redefining #1 shorthand \string#2\string#3\\%
1400 in language \CurrentOption}%
1401 \fi}%
1402 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1403 \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.

```

1404 \def\textormath{%
1405 \ifmmode
1406 \expandafter\@secondoftwo
1407 \else
1408 \expandafter\@firstoftwo
1409 \fi}

```

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

```

1410 \def\user@group{user}
1411 \def\language@group{english} % TODO. I don't like defaults
1412 \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.

```

1413 \def\useshorthands{%
1414 \ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1415 \def\bbl@usesh@s#1{%
1416 \bbl@usesh@x
1417 {\AddBabelHook{babel-sh-\string#1}{afterextras}}{\bbl@activate{#1}}}%
1418 {#1}}
1419 \def\bbl@usesh@x#1#2{%
1420 \bbl@ifshorthand{#2}%
1421 {\def\user@group{user}%

```

```

1422 \initiate@active@char{#2}%
1423 #1%
1424 \bbl@activate{#2}}%
1425 {\bbl@error
1426 {I can't declare a shorthand turned off (\string#2)}
1427 {Sorry, but you can't use shorthands which have been\\%
1428 turned off in the package options}}}

```

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

```

1429 \def\user@language@group{user@\language@group}
1430 \def\bbl@set@user@generic#1#2{%
1431 \bbl@ifunset{user@generic@active#1}%
1432 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1433 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1434 \expandafter\edef\csname#2@sh@#1@\endcsname{%
1435 \expandafter\noexpand\csname normal@char#1\endcsname}%
1436 \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1437 \expandafter\noexpand\csname user@active#1\endcsname}}%
1438 \@empty}
1439 \newcommand\defineshorthand[3][user]{%
1440 \edef\bbl@tempa{\zap@space#1 \@empty}%
1441 \bbl@for\bbl@tempb\bbl@tempa{%
1442 \if*\expandafter\@car\bbl@tempb\@nil
1443 \edef\bbl@tempb{user\expandafter@gobble\bbl@tempb}%
1444 \@expandtwoargs
1445 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1446 \fi
1447 \declare@shorthand{\bbl@tempb}{#2}{#3}}}

```

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

```

1448 \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`.

```

1449 \def\aliasshorthand#1#2{%
1450 \bbl@ifshorthand{#2}%
1451 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1452 \ifx\document\@notprerr
1453 \@notshorthand{#2}%
1454 \else
1455 \initiate@active@char{#2}%
1456 \expandafter\let\csname active@char\string#2\expandafter\endcsname
1457 \csname active@char\string#1\endcsname
1458 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1459 \csname normal@char\string#1\endcsname
1460 \bbl@activate{#2}%
1461 \fi
1462 \fi}%
1463 {\bbl@error
1464 {Cannot declare a shorthand turned off (\string#2)}
1465 {Sorry, but you cannot use shorthands which have been\\%
1466 turned off in the package options}}}

```

\@notshorthand

```
1467 \def\@notshorthand#1{%
1468   \bbl@error{%
1469     The character '\string #1' should be made a shorthand character;\%
1470     add the command \string\usesshorthands\string{#1\string} to
1471     the preamble.\%
1472     I will ignore your instruction}%
1473   {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.

```
1474 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1475 \DeclareRobustCommand*\shorthandoff{%
1476   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1477 \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.

```
1478 \def\bbl@switch@sh#1#2{%
1479   \ifx#2\@nnil\else
1480     \bbl@ifunset{\bbl@active@\string#2}%
1481     {\bbl@error
1482       {I can't switch '\string#2' on or off--not a shorthand}%
1483       {This character is not a shorthand. Maybe you made\%
1484         a typing mistake? I will ignore your instruction.}}%
1485     {\ifcase#1%   off, on, off*
1486       \catcode`#212\relax
1487     \or
1488       \catcode`#2\active
1489       \bbl@ifunset{\bbl@shdef@\string#2}%
1490       {}%
1491       {\bbl@withactive{\expandafter\let\expandafter}#2%
1492         \csname bbl@shdef@\string#2\endcsname
1493         \bbl@csarg\let{\shdef@\string#2}\relax}%
1494       \ifcase\bbl@activated\or
1495         \bbl@activate{#2}%
1496       \else
1497         \bbl@deactivate{#2}%
1498       \fi
1499     \or
1500       \bbl@ifunset{\bbl@shdef@\string#2}%
1501       {\bbl@withactive{\bbl@csarg\let{\shdef@\string#2}}#2}%
1502       {}%
1503       \csname bbl@oricat@\string#2\endcsname
1504       \csname bbl@oridef@\string#2\endcsname
1505       \fi}%
1506   \bbl@afterfi\bbl@switch@sh#1%
1507   \fi}
```

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

```
1508 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1509 \def\bbl@putsh#1{%
1510   \bbl@ifunset{\bbl@active@\string#1}%
```

```

1511      {\bbl@putsh@i#1\@empty\@nnil}%
1512      {\csname bbl@active@string#1\endcsname}}
1513 \def\bbl@putsh@i#1#2\@nnil{%
1514   \csname\language@group @sh@string#1@%
1515     \ifx\@empty#2\else\string#2\fi\endcsname}
1516 \ifx\bbl@opt@shorthands\@nnil\else
1517   \let\bbl@s@initiate@active@char\initiate@active@char
1518   \def\initiate@active@char#1{%
1519     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1520   \let\bbl@s@switch@sh\bbl@switch@sh
1521   \def\bbl@switch@sh#1#2{%
1522     \ifx#2\@nnil\else
1523       \bbl@afterfi
1524       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1525     \fi}
1526   \let\bbl@s@activate\bbl@activate
1527   \def\bbl@activate#1{%
1528     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1529   \let\bbl@s@deactivate\bbl@deactivate
1530   \def\bbl@deactivate#1{%
1531     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1532 \fi

```

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

```

1533 \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.

```

1534 \def\bbl@prim@s{%
1535   \prime\futurelet\@let@token\bbl@pr@m@s}
1536 \def\bbl@if@primes#1#2{%
1537   \ifx#1\@let@token
1538     \expandafter\@firstoftwo
1539   \else\ifx#2\@let@token
1540     \bbl@afterelse\expandafter\@firstoftwo
1541   \else
1542     \bbl@afterfi\expandafter\@secondoftwo
1543   \fi\fi}
1544 \begingroup
1545   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1546   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1547   \lowercase{%
1548     \gdef\bbl@pr@m@s{%
1549       \bbl@if@primes""%
1550       \pr@@@s
1551       {\bbl@if@primes*\^{\pr@@@t\egroup}}}}
1552 \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).

```

1553 \initiate@active@char{~}
1554 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }

```

```
1555 \bbl@activate{~}
```

\OT1dpos 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.

```
1556 \expandafter\def\csname OT1dpos\endcsname{127}
```

```
1557 \expandafter\def\csname T1dpos\endcsname{4}
```

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

```
1558 \ifx\f@encoding\@undefined
```

```
1559 \def\f@encoding{OT1}
```

```
1560 \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.

```
1561 \bbl@trace{Language attributes}
```

```
1562 \newcommand\languageattribute[2]{%
```

```
1563 \def\bbl@tempc{#1}%
```

```
1564 \bbl@fixname\bbl@tempc
```

```
1565 \bbl@iflanguage\bbl@tempc{%
```

```
1566 \bbl@vforeach{#2}{%
```

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

```
1567 \ifx\bbl@known@attrs\@undefined
```

```
1568 \in@false
```

```
1569 \else
```

```
1570 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
```

```
1571 \fi
```

```
1572 \ifin@
```

```
1573 \bbl@warning{%
```

```
1574 You have more than once selected the attribute '##1'\%
```

```
1575 for language #1. Reported}%
```

```
1576 \else
```

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

```
1577 \bbl@exp{%
```

```
1578 \bbl@add@list\bbl@known@attrs{\bbl@tempc-##1}}%
```

```
1579 \edef\bbl@tempa{\bbl@tempc-##1}%
```

```
1580 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
```

```
1581 {\csname\bbl@tempc @attr@##1\endcsname}%
```

```
1582 {\@attrerr{\bbl@tempc}{##1}}%
```

```
1583 \fi}}}
```

```
1584 \@onlypreamble\languageattribute
```

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

```
1585 \newcommand*{\@attrerr}[2]{%
```

```
1586 \bbl@error
```

```
1587 {The attribute #2 is unknown for language #1.}%
```

```
1588 {Your command will be ignored, type <return> to proceed}}
```

`\bbl@declare@ttribute` This command adds the new language/attribute combination to the list of known attributes. Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro `\extras...` for the current language is extended, otherwise the attribute will not work as its code is removed from memory at `\begin{document}`.

```

1589 \def\bbl@declare@ttribute#1#2#3{%
1590   \bbl@xin@{,#2,},{,\BabelModifiers,}%
1591   \ifin@
1592     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1593   \fi
1594   \bbl@add@list\bbl@attributes{#1-#2}%
1595   \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

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

```

1596 \def\bbl@ifattributeset#1#2#3#4{%
1597   \ifx\bbl@known@attribs\@undefined
1598     \in@false
1599   \else
1600     \bbl@xin@{,#1-#2,},{,\bbl@known@attribs,}%
1601   \fi
1602   \ifin@
1603     \bbl@afterelse#3%
1604   \else
1605     \bbl@afterfi#4%
1606   \fi}

```

`\bbl@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.

```

1607 \def\bbl@ifknown@trib#1#2{%
1608   \let\bbl@tempa\@secondoftwo
1609   \bbl@loopx\bbl@tempb{#2}{%
1610     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1611     \ifin@
1612       \let\bbl@tempa\@firstoftwo
1613     \else
1614       \fi}%
1615   \bbl@tempa}

```

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

```

1616 \def\bbl@clear@ttribs{%
1617   \ifx\bbl@attributes\@undefined\else
1618     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1619       \expandafter\bbl@clear@trib\bbl@tempa.
1620     }%
1621     \let\bbl@attributes\@undefined
1622   \fi}
1623 \def\bbl@clear@trib#1-#2.{%
1624   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1625 \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`

```
1626 \bbl@trace{Macros for saving definitions}  
1627 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1628 \newcount\babel@savecnt  
1629 \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.

```
1630 \def\babel@save#1{%  
1631   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax  
1632   \toks@\expandafter{\originalTeX\let#1=}%  
1633   \bbl@exp{%  
1634     \def\originalTeX{\the\toks@<\babel@\number\babel@savecnt>\relax}}%  
1635   \advance\babel@savecnt\@ne}  
1636 \def\babel@savevariable#1{%  
1637   \toks@\expandafter{\originalTeX #1=}%  
1638   \bbl@exp{\def\originalTeX{\the\toks@the#1\relax}}}
```

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

```
1639 \def\bbl@frenchspacing{%  
1640   \ifnum\the\sfcode`\.=\@m  
1641     \let\bbl@nonfrenchspacing\relax  
1642   \else  
1643     \frenchspacing  
1644     \let\bbl@nonfrenchspacing\nonfrenchspacing  
1645   \fi}  
1646 \let\bbl@nonfrenchspacing\nonfrenchspacing  
1647 \let\bbl@elt\relax  
1648 \edef\bbl@fs@chars{%  
1649   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%  
1650   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%  
1651   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}  
1652 \def\bbl@pre@fs{%  
1653   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%  
1654   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%  
1655 \def\bbl@post@fs{%  
1656   \bbl@save@sfcodes  
1657   \edef\bbl@tempa{\bbl@cl{frspc}}%
```

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


```

1658 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1659 \if u\bbl@tempa % do nothing
1660 \else\if n\bbl@tempa % non french
1661 \def\bbl@elt##1##2##3{%
1662 \ifnum\sfcode`##1=##2\relax
1663 \babel@savevariable{\sfcode`##1}%
1664 \sfcode`##1=##3\relax
1665 \fi}%
1666 \bbl@fs@chars
1667 \else\if y\bbl@tempa % french
1668 \def\bbl@elt##1##2##3{%
1669 \ifnum\sfcode`##1=##3\relax
1670 \babel@savevariable{\sfcode`##1}%
1671 \sfcode`##1=##2\relax
1672 \fi}%
1673 \bbl@fs@chars
1674 \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.

```

1675 \bbl@trace{Short tags}
1676 \def\babeltags#1{%
1677 \edef\bbl@tempa{\zap@space#1 \@empty}%
1678 \def\bbl@tempb##1=##2\@@{%
1679 \edef\bbl@tempc{%
1680 \noexpand\newcommand
1681 \expandafter\noexpand\csname ##1\endcsname{%
1682 \noexpand\protect
1683 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1684 \noexpand\newcommand
1685 \expandafter\noexpand\csname text##1\endcsname{%
1686 \noexpand\foreignlanguage{##2}}}}
1687 \bbl@tempc}%
1688 \bbl@for\bbl@tempa\bbl@tempa{%
1689 \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.

```

1690 \bbl@trace{Hyphens}
1691 \@onlypreamble\babelhyphenation
1692 \AtEndOfPackage{%
1693 \newcommand\babelhyphenation[2][\@empty]{%
1694 \ifx\bbl@hyphenation@\relax
1695 \let\bbl@hyphenation@\@empty
1696 \fi
1697 \ifx\bbl@hyphlist\@empty\else
1698 \bbl@warning{%
1699 You must not intermingle \string\selectlanguage\space and\%
1700 \string\babelhyphenation\space or some exceptions will not\%
1701 be taken into account. Reported}%
1702 \fi
1703 \ifx\@empty#1%

```

```

1704 \protected@edef\bb1@hyphenation@{\bb1@hyphenation@\space#2}%
1705 \else
1706 \bb1@vforeach{#1}{%
1707 \def\bb1@tempa{##1}%
1708 \bb1@fixname\bb1@tempa
1709 \bb1@iflanguage\bb1@tempa{%
1710 \bb1@csarg\protected@edef{hyphenation@\bb1@tempa}{%
1711 \bb1@ifunset{bb1@hyphenation@\bb1@tempa}%
1712 {}%
1713 {\csname bb1@hyphenation@\bb1@tempa\endcsname\space}%
1714 #2}}}%
1715 \fi}}

```

`\bb1@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`³².

```

1716 \def\bb1@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1717 \def\bb1@t@one{T1}
1718 \def\allowhyphens{\ifx\cf@encoding\bb1@t@one\else\bb1@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`.

```

1719 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1720 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
1721 \def\bb1@hyphen{%
1722 \@ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i @empty}}
1723 \def\bb1@hyphen@i#1#2{%
1724 \bb1@ifunset{bb1@hy@#1#2@empty}%
1725 {\csname bb1@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1726 {\csname bb1@hy@#1#2@empty\endcsname}}

```

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

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

```

1727 \def\bb1@usehyphen#1{%
1728 \leavevmode
1729 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1730 \nobreak\hskip\z@skip}
1731 \def\bb1@@usehyphen#1{%
1732 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1733 \def\bb1@hyphenchar{%
1734 \ifnum\hyphenchar\font=\m@ne
1735 \babelnullhyphen
1736 \else
1737 \char\hyphenchar\font
1738 \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 `\bb1@hy@nobreak` is redundant.

```

1739 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
1740 \def\bb1@hy@@soft{\bb1@@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}

```

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

```

1741 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1742 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1743 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}}
1744 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1745 \def\bbl@hy@repeat{%
1746   \bbl@usehyphen{%
1747     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1748 \def\bbl@hy@repeat{%
1749   \bbl@usehyphen{%
1750     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1751 \def\bbl@hy@empty{\hskip\z@skip}
1752 \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.

```

1753 \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.

```

1754 \bbl@trace{Multiencoding strings}
1755 \def\bbl@tglobal#1{\global\let#1#1}
1756 \def\bbl@recatcode#1{% TODO. Used only once?
1757   \@tempcnta="7F
1758   \def\bbl@tempa{%
1759     \ifnum\@tempcnta>"FF\else
1760       \catcode\@tempcnta=#1\relax
1761       \advance\@tempcnta\@ne
1762       \expandafter\bbl@tempa
1763     \fi}%
1764   \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).

```

1765 \@ifpackagewith{babel}{nocase}%
1766   {\let\bbl@patchuclc\relax}%
1767   {\def\bbl@patchuclc{%
1768     \global\let\bbl@patchuclc\relax
1769     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}}%
1770   \gdef\bbl@uclc##1{%
1771     \let\bbl@encoded\bbl@encoded@uclc
1772     \bbl@ifunset{\language @bbl@uclc}% and resumes it
1773     {##1}%

```

```

1774      {\let\bbl@tempa##1\relax % Used by LANG@bbl@uc1c
1775      \csname\language @bbl@uc1c\endcsname}%
1776      {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1777      \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1778      \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}}
1779 <<(*More package options)>> ≡
1780 \DeclareOption{nocase}{}
1781 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1782 <<(*More package options)>> ≡
1783 \let\bbl@opt@strings\@nnil % accept strings=value
1784 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1785 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1786 \def\BabelStringsDefault{generic}
1787 <</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.

```

1788 \@onlypreamble\StartBabelCommands
1789 \def\StartBabelCommands{%
1790   \begingroup
1791   \bbl@recatcode{11}%
1792   <<Macros local to BabelCommands>>
1793   \def\bbl@provstring##1##2{%
1794     \providecommand##1{##2}%
1795     \bbl@tglobal##1}%
1796   \global\let\bbl@scafter\@empty
1797   \let\StartBabelCommands\bbl@startcmds
1798   \ifx\BabelLanguages\relax
1799     \let\BabelLanguages\CurrentOption
1800   \fi
1801   \begingroup
1802   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1803   \StartBabelCommands}
1804 \def\bbl@startcmds{%
1805   \ifx\bbl@screset\@nnil\else
1806     \bbl@usehooks{stopcommands}}}%
1807   \fi
1808   \endgroup
1809   \begingroup
1810   \@ifstar
1811     {\ifx\bbl@opt@strings\@nnil
1812       \let\bbl@opt@strings\BabelStringsDefault
1813     \fi
1814     \bbl@startcmds@i}%
1815     \bbl@startcmds@i}
1816 \def\bbl@startcmds@i#1#2{%
1817   \edef\bbl@L{\zap@space#1 \@empty}%
1818   \edef\bbl@G{\zap@space#2 \@empty}%
1819   \bbl@startcmds@ii}
1820 \let\bbl@startcmds\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the

strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1821 \newcommand\bb1@startcmds@ii[1][\@empty]{%
1822   \let\SetString\@gobbletwo
1823   \let\bb1@stringdef\@gobbletwo
1824   \let\AfterBabelCommands\@gobble
1825   \ifx\@empty#1%
1826     \def\bb1@sc@label{generic}%
1827     \def\bb1@encstring##1##2{%
1828       \ProvideTextCommandDefault##1{##2}%
1829       \bb1@tglobal##1%
1830       \expandafter\bb1@tglobal\csname\string?\string##1\endcsname}%
1831     \let\bb1@sctest\in@true
1832   \else
1833     \let\bb1@sc@charset\space % <- zapped below
1834     \let\bb1@sc@fontenc\space % <- " "
1835     \def\bb1@tempa##1=##2\@nil{%
1836       \bb1@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1837     \bb1@vforeach{label=#1}{\bb1@tempa##1\@nil}%
1838     \def\bb1@tempa##1 ##2{% space -> comma
1839       ##1%
1840       \ifx\@empty##2\else\ifx,##1,\else,\fi\bb1@afterfi\bb1@tempa##2\fi}%
1841     \edef\bb1@sc@fontenc{\expandafter\bb1@tempa\bb1@sc@fontenc\@empty}%
1842     \edef\bb1@sc@label{\expandafter\zap@space\bb1@sc@label\@empty}%
1843     \edef\bb1@sc@charset{\expandafter\zap@space\bb1@sc@charset\@empty}%
1844     \def\bb1@encstring##1##2{%
1845       \bb1@foreach\bb1@sc@fontenc{%
1846         \bb1@ifunset{T@###1}%
1847         {}%
1848         {\ProvideTextCommand##1{####1}{##2}%
1849         \bb1@tglobal##1%
1850         \expandafter
1851         \bb1@tglobal\csname####1\string##1\endcsname}}}%
1852     \def\bb1@sctest{%
1853       \bb1@xin@{\bb1@opt@strings,}{,\bb1@sc@label,\bb1@sc@fontenc,}}%
1854   \fi
1855   \ifx\bb1@opt@strings\@nnil % ie, no strings key -> defaults
1856   \else\ifx\bb1@opt@strings\relax % ie, strings=encoded
1857     \let\AfterBabelCommands\bb1@aftercmds
1858     \let\SetString\bb1@setstring
1859     \let\bb1@stringdef\bb1@encstring
1860   \else % ie, strings=value
1861     \bb1@sctest
1862   \ifin@
1863     \let\AfterBabelCommands\bb1@aftercmds
1864     \let\SetString\bb1@setstring
1865     \let\bb1@stringdef\bb1@provstring
1866   \fi\fi\fi
1867   \bb1@scswitch
1868   \ifx\bb1@G\@empty
1869     \def\SetString##1##2{%
1870       \bb1@error{Missing group for string \string##1}%
1871       {You must assign strings to some category, typically\\%
1872       captions or extras, but you set none}}%
1873   \fi

```

```

1874 \ifx\@empty#1%
1875 \bbl@usehooks{defaultcommands}{}%
1876 \else
1877 \@expandtwoargs
1878 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1879 \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).

```

1880 \def\bbl@forlang#1#2{%
1881 \bbl@for#1\bbl@L{%
1882 \bbl@xin@{,#1,}{,\BabelLanguages,}%
1883 \ifin@#2\relax\fi}}
1884 \def\bbl@scswitch{%
1885 \bbl@forlang\bbl@tempa{%
1886 \ifx\bbl@G\@empty\else
1887 \ifx\SetString\@gobbles\else
1888 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1889 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1890 \ifin@\else
1891 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1892 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1893 \fi
1894 \fi
1895 \fi}}
1896 \AtEndOfPackage{%
1897 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1898 \let\bbl@scswitch\relax}
1899 \@onlypreamble\EndBabelCommands
1900 \def\EndBabelCommands{%
1901 \bbl@usehooks{stopcommands}{}%
1902 \endgroup
1903 \endgroup
1904 \bbl@scafter}
1905 \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.

```

1906 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1907 \bbl@forlang\bbl@tempa{%
1908 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1909 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1910 {\bbl@exp{%
1911 \global\bbbl@add<\bbl@G\bbl@tempa>{\bbbl@scset\#1<\bbl@LC>}}}%
1912 }%
1913 \def\BabelString{#2}%
1914 \bbl@usehooks{stringprocess}{}%
1915 \expandafter\bbl@stringdef

```

```
1916 \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`.

```
1917 \ifx\bbl@opt@strings\relax
1918 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1919 \bbl@patchuclc
1920 \let\bbl@encoded\relax
1921 \def\bbl@encoded@uclc#1{%
1922   \@inmathwarn#1%
1923   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1924     \expandafter\ifx\csname ?\string#1\endcsname\relax
1925       \TextSymbolUnavailable#1%
1926     \else
1927       \csname ?\string#1\endcsname
1928     \fi
1929   \else
1930     \csname\cf@encoding\string#1\endcsname
1931   \fi}
1932 \else
1933 \def\bbl@scset#1#2{\def#1{#2}}
1934 \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.

```
1935 <<(*Macros local to BabelCommands)>> ≡
1936 \def\SetStringLoop##1##2{%
1937   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1938   \count@\z@
1939   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1940     \advance\count@\@ne
1941     \toks@\expandafter{\bbl@tempa}%
1942     \bbl@exp{%
1943       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1944       \count@=\the\count@\relax}}}%
1945 <</Macros local to BabelCommands>>
```

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

```
1946 \def\bbl@aftercmds#1{%
1947   \toks@\expandafter{\bbl@scafter#1}%
1948   \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.

```
1949 <<(*Macros local to BabelCommands)>> ≡
1950 \newcommand\SetCase[3][{}]{%
1951   \bbl@patchuclc
1952   \bbl@forlang\bbl@tempa{%
1953     \expandafter\bbl@encstring
1954     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1955     \expandafter\bbl@encstring
1956     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1957     \expandafter\bbl@encstring
1958     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1959 <</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.

```
1960 <<*Macros local to BabelCommands>> ≡
1961 \newcommand\SetHyphenMap[1]{%
1962   \bbl@forlang\bbl@tempa{%
1963     \expandafter\bbl@stringdef
1964     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1965 <</Macros local to BabelCommands>>
```

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

```
1966 \newcommand\BabelLower[2]{% one to one.
1967   \ifnum\lccode#1=#2\else
1968     \babel@savevariable{\lccode#1}%
1969     \lccode#1=#2\relax
1970   \fi}
1971 \newcommand\BabelLowerMM[4]{% many-to-many
1972   \@tempcnta=#1\relax
1973   \@tempcntb=#4\relax
1974   \def\bbl@tempa{%
1975     \ifnum\@tempcnta>#2\else
1976       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1977       \advance\@tempcnta#3\relax
1978       \advance\@tempcntb#3\relax
1979       \expandafter\bbl@tempa
1980     \fi}%
1981   \bbl@tempa}
1982 \newcommand\BabelLowerMO[4]{% many-to-one
1983   \@tempcnta=#1\relax
1984   \def\bbl@tempa{%
1985     \ifnum\@tempcnta>#2\else
1986       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1987       \advance\@tempcnta#3
1988       \expandafter\bbl@tempa
1989     \fi}%
1990   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1991 <<*More package options>> ≡
1992 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1993 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1994 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1995 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1996 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1997 <</More package options>>
```

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

```
1998 \AtEndOfPackage{%
1999   \ifx\bbl@opt@hyphenmap\undefined
2000     \bbl@xin@{,}{\bbl@language@opts}%
2001     \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
2002   \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.

```
2003 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2004   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2005 \def\bbl@setcaption@x#1#2#3{% language caption-name string
```



```

2006 \bbl@trim@def\bbl@tempa{#2}%
2007 \bbl@xin@{.template}{\bbl@tempa}%
2008 \ifin@
2009 \bbl@ini@captions@template{#3}{#1}%
2010 \else
2011 \edef\bbl@tempd{%
2012 \expandafter\expandafter\expandafter
2013 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2014 \bbl@xin@
2015 {\expandafter\string\csname #2name\endcsname}%
2016 {\bbl@tempd}%
2017 \ifin@ % Renew caption
2018 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2019 \ifin@
2020 \bbl@exp{%
2021 \\bbl@ifsamestring{\bbl@tempa}{\language}%
2022 {\bbl@scset\<#2name>\<#1#2name>}%
2023 {}}%
2024 \else % Old way converts to new way
2025 \bbl@ifunset{#1#2name}%
2026 {\bbl@exp{%
2027 \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2028 \\bbl@ifsamestring{\bbl@tempa}{\language}%
2029 {\def\<#2name>{\<#1#2name>}}%
2030 {}}}%
2031 {}}%
2032 \fi
2033 \else
2034 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2035 \ifin@ % New way
2036 \bbl@exp{%
2037 \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}%
2038 \\bbl@ifsamestring{\bbl@tempa}{\language}%
2039 {\bbl@scset\<#2name>\<#1#2name>}%
2040 {}}%
2041 \else % Old way, but defined in the new way
2042 \bbl@exp{%
2043 \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2044 \\bbl@ifsamestring{\bbl@tempa}{\language}%
2045 {\def\<#2name>{\<#1#2name>}}%
2046 {}}%
2047 \fi%
2048 \fi
2049 \@namedef{#1#2name}{#3}%
2050 \toks@\expandafter{\bbl@captionslist}%
2051 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2052 \ifin@\else
2053 \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2054 \bbl@toGLOBAL\bbl@captionslist
2055 \fi
2056 \fi}
2057 % \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.

```
2058 \bbl@trace{Macros related to glyphs}
```

```

2059 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2060   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2061   \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.

```

2062 \def\save@sf@q#1{\leavevmode
2063   \begingroup
2064   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2065   \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.

```

2066 \ProvideTextCommand{\quotedblbase}{OT1}{%
2067   \save@sf@q{\set@low@box{\textquotedblright\}}%
2068   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

2069 \ProvideTextCommandDefault{\quotedblbase}{%
2070   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2071 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2072   \save@sf@q{\set@low@box{\textquoteright\}}%
2073   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

2074 \ProvideTextCommandDefault{\quotesinglbase}{%
2075   \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2076 \ProvideTextCommand{\guillemetleft}{OT1}{%
2077   \ifmmode
2078     \ll
2079   \else
2080     \save@sf@q{\nobreak
2081       \raise.2ex\hbox{\scriptscriptstyle\ll}\bb1@allowhyphens}%
2082     \fi}
2083 \ProvideTextCommand{\guillemetright}{OT1}{%
2084   \ifmmode
2085     \gg
2086   \else
2087     \save@sf@q{\nobreak
2088       \raise.2ex\hbox{\scriptscriptstyle\gg}\bb1@allowhyphens}%
2089     \fi}
2090 \ProvideTextCommand{\guillemotleft}{OT1}{%
2091   \ifmmode
2092     \ll
2093   \else
2094     \save@sf@q{\nobreak

```

```

2095      \raise.2ex\hbox{$\scriptscriptstyle\l1$}\bbl@allowhyphens}%
2096 \fi}
2097 \ProvideTextCommand{\guillemotright}{OT1}{%
2098   \ifmmode
2099     \gg
2100   \else
2101     \save@sf@q{\nobreak
2102       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2103   \fi}

```

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

```

2104 \ProvideTextCommandDefault{\guillemetleft}{%
2105   \UseTextSymbol{OT1}{\guillemetleft}}
2106 \ProvideTextCommandDefault{\guillemetright}{%
2107   \UseTextSymbol{OT1}{\guillemetright}}
2108 \ProvideTextCommandDefault{\guillemotleft}{%
2109   \UseTextSymbol{OT1}{\guillemotleft}}
2110 \ProvideTextCommandDefault{\guillemotright}{%
2111   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2112 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2113   \ifmmode
2114     <%
2115   \else
2116     \save@sf@q{\nobreak
2117       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2118   \fi}
2119 \ProvideTextCommand{\guilsinglright}{OT1}{%
2120   \ifmmode
2121     >%
2122   \else
2123     \save@sf@q{\nobreak
2124       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2125   \fi}

```

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

```

2126 \ProvideTextCommandDefault{\guilsinglleft}{%
2127   \UseTextSymbol{OT1}{\guilsinglleft}}
2128 \ProvideTextCommandDefault{\guilsinglright}{%
2129   \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.

```

2130 \DeclareTextCommand{\ij}{OT1}{%
2131   i\kern-0.02em\bbl@allowhyphens j}
2132 \DeclareTextCommand{\IJ}{OT1}{%
2133   I\kern-0.02em\bbl@allowhyphens J}
2134 \DeclareTextCommand{\ij}{T1}{\char188}
2135 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2136 \ProvideTextCommandDefault{\ij}{%
2137   \UseTextSymbol{OT1}{\ij}}
2138 \ProvideTextCommandDefault{\IJ}{%
2139   \UseTextSymbol{OT1}{\IJ}}

```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, but not in the OT1 encoding by default.
`\DJ` Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```

2140 \def\crrtic@{\hrule height0.1ex width0.3em}
2141 \def\crttic@{\hrule height0.1ex width0.33em}
2142 \def\ddj@{%
2143   \setbox0\hbox{\dj}\dimen@=\ht0
2144   \advance\dimen@1ex
2145   \dimen@.45\dimen@
2146   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2147   \advance\dimen@ii.5ex
2148   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2149 \def\DDJ@{%
2150   \setbox0\hbox{\DJ}\dimen@=.55\ht0
2151   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2152   \advance\dimen@ii.15ex % correction for the dash position
2153   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2154   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2155   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2156 %
2157 \DeclareTextCommand{\dj}{OT1}{\ddj@ \dj}
2158 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ \DJ}

```

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

```

2159 \ProvideTextCommandDefault{\dj}{%
2160   \UseTextSymbol{OT1}{\dj}}
2161 \ProvideTextCommandDefault{\DJ}{%
2162   \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.

```

2163 \DeclareTextCommand{\SS}{OT1}{\SS}
2164 \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`

```

2165 \ProvideTextCommandDefault{\glq}{%
2166   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2167 \ProvideTextCommand{\grq}{T1}{%
2168   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2169 \ProvideTextCommand{\grq}{TU}{%
2170   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2171 \ProvideTextCommand{\grq}{OT1}{%
2172   \save@sf@q{\kern-.0125em
2173     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2174     \kern.07em\relax}}
2175 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

`\glqq` The ‘german’ double quotes.

`\grqq`

```

2176 \ProvideTextCommandDefault{\glqq}{%
2177   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

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

`\flq` The ‘french’ single guillemets.

```
\frq
2187 \ProvideTextCommandDefault{\flq}{%
2188   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2189 \ProvideTextCommandDefault{\frq}{%
2190   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq
2191 \ProvideTextCommandDefault{\flqq}{%
2192   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2193 \ProvideTextCommandDefault{\frqq}{%
2194   \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 default will be `\umlauthigh` (the normal positioning).

`\umlautlow`

```
2195 \def\umlauthigh{%
2196   \def\bbl@umlauta##1{\leavevmode\bgroup%
2197     \expandafter\accent\csname\fontencoding dpos\endcsname
2198     ##1\bbl@allowhyphens\egroup}%
2199   \let\bbl@umlaute\bbl@umlauta}
2200 \def\umlautlow{%
2201   \def\bbl@umlauta{\protect\lower@umlaut}}
2202 \def\umlautelow{%
2203   \def\bbl@umlaute{\protect\lower@umlaut}}
2204 \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.

```
2205 \expandafter\ifx\csname U@D\endcsname\relax
2206   \csname newdimen\endcsname\U@D
2207 \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.

```
2208 \def\lower@umlaut#1{%
```

```

2209 \leavevmode\bgroup
2210 \U@D 1ex%
2211 {\setbox\z@\hbox{%
2212   \expandafter\char\csname\fontencoding dqpos\endcsname}%
2213   \dimen@ -.45ex\advance\dimen@\ht\z@
2214   \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2215   \expandafter\accent\csname\fontencoding dqpos\endcsname
2216   \fontdimen5\font\U@D #1%
2217 \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).

```

2218 \AtBeginDocument{%
2219   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2220   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2221   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2222   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2223   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2224   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2225   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2226   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2227   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2228   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2229   \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.

```

2230 \ifx\l@english\@undefined
2231   \chardef\l@english\z@
2232 \fi
2233 % The following is used to cancel rules in ini files (see Amharic).
2234 \ifx\l@unhyphenated\@undefined
2235   \newlanguage\l@unhyphenated
2236 \fi

```

8.13 Layout

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

```

2237 \bbl@trace{Bidi layout}
2238 \providecommand\IfBabelLayout[3]{#3}%
2239 \newcommand\BabelPatchSection[1]{%
2240   \@ifundefined{#1}{}{%
2241     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2242     \@namedef{#1}{%
2243       \@ifstar{\bbl@presec@#1}{%
2244         {\@dblarg{\bbl@presec@x{#1}}}}}%
2245   \def\bbl@presec@x#1[#2]#3{%
2246     \bbl@exp{%
2247       \\select@language@x{\bbl@main@language}%
2248       \\bbl@cs{sspre@#1}%
2249       \\bbl@cs{ss@#1}%
2250       [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2251       {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2252       \\select@language@x{\language}}}}

```

```

2253 \def\bbl@presec@s#1#2{%
2254   \bbl@exp{%
2255     \select@language@x{\bbl@main@language}%
2256     \bbl@cs{sspre@#1}%
2257     \bbl@cs{ss@#1}*%
2258     {\foreignlanguage{\language}{\unexpanded{#2}}}%
2259     \select@language@x{\language}}%
2260 \IfBabelLayout{sectioning}%
2261   {\BabelPatchSection{part}%
2262    \BabelPatchSection{chapter}%
2263    \BabelPatchSection{section}%
2264    \BabelPatchSection{subsection}%
2265    \BabelPatchSection{subsubsection}%
2266    \BabelPatchSection{paragraph}%
2267    \BabelPatchSection{subparagraph}%
2268    \def\babel@toc#1{%
2269      \select@language@x{\bbl@main@language}}}%
2270 \IfBabelLayout{captions}%
2271   {\BabelPatchSection{caption}}%

```

8.14 Load engine specific macros

```

2272 \bbl@trace{Input engine specific macros}
2273 \ifcase\bbl@engine
2274   \input txtbabel.def
2275 \or
2276   \input luababel.def
2277 \or
2278   \input xebabel.def
2279 \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.

```

2280 \bbl@trace{Creating languages and reading ini files}
2281 \let\bbl@extend@ini@gobble
2282 \newcommand\babelprovide[2][{}]{%
2283   \let\bbl@savelangname\language
2284   \edef\bbl@savelocaleid{\the\localeid}%
2285   % Set name and locale id
2286   \edef\language{#2}%
2287   \bbl@id@assign
2288   % Initialize keys
2289   \let\bbl@KVP@captions\@nil
2290   \let\bbl@KVP@date\@nil
2291   \let\bbl@KVP@import\@nil
2292   \let\bbl@KVP@main\@nil
2293   \let\bbl@KVP@script\@nil
2294   \let\bbl@KVP@language\@nil
2295   \let\bbl@KVP@hyphenrules\@nil
2296   \let\bbl@KVP@linebreaking\@nil
2297   \let\bbl@KVP@justification\@nil
2298   \let\bbl@KVP@mapfont\@nil
2299   \let\bbl@KVP@maparabic\@nil
2300   \let\bbl@KVP@mapdigits\@nil
2301   \let\bbl@KVP@intraspace\@nil
2302   \let\bbl@KVP@intrapenalty\@nil

```

```

2303 \let\bb1@KVP@onchar\@nil
2304 \let\bb1@KVP@transforms\@nil
2305 \global\let\bb1@release@transforms\@empty
2306 \let\bb1@KVP@alph\@nil
2307 \let\bb1@KVP@Alph\@nil
2308 \let\bb1@KVP@labels\@nil
2309 \bb1@csarg\let{KVP@labels*}\@nil
2310 \global\let\bb1@inidata\@empty
2311 \global\let\bb1@extend@ini\@gobble
2312 \gdef\bb1@key@list{;}%
2313 \bb1@forkv{#1}{% TODO - error handling
2314   \in@{/}{##1}%
2315   \ifin@
2316     \global\let\bb1@extend@ini\bb1@extend@ini@aux
2317     \bb1@renewinikey##1\@{##2}%
2318   \else
2319     \bb1@csarg\def{KVP@##1}{##2}%
2320   \fi}%
2321 \chardef\bb1@howloaded=% 0:none; 1:ldf without ini; 2:ini
2322 \bb1@ifunset{date#2}\z@{\bb1@ifunset{bb1@llevel@#2}\@ne\tw@}%
2323 % == init ==
2324 \ifx\bb1@screset\@undefined
2325   \bb1@ldfinit
2326 \fi
2327 % ==
2328 \let\bb1@lbkflag\relax % \@empty = do setup linebreak
2329 \ifcase\bb1@howloaded
2330   \let\bb1@lbkflag\@empty % new
2331 \else
2332   \ifx\bb1@KVP@hyphenrules\@nil\else
2333     \let\bb1@lbkflag\@empty
2334   \fi
2335   \ifx\bb1@KVP@import\@nil\else
2336     \let\bb1@lbkflag\@empty
2337   \fi
2338 \fi
2339 % == import, captions ==
2340 \ifx\bb1@KVP@import\@nil\else
2341   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
2342   {\ifx\bb1@initoload\relax
2343     \begingroup
2344       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
2345       \bb1@input@texini{#2}%
2346     \endgroup
2347   \else
2348     \xdef\bb1@KVP@import{\bb1@initoload}%
2349   \fi}%
2350 {}%
2351 \fi
2352 \ifx\bb1@KVP@captions\@nil
2353   \let\bb1@KVP@captions\bb1@KVP@import
2354 \fi
2355 % ==
2356 \ifx\bb1@KVP@transforms\@nil\else
2357   \bb1@replace\bb1@KVP@transforms{ },}%
2358 \fi
2359 % == Load ini ==
2360 \ifcase\bb1@howloaded
2361   \bb1@provide@new{#2}%

```



```

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

```

```

2421 \directlua{
2422   if Babel.script_blocks['\bbl@cl{sbc}'] then
2423     Babel.loc_to_scr[\the\localeid] =
2424       Babel.script_blocks['\bbl@cl{sbc}']
2425     Babel.locale_props[\the\localeid].lc = \the\localeid\space
2426     Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@\language}\space
2427   end
2428 }%
2429 \fi
2430 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2431 \ifin@
2432   \bbl@ifunset{\bbl@sys@\language}{\bbl@provide@sys@\language}}{}%
2433   \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}{}%
2434   \directlua{
2435     if Babel.script_blocks['\bbl@cl{sbc}'] then
2436       Babel.loc_to_scr[\the\localeid] =
2437         Babel.script_blocks['\bbl@cl{sbc}']
2438     end}%
2439   \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2440     \AtBeginDocument{%
2441       \bbl@patchfont{\bbl@mapselect}}%
2442       {\selectfont}}%
2443     \def\bbl@mapselect{%
2444       \let\bbl@mapselect\relax
2445       \edef\bbl@prefontid{\fontid\font}}%
2446     \def\bbl@mapdir##1{%
2447       {\def\language{##1}%
2448        \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2449        \bbl@switchfont
2450        \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2451          \directlua{
2452            Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2453              ['/\bbl@prefontid'] = \fontid\font\space}%
2454          \fi}}%
2455     \fi
2456     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2457   \fi
2458   % TODO - catch non-valid values
2459 \fi
2460 % == mapfont ==
2461 % For bidi texts, to switch the font based on direction
2462 \ifx\bbl@KVP@mapfont\@nil\else
2463   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2464   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2465     mapfont. Use 'direction'.%
2466     {See the manual for details.}}}%
2467   \bbl@ifunset{\bbl@sys@\language}{\bbl@provide@sys@\language}}{}%
2468   \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}{}%
2469   \ifx\bbl@mapselect\undefined % TODO. See onchar.
2470     \AtBeginDocument{%
2471       \bbl@patchfont{\bbl@mapselect}}%
2472       {\selectfont}}%
2473     \def\bbl@mapselect{%
2474       \let\bbl@mapselect\relax
2475       \edef\bbl@prefontid{\fontid\font}}%
2476     \def\bbl@mapdir##1{%
2477       {\def\language{##1}%
2478        \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2479        \bbl@switchfont

```

```

2480         \directlua{Babel.fontmap
2481         [\the\csname bbl@wdir@##1\endcsname]%
2482         [\bbl@prefontid]=\fontid\font}}}%
2483     \fi
2484     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language\language}}}%
2485 \fi
2486 % == Line breaking: intraspace, intrapenalty ==
2487 % For CJK, East Asian, Southeast Asian, if interspace in ini
2488 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2489     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2490 \fi
2491 \bbl@provide@intraspace
2492 % == Line breaking: CJK quotes ==
2493 \ifcase\bbl@engine\or
2494     \bbl@xin@{/c}{/\bbl@cl{lbrk}}}%
2495     \ifin@
2496         \bbl@ifunset{bbl@quote@\language\language}{}%
2497         {\directlua{
2498             Babel.locale_props[\the\localeid].cjk_quotes = {}
2499             local cs = 'op'
2500             for c in string.utfvalues(
2501                 [[\csname bbl@quote@\language\language\endcsname]]) do
2502                 if Babel.cjk_characters[c].c == 'qu' then
2503                     Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2504                 end
2505                 cs = (cs == 'op') and 'cl' or 'op'
2506             end
2507         }}%
2508     \fi
2509 \fi
2510 % == Line breaking: justification ==
2511 \ifx\bbl@KVP@justification\@nil\else
2512     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2513 \fi
2514 \ifx\bbl@KVP@linebreaking\@nil\else
2515     \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2516     \ifin@
2517         \bbl@csarg\xdef
2518             {lbrk@\language\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2519     \fi
2520 \fi
2521 \bbl@xin@{/e}{/\bbl@cl{lbrk}}}%
2522 \ifin@else\bbl@xin@{/k}{/\bbl@cl{lbrk}}\fi
2523 \ifin@\bbl@arabicjust\fi
2524 % == Line breaking: hyphenate.other.(locale|script) ==
2525 \ifx\bbl@lbkflag\@empty
2526     \bbl@ifunset{bbl@hyotl@\language\language}{}%
2527     {\bbl@csarg\bbl@replace{hyotl@\language\language}{ }{,}%
2528     \bbl@startcommands*\language\language}%
2529     \bbl@csarg\bbl@foreach{hyotl@\language\language}{%
2530     \ifcase\bbl@engine
2531         \ifnum##1<257
2532             \SetHyphenMap{\BabelLower{##1}{##1}}%
2533         \fi
2534     \else
2535         \SetHyphenMap{\BabelLower{##1}{##1}}%
2536     \fi}%
2537     \bbl@endcommands}%
2538 \bbl@ifunset{bbl@hyots@\language\language}{}%

```

```

2539 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2540 \bbl@csarg\bbl@foreach{hyots@language}{%
2541 \ifcase\bbl@engine
2542 \ifnum##1<257
2543 \global\lccode##1=##1\relax
2544 \fi
2545 \else
2546 \global\lccode##1=##1\relax
2547 \fi}}%
2548 \fi
2549 % == Counters: maparabic ==
2550 % Native digits, if provided in ini (TeX level, xe and lua)
2551 \ifcase\bbl@engine\else
2552 \bbl@ifunset{bbl@dgnat@language}{}%
2553 {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
2554 \expandafter\expandafter\expandafter
2555 \bbl@setdigits\csname bbl@dgnat@language\endcsname
2556 \ifx\bbl@KVP@maparabic\@nil\else
2557 \ifx\bbl@latinarabic\@undefined
2558 \expandafter\let\expandafter\@arabic
2559 \csname bbl@counter@language\endcsname
2560 \else % ie, if layout=counters, which redefines \@arabic
2561 \expandafter\let\expandafter\bbl@latinarabic
2562 \csname bbl@counter@language\endcsname
2563 \fi
2564 \fi
2565 \fi}%
2566 \fi
2567 % == Counters: mapdigits ==
2568 % Native digits (lua level).
2569 \ifodd\bbl@engine
2570 \ifx\bbl@KVP@mapdigits\@nil\else
2571 \bbl@ifunset{bbl@dgnat@language}{}%
2572 {\RequirePackage{luatexbase}%
2573 \bbl@activate@preotf
2574 \directlua{
2575 Babel = Babel or {} %%% -> presets in luababel
2576 Babel.digits_mapped = true
2577 Babel.digits = Babel.digits or {}
2578 Babel.digits[\the\localeid] =
2579 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2580 if not Babel.numbers then
2581 function Babel.numbers(head)
2582 local LOCALE = Babel.attr_locale
2583 local GLYPH = node.id'glyph'
2584 local inmath = false
2585 for item in node.traverse(head) do
2586 if not inmath and item.id == GLYPH then
2587 local temp = node.get_attribute(item, LOCALE)
2588 if Babel.digits[temp] then
2589 local chr = item.char
2590 if chr > 47 and chr < 58 then
2591 item.char = Babel.digits[temp][chr-47]
2592 end
2593 end
2594 elseif item.id == node.id'math' then
2595 inmath = (item.subtype == 0)
2596 end
2597 end

```

```

2598         return head
2599     end
2600 end
2601 }}%
2602 \fi
2603 \fi
2604 % == Counters: alph, Alph ==
2605 % What if extras<lang> contains a \babel@save\@alph? It won't be
2606 % restored correctly when exiting the language, so we ignore
2607 % this change with the \bbl@alph@saved trick.
2608 \ifx\bbl@KVP@alph\@nil\else
2609     \bbl@extras@wrap{\bbl@alph@saved}%
2610     {\let\bbl@alph@saved\@alph}%
2611     {\let\@alph\bbl@alph@saved
2612     \babel@save\@alph}%
2613     \bbl@exp{%
2614         \bbl@add\<extras\language>{%
2615             \let\@alph\<bbl@cntr\bbl@KVP@alph @\language>}}}%
2616 \fi
2617 \ifx\bbl@KVP@Alph\@nil\else
2618     \bbl@extras@wrap{\bbl@Alph@saved}%
2619     {\let\bbl@Alph@saved\@Alph}%
2620     {\let\@Alph\bbl@Alph@saved
2621     \babel@save\@Alph}%
2622     \bbl@exp{%
2623         \bbl@add\<extras\language>{%
2624             \let\@Alph\<bbl@cntr\bbl@KVP@Alph @\language>}}}%
2625 \fi
2626 % == require.babel in ini ==
2627 % To load or reload the babel-*.tex, if require.babel in ini
2628 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2629     \bbl@ifunset\bbl@rqtex\@language\{}%
2630     {\expandafter\ifx\csname bbl@rqtex\@language\endcsname\@empty\else
2631         \let\BabelBeforeIni@gobbletwo
2632         \chardef\atcatcode=\catcode`\@
2633         \catcode`\@=11\relax
2634         \bbl@input@texini{\bbl@cs{rqtex\@language}}%
2635         \catcode`\@=\atcatcode
2636         \let\atcatcode\relax
2637         \global\bbl@csarg\let{rqtex\@language}\relax
2638     \fi}%
2639 \fi
2640 % == frenchspacing ==
2641 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2642 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2643 \ifin@
2644     \bbl@extras@wrap{\bbl@pre@fs}%
2645     {\bbl@pre@fs}%
2646     {\bbl@post@fs}%
2647 \fi
2648 % == Release saved transforms ==
2649 \bbl@release@transforms\relax % \relax closes the last item.
2650 % == main ==
2651 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
2652     \let\language\bbl@savelangname
2653     \chardef\localeid\bbl@savelocaleid\relax
2654 \fi}

```

Depending on whether or not the language exists (based on \date<language>), we define two

macros. Remember \bbl@startcommands opens a group.

```

2655 \def\bbl@provide@new#1{%
2656   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2657   \@namedef{extras#1}{}%
2658   \@namedef{noextras#1}{}%
2659   \bbl@startcommands*{#1}{captions}%
2660   \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
2661     \def\bbl@tempb##1{%             elt for \bbl@captionslist
2662       \ifx##1\@empty\else
2663         \bbl@exp{%
2664           \\SetString\\##1{%
2665             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2666           \expandafter\bbl@tempb
2667         \fi}%
2668     \expandafter\bbl@tempb\bbl@captionslist\@empty
2669   \else
2670     \ifx\bbl@initoload\relax
2671       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2672     \else
2673       \bbl@read@ini{\bbl@initoload}2%    % Same
2674     \fi
2675   \fi
2676   \StartBabelCommands*{#1}{date}%
2677   \ifx\bbl@KVP@import\@nil
2678     \bbl@exp{%
2679       \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
2680   \else
2681     \bbl@savetoday
2682     \bbl@savestate
2683   \fi
2684   \bbl@endcommands
2685   \bbl@load@basic{#1}%
2686   % == hyphenmins == (only if new)
2687   \bbl@exp{%
2688     \gdef\<#1hyphenmins>{%
2689       {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2690       {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
2691     % == hyphenrules (also in renew) ==
2692     \bbl@provide@hyphens{#1}%
2693     \ifx\bbl@KVP@main\@nil\else
2694       \expandafter\main@language\expandafter{#1}%
2695     \fi
2696   %
2697   \def\bbl@provide@renew#1{%
2698     \ifx\bbl@KVP@captions\@nil\else
2699       \StartBabelCommands*{#1}{captions}%
2700       \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2701       \EndBabelCommands
2702     \fi
2703     \ifx\bbl@KVP@import\@nil\else
2704       \StartBabelCommands*{#1}{date}%
2705       \bbl@savetoday
2706       \bbl@savestate
2707       \EndBabelCommands
2708     \fi
2709     % == hyphenrules (also in new) ==
2710     \ifx\bbl@lbkflag\@empty
2711       \bbl@provide@hyphens{#1}%

```

2712 \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.)

```

2713 \def\bbl@load@basic#1{%
2714   \ifcase\bbl@howloaded\or\or
2715     \ifcase\csname bbl@llevel@\language\endcsname
2716       \bbl@csarg\let{lname@\language}\relax
2717     \fi
2718   \fi
2719   \bbl@ifunset{bbl@lname@#1}%
2720   {\def\BabelBeforeIni##1##2{%
2721     \begingroup
2722       \let\bbl@ini@captions@aux\@gobbletwo
2723       \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2724       \bbl@read@ini{##1}1%
2725       \ifx\bbl@initoload\relax\endinput\fi
2726     \endgroup}%
2727     \begingroup      % boxed, to avoid extra spaces:
2728     \ifx\bbl@initoload\relax
2729       \bbl@input@texini{##1}%
2730     \else
2731       \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2732     \fi
2733   \endgroup}%
2734   {}}
```

The hyphenrules option is handled with an auxiliary macro.

```

2735 \def\bbl@provide@hyphens#1{%
2736   \let\bbl@tempa\relax
2737   \ifx\bbl@KVP@hyphenrules\@nil\else
2738     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2739     \bbl@foreach\bbl@KVP@hyphenrules{%
2740       \ifx\bbl@tempa\relax      % if not yet found
2741         \bbl@ifsamestring{##1}{+}%
2742         {{\bbl@exp{\addlanguage\<l@##1>}}}%
2743         {}}%
2744         \bbl@ifunset{l@##1}%
2745         {}%
2746         {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
2747       \fi}%
2748   \fi
2749   \ifx\bbl@tempa\relax %      if no opt or no language in opt found
2750     \ifx\bbl@KVP@import\@nil
2751       \ifx\bbl@initoload\relax\else
2752         \bbl@exp{%
2753           \bbl@ifblank{\bbl@cs{hyphr@#1}}%
2754           {}%
2755           {\let\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2756       \fi
2757     \else % if importing
2758       \bbl@exp{%
2759         \bbl@ifblank{\bbl@cs{hyphr@#1}}%
2760         {}%
2761         {\let\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2762     \fi
2763   \fi
2764   \bbl@ifunset{bbl@tempa}%      ie, relax or undefined
```

```

2765 {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
2766 {\bbl@exp{\addialekt\<l@#1>\language}}%
2767 {}}%          so, l@<lang> is ok - nothing to do
2768 {\bbl@exp{\addialekt\<l@#1>\bbl@tempa}}}% found in opt list or ini

```

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

```

2769 \def\bbl@input@texini#1{%
2770 \bbl@bsphack
2771 \bbl@exp{%
2772 \catcode\=\14 \catcode\=0
2773 \catcode\={1 \catcode\}=2
2774 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2775 \catcode\=\the\catcode\relax
2776 \catcode\={the\catcode\relax
2777 \catcode\={the\catcode\relax
2778 \catcode\}=the\catcode\relax}%
2779 \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.

```

2780 \def\bbl@inline#1\bbl@inline{%
2781 \@ifnextchar[\bbl@inisect{\ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2782 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2783 \def\bbl@iniskip#1\@@{%          if starts with ;
2784 \def\bbl@inistore#1=#2\@@{%          full (default)
2785 \bbl@trim@def\bbl@tempa{#1}%
2786 \bbl@trim\toks@{#2}%
2787 \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2788 \ifin@
2789 \bbl@exp{%
2790 \g@addto@macro\bbl@inidata{%
2791 \bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2792 \fi}
2793 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2794 \bbl@trim@def\bbl@tempa{#1}%
2795 \bbl@trim\toks@{#2}%
2796 \bbl@xin@{.identification.}{.\bbl@section.}%
2797 \ifin@
2798 \bbl@exp{\g@addto@macro\bbl@inidata{%
2799 \bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2800 \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.

```

2801 \ifx\bbl@readstream\undefined
2802 \csname newread\endcsname\bbl@readstream
2803 \fi
2804 \def\bbl@read@ini#1#2{%
2805 \global\let\bbl@extend@ini\gobble
2806 \openin\bbl@readstream=babel-#1.ini
2807 \ifeof\bbl@readstream
2808 \bbl@error
2809 {There is no ini file for the requested language\%
2810 (#1). Perhaps you misspelled it or your installation\%

```



```

2811         is not complete.}%
2812     {Fix the name or reinstall babel.}%
2813 \else
2814     % == Store ini data in \bbl@inidata ==
2815     \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
2816     \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2817     \bbl@info{Importing
2818         \ifcase#2font and identification \or basic \fi
2819         data for \language\name\%
2820         from babel-#1.ini. Reported}%
2821     \ifnum#2=\z@
2822         \global\let\bbl@inidata\@empty
2823         \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2824     \fi
2825     \def\bbl@section{identification}%
2826     \bbl@exp{\ \bbl@inistore tag.ini=#1\ \ \ \ \}%
2827     \bbl@inistore load.level=#2\ \ \ \
2828     \loop
2829     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2830         \endlinechar\m@ne
2831         \read\bbl@readstream to \bbl@line
2832         \endlinechar\^^M
2833         \ifx\bbl@line\@empty\else
2834             \expandafter\bbl@iniline\bbl@line\bbl@iniline
2835         \fi
2836     \repeat
2837     % == Process stored data ==
2838     \bbl@csarg\def\lini@language\language\{#1}%
2839     \bbl@read@ini@aux
2840     % == 'Export' data ==
2841     \bbl@ini@exports{#2}%
2842     \global\bbl@csarg\let\inidata@language\language\bbl@inidata
2843     \global\let\bbl@inidata\@empty
2844     \bbl@exp{\ \bbl@add@list\ \bbl@ini@loaded\language}%
2845     \bbl@tglobal\bbl@ini@loaded
2846     \fi}
2847 \def\bbl@read@ini@aux{%
2848     \let\bbl@savestrings\@empty
2849     \let\bbl@savetoday\@empty
2850     \let\bbl@savestate\@empty
2851     \def\bbl@elt##1##2##3{%
2852         \def\bbl@section{##1}%
2853         \in@{=date.}{=##1}% Find a better place
2854         \ifin@
2855             \bbl@ini@calendar{##1}%
2856         \fi
2857         \bbl@ifunset\bbl@inikv@##1}{}%
2858         {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2859     \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.

```

2860 \def\bbl@extend@ini@aux#1{%
2861     \bbl@startcommands*{#1}{captions}%
2862     % Activate captions/... and modify exports
2863     \bbl@csarg\def\inikv@captions.licr{##1##2}%
2864     \setlocalecaption{#1}{##1}{##2}%
2865     \def\bbl@inikv@captions##1##2{%
2866         \bbl@ini@captions@aux{##1}{##2}%

```

```

2867 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2868 \def\bbl@exportkey##1##2##3{%
2869   \bbl@ifunset{\bbl@kv@##2}{}%
2870   {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2871     \bbl@exp{\global\let<\bbl@##1@<\language>\<\bbl@kv@##2>}%
2872     \fi}}%
2873 % As with \bbl@read@ini, but with some changes
2874 \bbl@read@ini@aux
2875 \bbl@ini@exports\tw@
2876 % Update inidata@lang by pretending the ini is read.
2877 \def\bbl@elt##1##2##3{%
2878   \def\bbl@section{##1}%
2879   \bbl@iniline##2=##3\bbl@iniline}%
2880 \csname bbl@inidata@#1\endcsname
2881 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2882 \StartBabelCommands*{#1}{date}% And from the import stuff
2883 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2884 \bbl@savetoday
2885 \bbl@savestate
2886 \bbl@endcommands}

```

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

```

2887 \def\bbl@ini@calendar#1{%
2888   \lowercase{\def\bbl@tempa{=#1=}}%
2889   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2890   \bbl@replace\bbl@tempa{=date.}{}%
2891   \in@{.licr}{#1}%
2892   \ifin@
2893     \ifcase\bbl@engine
2894       \bbl@replace\bbl@tempa{.licr}{}%
2895     \else
2896       \let\bbl@tempa\relax
2897     \fi
2898   \fi
2899   \ifx\bbl@tempa\relax\else
2900     \bbl@replace\bbl@tempa{=}{}%
2901     \bbl@exp{%
2902       \def<\bbl@inikv@#1>####1####2{%
2903         \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
2904     \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).

```

2905 \def\bbl@renewinikey#1/#2\@#3{%
2906   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2907   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2908   \bbl@trim\toks@{#3}% value
2909   \bbl@exp{%
2910     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb}%
2911     \\g@addto@macro\\bbl@inidata{%
2912       \\\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.

```

2913 \def\bbl@exportkey#1#2##3{%
2914   \bbl@ifunset{\bbl@kv@##2}%
2915   {\bbl@csarg\gdef{#1@<\language>}{#3}}%

```

```

2916 {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2917   \bbl@csarg\gdef{#1@\language}\{#3}%
2918   \else
2919     \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@@kv@#2>}%
2920   \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.

```

2921 \def\bbl@iniwarning#1{%
2922   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2923   {\bbl@warning{%
2924     From babel-\bbl@cs{lini@\language}.ini:\\
2925     \bbl@cs{@kv@identification.warning#1}\\
2926     Reported }}}
2927 %
2928 \let\bbl@release@transforms\@empty
2929 %
2930 \def\bbl@ini@exports#1{%
2931   % Identification always exported
2932   \bbl@iniwarning{%
2933     \ifcase\bbl@engine
2934       \bbl@iniwarning{.pdflatex}%
2935     \or
2936       \bbl@iniwarning{.lualatex}%
2937     \or
2938       \bbl@iniwarning{.xelatex}%
2939     \fi%
2940     \bbl@exportkey{llevel}{identification.load.level}{}%
2941     \bbl@exportkey{elname}{identification.name.english}{}%
2942     \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2943       {\csname bbl@elname@\language\endcsname}}%
2944     \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2945     \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2946     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2947     \bbl@exportkey{esname}{identification.script.name}{}%
2948     \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2949       {\csname bbl@esname@\language\endcsname}}%
2950     \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2951     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2952     % Also maps bcp47 -> language
2953     \ifbbl@bcptoname
2954       \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\language}%
2955     \fi
2956     % Conditional
2957     \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
2958       \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2959       \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2960       \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2961       \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2962       \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2963       \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2964       \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2965       \bbl@exportkey{intsp}{typography.intraspaces}{}%
2966       \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2967       \bbl@exportkey{chrng}{characters.ranges}{}%
2968       \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2969       \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2970     \ifnum#1=\tw@ % only (re)new

```

```

2971 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2972 \bbl@tglobal\bbl@savetoday
2973 \bbl@tglobal\bbl@savedate
2974 \bbl@savestrings
2975 \fi
2976 \fi}

```

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

```

2977 \def\bbl@inikv#1#2{%      key=value
2978 \toks@{#2}%              This hides #'s from ini values
2979 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2980 \let\bbl@inikv@identification\bbl@inikv
2981 \let\bbl@inikv@typography\bbl@inikv
2982 \let\bbl@inikv@characters\bbl@inikv
2983 \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’.

```

2984 \def\bbl@inikv@counters#1#2{%
2985 \bbl@ifsamestring{#1}{digits}%
2986 {\bbl@error{The counter name 'digits' is reserved for mapping\\
2987             decimal digits}%
2988 {Use another name.}}%
2989 }%
2990 \def\bbl@tempc{#1}%
2991 \bbl@trim@def{\bbl@tempb*}{#2}%
2992 \in@{.1$}{#1$}%
2993 \ifin@
2994 \bbl@replace\bbl@tempc{.1}{}%
2995 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
2996 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2997 \fi
2998 \in@{.F.}{#1}%
2999 \ifin@else\in@{.S.}{#1}\fi
3000 \ifin@
3001 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3002 \else
3003 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3004 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3005 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3006 \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.

```

3007 \ifcase\bbl@engine
3008 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3009 \bbl@ini@captions@aux{#1}{#2}}
3010 \else
3011 \def\bbl@inikv@captions#1#2{%
3012 \bbl@ini@captions@aux{#1}{#2}}
3013 \fi

```

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

```

3014 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3015 \bbl@replace\bbl@tempa{.template}{}%
3016 \def\bbl@toreplace{#1}{}}%

```

```

3017 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3018 \bbl@replace\bbl@toreplace{[{}]{\csname}%
3019 \bbl@replace\bbl@toreplace{[{}]{\csname the}%
3020 \bbl@replace\bbl@toreplace{[{}]{\name\endcsname{}}}%
3021 \bbl@replace\bbl@toreplace{[{}]{\endcsname{}}}%
3022 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3023 \ifin@
3024 \@nameuse{\bbl@patch\bbl@tempa}%
3025 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3026 \fi
3027 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3028 \ifin@
3029 \toks@\expandafter{\bbl@toreplace}%
3030 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3031 \fi}
3032 \def\bbl@ini@captions@aux#1#2{%
3033 \bbl@trim@def\bbl@tempa{#1}%
3034 \bbl@xin@{.template}{\bbl@tempa}%
3035 \ifin@
3036 \bbl@ini@captions@template{#2}\language name
3037 \else
3038 \bbl@ifblank{#2}%
3039 {\bbl@exp{%
3040 \toks@{\bbl@nocaption{\bbl@tempa}{\language name\bbl@tempa name}}}%
3041 {\bbl@trim\toks@{#2}}}%
3042 \bbl@exp{%
3043 \bbl@add\bbl@savestrings{%
3044 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3045 \toks@\expandafter{\bbl@captionslist}%
3046 \bbl@exp{\in@{\<\bbl@tempa name>{\the\toks@}}}%
3047 \ifin@\else
3048 \bbl@exp{%
3049 \bbl@add\<\bbl@extracaps@\language name>{\<\bbl@tempa name>}%
3050 \bbl@toglobal\<\bbl@extracaps@\language name>}%
3051 \fi
3052 \fi}

```

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

```

3053 \def\bbl@list@the{%
3054 part,chapter,section,subsection,subsubsection,paragraph,%
3055 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3056 table,page,footnote,mpfootnote,mpfn}
3057 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3058 \bbl@ifunset{\bbl@map@#1\language name}%
3059 {\@nameuse{#1}}%
3060 {\@nameuse{\bbl@map@#1\language name}}}
3061 \def\bbl@inikv@labels#1#2{%
3062 \in@{.map}{#1}%
3063 \ifin@
3064 \ifx\bbl@KVP@labels\@nil\else
3065 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3066 \ifin@
3067 \def\bbl@tempc{#1}%
3068 \bbl@replace\bbl@tempc{.map}{}%
3069 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3070 \bbl@exp{%
3071 \gdef\<\bbl@map@\bbl@tempc @\language name>%
3072 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3073 \bbl@foreach\bbl@list@the{%

```

```

3074 \bbl@ifunset{the##1}{}%
3075 {\bbl@exp{\let\bbl@tempd\<the##1>}%
3076 \bbl@exp{%
3077 \bbl@sreplace\<the##1>%
3078 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3079 \bbl@sreplace\<the##1>%
3080 {\<\empty @\bbl@tempc>\<c##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3081 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3082 \toks@ \expandafter\expandafter\expandafter{%
3083 \csname the##1\endcsname}%
3084 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3085 \fi}}%
3086 \fi
3087 \fi
3088 %
3089 \else
3090 %
3091 % The following code is still under study. You can test it and make
3092 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3093 % language dependent.
3094 \in@{enumerate.}{#1}%
3095 \ifin@
3096 \def\bbl@tempa{#1}%
3097 \bbl@replace\bbl@tempa{enumerate.}{}%
3098 \def\bbl@toreplace{#2}%
3099 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3100 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3101 \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}}%
3102 \toks@ \expandafter{\bbl@toreplace}%
3103 % TODO. Execute only once:
3104 \bbl@exp{%
3105 \bbl@add\<extras\language>{%
3106 \babel@save\<labelenum\romannumeral\bbl@tempa>%
3107 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3108 \bbl@toglobal\<extras\language>}%
3109 \fi
3110 \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.

```

3111 \def\bbl@chapttype{chapter}
3112 \ifx\@makechapterhead\@undefined
3113 \let\bbl@patchchapter\relax
3114 \else\ifx\thechapter\@undefined
3115 \let\bbl@patchchapter\relax
3116 \else\ifx\ps@headings\@undefined
3117 \let\bbl@patchchapter\relax
3118 \else
3119 \def\bbl@patchchapter{%
3120 \global\let\bbl@patchchapter\relax
3121 \gdef\bbl@chfmt{%
3122 \bbl@ifunset{\bbl@chapttype fmt@language}%
3123 {\@chapapp\space\thechapter}
3124 {\@nameuse{\bbl@chapttype fmt@language}}}
3125 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3126 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3127 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%

```

```

3128 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3129 \bbl@tglobal\appendix
3130 \bbl@tglobal\ps@headings
3131 \bbl@tglobal\chaptermark
3132 \bbl@tglobal\@makechapterhead}
3133 \let\bbl@patchappendix\bbl@patchchapter
3134 \fi\fi\fi
3135 \ifx\@part\@undefined
3136 \let\bbl@patchpart\relax
3137 \else
3138 \def\bbl@patchpart{%
3139 \global\let\bbl@patchpart\relax
3140 \gdef\bbl@partformat{%
3141 \bbl@ifunset{\bbl@partfmt@\language}%
3142 {\partname\nobreakspace\thepart}
3143 {\@nameuse{\bbl@partfmt@\language}}}
3144 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3145 \bbl@tglobal\@part}
3146 \fi

```

Date. TODO. Document

```

3147% Arguments are _not_ protected.
3148 \let\bbl@calendar\@empty
3149 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3150 \def\bbl@localedate#1#2#3#4{%
3151 \begingroup
3152 \ifx\@empty#1\@empty\else
3153 \let\bbl@ld@calendar\@empty
3154 \let\bbl@ld@variant\@empty
3155 \edef\bbl@tempa{\zap@space#1 \@empty}%
3156 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld##1}{##2}}%
3157 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3158 \edef\bbl@calendar{%
3159 \bbl@ld@calendar
3160 \ifx\bbl@ld@variant\@empty\else
3161 .\bbl@ld@variant
3162 \fi}%
3163 \bbl@replace\bbl@calendar{\gregorian}{}}%
3164 \fi
3165 \bbl@cased
3166 {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3167 \endgroup}
3168% eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3169 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3170 \bbl@trim@def\bbl@tempa{#1.#2}%
3171 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3172 {\bbl@trim@def\bbl@tempa{#3}%
3173 \bbl@trim\toks@{#5}%
3174 \@temptokena\expandafter{\bbl@savestate}%
3175 \bbl@exp{% Reverse order - in ini last wins
3176 \def\\bbl@savestate{%
3177 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3178 \the\@temptokena}}%
3179 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3180 {\lowercase{\def\bbl@tempb{#6}}%
3181 \bbl@trim@def\bbl@toreplace{#5}%
3182 \bbl@TG@@date
3183 \bbl@ifunset{\bbl@date@\language @}%
3184 {\bbl@exp{% TODO. Move to a better place.

```

```

3185 \gdef\<\language name date>\<\protect\<\language name date >%
3186 \gdef\<\language name date >####1####2####3{%
3187 \<bbl@usedategrouprtrue
3188 \<bbl@ensure@\language name>{%
3189 \<\localedate{####1}{####2}{####3}}}%
3190 \<\bbl@add\<\bbl@savetoday{%
3191 \<\SetString\<\today{%
3192 \<\language name date>%
3193 {\<\the\year}{\<\the\month}{\<\the\day}}}}}%
3194 }}%
3195 \global\<bbl@csarg\let{date@\language name @}\bbl@toreplace
3196 \ifx\<bbl@tempb\@empty\else
3197 \global\<bbl@csarg\let{date@\language name @}\bbl@tempb}\bbl@toreplace
3198 \fi}%
3199 {}}}

```

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).

```

3200 \let\bbl@calendar\@empty
3201 \newcommand\BabelDateSpace{\nobreakspace}
3202 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3203 \newcommand\BabelDated[1]{\number#1}
3204 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3205 \newcommand\BabelDateM[1]{\number#1}
3206 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3207 \newcommand\BabelDateMMMM[1]{%
3208 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3209 \newcommand\BabelDatey[1]{\number#1}%
3210 \newcommand\BabelDateyy[1]{%
3211 \ifnum#1<10 0\number#1 %
3212 \else\ifnum#1<100 \number#1 %
3213 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3214 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3215 \else
3216 \bbl@error
3217 {Currently two-digit years are restricted to the\
3218 range 0-9999.}%
3219 {There is little you can do. Sorry.}%
3220 \fi\fi\fi\fi}}
3221 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3222 \def\bbl@replace@finish@iii#1{%
3223 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3224 \def\bbl@TG@date{%
3225 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3226 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3227 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3228 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3229 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3230 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3231 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3232 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3233 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3234 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3235 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3236 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3237 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%

```



```

3238 \bbl@replace@finish@iii\bbl@toreplace}
3239 \def\bbl@datectr{\expandafter\bbl@xdatectr\expandafter}
3240 \def\bbl@xdatectr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3241 \let\bbl@release@transforms\@empty
3242 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3243 \bbl@transforms\babelprehyphenation}
3244 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3245 \bbl@transforms\babelposthyphenation}
3246 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3247 \begingroup % A hack. TODO. Don't require an specific order
3248 \catcode`\%=12
3249 \catcode`\&=14
3250 \gdef\bbl@transforms#1#2#3{&%
3251 \ifx\bbl@KVP@transforms\@nil\else
3252 \directlua{
3253 str = [==[#2]==]
3254 str = str:gsub('%.%d+%.%d+$', '')
3255 tex.print([[ \def\string\babeltempa{]] .. str .. [[]]])
3256 }&%
3257 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3258 \ifin@
3259 \in@{.0$}{#2$}&%
3260 \ifin@
3261 \g@addto@macro\bbl@release@transforms{&%
3262 \relax\bbl@transforms@aux#1{\language}\{#3}&%
3263 \else
3264 \g@addto@macro\bbl@release@transforms{, {#3}&%
3265 \fi
3266 \fi
3267 \fi}
3268 \endgroup

```

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

```

3269 \def\bbl@provide@lsys#1{%
3270 \bbl@ifunset{bbl@lname@#1}%
3271 {\bbl@load@info{#1}}%
3272 }%
3273 \bbl@csarg\let{lsys@#1}\@empty
3274 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{ }%
3275 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{ }%
3276 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3277 \bbl@ifunset{bbl@lname@#1}{ }%
3278 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3279 \ifcase\bbl@engine\or\or
3280 \bbl@ifunset{bbl@prehc@#1}{ }%
3281 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3282 }%
3283 {\ifx\bbl@xenoxyph\@undefined
3284 \let\bbl@xenoxyph\bbl@xenoxyph@d
3285 \ifx\AtBeginDocument\@notprerr
3286 \expandafter\@secondoftwo % to execute right now
3287 \fi
3288 \AtBeginDocument{%
3289 \bbl@patchfont{\bbl@xenoxyph}%
3290 \expandafter\selectlanguage\expandafter{\language}}%
3291 \fi}}%

```

```

3292 \fi
3293 \bbl@csarg\bbl@toglobal{lsys@#1}}
3294 \def\bbl@xenohyph@d{%
3295 \bbl@ifset{\bbl@prehc@{language}}{
3296   {\ifnum\hyphenchar\font=\defaultshyphenchar
3297     \iffontchar\font\bbl@c1{prehc}\relax
3298     \hyphenchar\font\bbl@c1{prehc}\relax
3299     \else\iffontchar\font"200B
3300       \hyphenchar\font"200B
3301     \else
3302       \bbl@warning
3303       {Neither 0 nor ZERO WIDTH SPACE are available\\%
3304        in the current font, and therefore the hyphen\\%
3305        will be printed. Try changing the fontspec's\\%
3306        'HyphenChar' to another value, but be aware\\%
3307        this setting is not safe (see the manual)}%
3308       \hyphenchar\font\defaultshyphenchar
3309     \fi\fi
3310   \fi}%
3311   {\hyphenchar\font\defaultshyphenchar}}
3312 % \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).

```

3313 \def\bbl@load@info#1{%
3314 \def\BabelBeforeIni##1##2{%
3315   \begingroup
3316   \bbl@read@ini{##1}0%
3317   \endinput          % babel- .tex may contain onlypreamble's
3318   \endgroup}%        boxed, to avoid extra spaces:
3319   {\bbl@input@texini{#1}}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T_EX. Non-digits characters are kept. The first macro is the generic “localized” command.

```

3320 \def\bbl@setdigits#1#2#3#4#5{%
3321 \bbl@exp{%
3322   \def\<language> digits>####1{% ie, \langdigits
3323     \<bbl@digits@language>####1\\@nil}%
3324     \let\<bbl@cntr@digits@language>\<language> digits>%
3325     \def\<language> counter>####1{% ie, \langcounter
3326       \\\expandafter\<bbl@counter@language>%
3327       \\\csname c@####1\endcsname}%
3328     \def\<bbl@counter@language>####1{% ie, \bbl@counter@lang
3329       \\\expandafter\<bbl@digits@language>%
3330       \\\number####1\\@nil}}%
3331 \def\bbl@tempa##1##2##3##4##5{%
3332   \bbl@exp{% Wow, quite a lot of hashes! :-(
3333     \def\<bbl@digits@language>#####1{%
3334       \\\ifx#####1\\@nil          % ie, \bbl@digits@lang
3335       \\\else
3336         \\\ifx0#####1#1%
3337         \\\else\\\ifx1#####1#2%
3338         \\\else\\\ifx2#####1#3%
3339         \\\else\\\ifx3#####1#4%
3340         \\\else\\\ifx4#####1#5%
3341         \\\else\\\ifx5#####1#1%

```

[illegible]

```

3351 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3352   \ifx\\#1%           % \\ before, in case #1 is multiletter
3353     \bbl@exp{%
3354       \def\\bbl@tempa####1{%
3355         \<ifcase>####1\space\the\toks@\<else>\\\<ctrerr\<fi>}}%
3356   \else
3357     \toks@\expandafter{\the\toks@\or #1}%
3358     \expandafter\bbl@buildifcase
3359   \fi}

```

```

3360 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languageame}{#2}}
3361 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3362 \newcommand\localecounter[2]{%
3363   \expandafter\bbl@localecntr
3364   \expandafter{\number\csname c@#2\endcsname}{#1}}
3365 \def\bbl@alphnumeral#1#2{%
3366   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3367 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3368   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
3369     \bbl@alphnumeral@ii{#9}000000#1\or
3370     \bbl@alphnumeral@ii{#9}00000#1#2\or
3371     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3372     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3373     \bbl@alphnum@invalid{>9999}%
3374   \fi}
3375 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3376   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languageame}%
3377   {\bbl@cs{cntr@#1.4@\languageame}{#5%
3378     \bbl@cs{cntr@#1.3@\languageame}{#6%
3379     \bbl@cs{cntr@#1.2@\languageame}{#7%
3380     \bbl@cs{cntr@#1.1@\languageame}{#8%
3381     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3382       \bbl@ifunset{bbl@cntr@#1.S.321@\languageame}{}%
3383       {\bbl@cs{cntr@#1.S.321@\languageame}{}%
3384     \fi}%
3385     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languageame}}}}
3386 \def\bbl@alphnum@invalid#1{%
3387   \bbl@error{Alphabetic numeral too large (#1)}%
3388   {Currently this is the limit.}}

```

```
3389 \newcommand\localeinfo[1]{%
```

```

3390 \bbl@ifunset{\bbl@csname bbl@info@#1\endcsname @\language}%
3391 {\bbl@error{I've found no info for the current locale.\%
3392             The corresponding ini file has not been loaded\%
3393             Perhaps it doesn't exist}%
3394             {See the manual for details.}}%
3395 {\bbl@cs{\csname bbl@info@#1\endcsname @\language}}%
3396 % \@namedef{\bbl@info@name.locale}{lcname}
3397 \@namedef{\bbl@info@tag.ini}{lini}
3398 \@namedef{\bbl@info@name.english}{elname}
3399 \@namedef{\bbl@info@name.opentype}{lname}
3400 \@namedef{\bbl@info@tag.bcp47}{tbc}
3401 \@namedef{\bbl@info@language.tag.bcp47}{lbc}
3402 \@namedef{\bbl@info@tag.opentype}{lotf}
3403 \@namedef{\bbl@info@script.name}{esname}
3404 \@namedef{\bbl@info@script.name.opentype}{sname}
3405 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
3406 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3407 \let\bbl@ensureinfo\@gobble
3408 \newcommand\BabelEnsureInfo{%
3409   \ifx\InputIfFileExists\undefined\else
3410     \def\bbl@ensureinfo##1{%
3411       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}}%
3412   \fi
3413   \bbl@foreach\bbl@loaded{%
3414     \def\language{##1}%
3415     \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`.

```

3416 \newcommand\getlocaleproperty{%
3417   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3418 \def\bbl@getproperty@s#1#2#3{%
3419   \let#1\relax
3420   \def\bbl@elt##1##2##3{%
3421     \bbl@ifsamestring{##1/##2}{##3}%
3422     {\providecommand#1{##3}%
3423     \def\bbl@elt####1####2####3{}}%
3424     {}}%
3425   \bbl@cs{inidata@#2}%
3426   \def\bbl@getproperty@x#1#2#3{%
3427     \bbl@getproperty@s{#1}{#2}{#3}%
3428     \ifx#1\relax
3429       \bbl@error
3430       {Unknown key for locale '#2':\%
3431       #3\%
3432       \string#1 will be set to \relax}%
3433       {Perhaps you misspelled it.}%
3434     \fi}
3435   \let\bbl@ini@loaded\empty
3436   \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

9 Adjusting the Babel bahavior

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

```

3437 \newcommand\babeladjust[1]{% TODO. Error handling.
3438   \bbl@forkv{#1}{%
3439     \bbl@ifunset{\bbl@ADJ@##1@##2}%

```

```

3440      {\bbl@cs{ADJ@##1}{##2}}%
3441      {\bbl@cs{ADJ@##1@##2}}}%
3442 %
3443 \def\bbl@adjust@lua#1#2{%
3444   \ifvmode
3445     \ifnum\currentgrouplevel=\z@
3446       \directlua{ Babel.#2 }%
3447       \expandafter\expandafter\expandafter\@gobble
3448     \fi
3449   \fi
3450   {\bbl@error    % The error is gobbled if everything went ok.
3451     {Currently, #1 related features can be adjusted only\\%
3452       in the main vertical list.}%
3453     {Maybe things change in the future, but this is what it is.}}}
3454 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3455   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3456 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3457   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3458 \@namedef{bbl@ADJ@bidi.text@on}{%
3459   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3460 \@namedef{bbl@ADJ@bidi.text@off}{%
3461   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3462 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3463   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3464 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3465   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3466 %
3467 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3468   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3469 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3470   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3471 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3472   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3473 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3474   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3475 \@namedef{bbl@ADJ@justify.arabic@on}{%
3476   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3477 \@namedef{bbl@ADJ@justify.arabic@off}{%
3478   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3479 %
3480 \def\bbl@adjust@layout#1{%
3481   \ifvmode
3482     #1%
3483     \expandafter\@gobble
3484   \fi
3485   {\bbl@error    % The error is gobbled if everything went ok.
3486     {Currently, layout related features can be adjusted only\\%
3487       in vertical mode.}%
3488     {Maybe things change in the future, but this is what it is.}}}
3489 \@namedef{bbl@ADJ@layout.tabular@on}{%
3490   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3491 \@namedef{bbl@ADJ@layout.tabular@off}{%
3492   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3493 \@namedef{bbl@ADJ@layout.lists@on}{%
3494   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3495 \@namedef{bbl@ADJ@layout.lists@off}{%
3496   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3497 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3498   \bbl@activateposthyphen}

```

```

3499 %
3500 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3501   \bbl@bcpallowedtrue}
3502 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3503   \bbl@bcpallowedfalse}
3504 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3505   \def\bbl@bcp@prefix{#1}}
3506 \def\bbl@bcp@prefix{bcp47-}
3507 \@namedef{bbl@ADJ@autoload.options}#1{%
3508   \def\bbl@autoload@options{#1}}
3509 \let\bbl@autoload@bcptoptions\@empty
3510 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3511   \def\bbl@autoload@bcptoptions{#1}}
3512 \newif\ifbbl@bcptoname
3513 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3514   \bbl@bcptonametrue}
3515   \BabelEnsureInfo}
3516 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3517   \bbl@bcptonamefalse}
3518 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3519   \directlua{ Babel.ignore_pre_char = function(node)
3520     return (node.lang == \the\csname l@nohyphenation\endcsname)
3521   end }}
3522 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3523   \directlua{ Babel.ignore_pre_char = function(node)
3524     return false
3525   end }}
3526 \@namedef{bbl@ADJ@select.write@shift}{%
3527   \let\bbl@restorelastskip\relax
3528   \def\bbl@savelastskip{%
3529     \let\bbl@restorelastskip\relax
3530     \ifvmode
3531       \ifdim\lastskip=\z@
3532         \let\bbl@restorelastskip\nobreak
3533       \else
3534         \bbl@exp{%
3535           \def\\bbl@restorelastskip{%
3536             \skip@=\the\lastskip
3537             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3538         \fi
3539       \fi}}
3540 \@namedef{bbl@ADJ@select.write@keep}{%
3541   \let\bbl@restorelastskip\relax
3542   \let\bbl@savelastskip\relax}
3543 \@namedef{bbl@ADJ@select.write@omit}{%
3544   \let\bbl@restorelastskip\relax
3545   \def\bbl@savelastskip##1\bbl@restorelastskip{}}

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

3546 \ifx\directlua\@undefined\else
3547   \ifx\bbl@luapatterns\@undefined
3548     \input luabel.def
3549   \fi
3550 \fi

Continue with  $\LaTeX$ .

3551 </package | core>
3552 <*package>

```

9.1 Cross referencing macros

The \TeX 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.

```
3553 <<*More package options>> ≡
3554 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3555 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3556 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3557 <</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.

```
3558 \bbl@trace{Cross referencing macros}
3559 \ifx\bbl@opt@safe\@empty\else
3560   \def\@newl@bel#1#2#3{%
3561     {\@safe@activestrue
3562       \bbl@ifunset{#1@#2}%
3563       \relax
3564       {\gdef\@multiplelabels{%
3565         \@latex@warning@no@line{There were multiply-defined labels}}%
3566         \@latex@warning@no@line{Label `#2' multiply defined}}%
3567       \global\@namedef{#1@#2}{#3}}}
```

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

```
3568 \CheckCommand*\@testdef[3]{%
3569   \def\reserved@a{#3}%
3570   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3571   \else
3572     \@tempswatrue
3573   \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.

```
3574 \def\@testdef#1#2#3{% TODO. With @samestring?
3575   \@safe@activestrue
3576   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3577   \def\bbl@tempb{#3}%
3578   \@safe@activesfalse
3579   \ifx\bbl@tempa\relax
3580   \else
3581     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3582   \fi
3583   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3584   \ifx\bbl@tempa\bbl@tempb
3585   \else
3586     \@tempswatrue
```

```
3587 \fi}
3588 \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.

```
3589 \bbl@xin@{R}\bbl@opt@safe
3590 \ifin@
3591 \bbl@redefineroobust\ref#1{%
3592   \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3593 \bbl@redefineroobust\pageref#1{%
3594   \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3595 \else
3596   \let\org@ref\ref
3597   \let\org@pageref\pageref
3598 \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.

```
3599 \bbl@xin@{B}\bbl@opt@safe
3600 \ifin@
3601 \bbl@redefine\@citex[#1]#2{%
3602   \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3603   \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.

```
3604 \AtBeginDocument{%
3605   \@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.)

```
3606   \def\@citex[#1][#2]#3{%
3607     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3608     \org@@citex[#1][#2]{\@tempa}}%
3609   }{}}
```

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

```
3610 \AtBeginDocument{%
3611   \@ifpackageloaded{cite}{%
3612     \def\@citex[#1]#2{%
3613       \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3614     }{}}
```

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

```
3615 \bbl@redefine\nocite#1{%
3616   \@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 \bibtex is needed we define \bibtex in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibtex. This new definition is then activated.

```
3617 \bbl@redefine\bibtex{%
3618   \bbl@cite@choice
3619   \bibtex}
```

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

```
3620 \def\bbl@bibtex#1#2{%
3621   \org@bibtex{#1}{\@safe@activesfalse#2}}
```

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

```
3622 \def\bbl@cite@choice{%
3623   \global\let\bibtex\bbl@bibtex
3624   \@ifpackageloaded{natbib}{\global\let\bibtex\org@bibtex}{}%
3625   \@ifpackageloaded{cite}{\global\let\bibtex\org@bibtex}{}%
3626   \global\let\bbl@cite@choice\relax}
```

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

```
3627 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3628 \bbl@redefine\@bibitem#1{%
3629   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3630 \else
3631   \let\org@nocite\nocite
3632   \let\org@citex\citex
3633   \let\org@bibtex\bibtex
3634   \let\org@bibitem\@bibitem
3635 \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.

```
3636 \bbl@trace{Marks}
3637 \IfBabelLayout{sectioning}
3638   {\ifx\bbl@opt@headfoot\@nnil
3639     \g@addto@macro\resetactivechars{%
3640       \set@typeset@protect
3641       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3642       \let\protect\noexpand
3643       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3644         \edef\thepage{%
3645           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3646       \fi}%
3647   \fi}
3648 {\ifbbl@single\else
3649   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3650     \markright#1{%
```

```

3651      \bbl@ifblank{#1}%
3652      {\org@markright{}}%
3653      {\toks@{#1}}%
3654      \bbl@exp{%
3655          \org@markright{\protect\foreignlanguage{\language}%
3656              {\protect\bbl@restore@actives\the\toks@}}}%

\markboth The definition of \markboth is equivalent to that of \markright, except that we need two token
\@mkboth 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, LATEX stores the definition in an intermediate macro, so it's not necessary anymore,
but it's preserved for older versions.)

3657      \ifx\@mkboth\markboth
3658          \def\bbl@tempc{\let\@mkboth\markboth}
3659      \else
3660          \def\bbl@tempc{
3661              \fi
3662              \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3663              \markboth#1#2{%
3664                  \protected@edef\bbl@tempb##1{%
3665                      \protect\foreignlanguage
3666                      {\language}{\protect\bbl@restore@actives##1}}%
3667                  \bbl@ifblank{#1}%
3668                  {\toks@{}}%
3669                  {\toks@\expandafter{\bbl@tempb{#1}}}%
3670                  \bbl@ifblank{#2}%
3671                  {\@temptokena{}}%
3672                  {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3673                  \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
3674                  \bbl@tempc
3675              \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.

```

3676 \bbl@trace{Preventing clashes with other packages}
3677 \bbl@xin@{R}\bbl@opt@safe
3678 \ifin@
3679 \AtBeginDocument{%
3680     \@ifpackageloaded{ifthen}{%
3681         \bbl@redefine@long\ifthenelse#1#2#3{%

```

```

3682     \let\bbl@temp@pref\pageref
3683     \let\pageref\org@pageref
3684     \let\bbl@temp@ref\ref
3685     \let\ref\org@ref
3686     \@safe@activetrue
3687     \org@ifthenelse{#1}%
3688       {\let\pageref\bbl@temp@pref
3689        \let\ref\bbl@temp@ref
3690        \@safe@activetrue
3691        #2}%
3692       {\let\pageref\bbl@temp@pref
3693        \let\ref\bbl@temp@ref
3694        \@safe@activetrue
3695        #3}%
3696     }%
3697   }{}%
3698 }

```

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`.

```

3699 \AtBeginDocument{%
3700   \ifpackageloaded{varioref}{%
3701     \bbl@redefine\@vpageref#1[#2]#3{%
3702       \@safe@activetrue
3703       \org@@@vpageref{#1}[#2]{#3}%
3704       \@safe@activetrue}%
3705     \bbl@redefine\vrefpagemum#1#2{%
3706       \@safe@activetrue
3707       \org@vrefpagemum{#1}[#2]%
3708       \@safe@activetrue}%

```

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.

```

3709   \expandafter\def\csname Ref \endcsname#1{%
3710     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3711   }{}%
3712 }
3713 \fi

```

9.3.3 hline

`\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.

```

3714 \AtEndOfPackage{%
3715   \AtBeginDocument{%
3716     \ifpackageloaded{hhline}%
3717       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3718        \else
3719          \makeatletter

```

```

3720         \def\@currname{hhline}\input{hhline.sty}\makeatother
3721         \fi}%
3722     {}}}

```

`\substitutefontfamily` Deprecated. Use the tools provides by \TeX . The command `\substitutefontfamily` creates an .fd file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names.

```

3723 \def\substitutefontfamily#1#2#3{%
3724     \lowercase{\immediate\openout15=#1#2.fd\relax}%
3725     \immediate\write15{%
3726         \string\ProvidesFile{#1#2.fd}%
3727         [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3728         \space generated font description file]^^J
3729         \string\DeclareFontFamily{#1}{#2}{^^J
3730         \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3731         \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3732         \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3733         \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3734         \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3735         \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3736         \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3737         \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3738         }%
3739     \closeout15
3740 }
3741 \@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`

```

3742 \bbl@trace{Encoding and fonts}
3743 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3744 \newcommand\BabelNonText{TS1,T3,TS3}
3745 \let\org@TeX\TeX
3746 \let\org@LaTeX\LaTeX
3747 \let\ensureascii\@firstofone
3748 \AtBeginDocument{%
3749     \def\@elt#1{,#1,}%
3750     \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3751     \let\@elt\relax
3752     \let\bbl@tempb\@empty
3753     \def\bbl@tempc{OT1}%
3754     \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3755         \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3756     \bbl@foreach\bbl@tempa{%
3757         \bbl@xin@{#1}{\BabelNonASCII}%
3758         \ifin@
3759             \def\bbl@tempb{#1}% Store last non-ascii
3760         \else\bbl@xin@{#1}{\BabelNonText}% Pass
3761         \ifin@
3762             \def\bbl@tempc{#1}% Store last ascii
3763         \fi

```

```

3764   \fi}%
3765   \ifx\bb1@tempb\@empty\else
3766     \bb1@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3767     \ifin@\else
3768       \edef\bb1@tempc{\cf@encoding}% The default if ascii wins
3769       \fi
3770       \edef\ensureascii#1{%
3771         {\noexpand\fontencoding{\bb1@tempc}\noexpand\selectfont#1}}%
3772       \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3773       \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3774   \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.

```

3775 \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.

```

3776 \AtBeginDocument{%
3777   \ifpackageloaded{fontspec}%
3778     {\xdef\latinencoding{%
3779       \ifx\UTFfencname\@undefined
3780         EU\ifcase\bb1@engine\or2\or1\fi
3781       \else
3782         \UTFfencname
3783       \fi}}%
3784   {\gdef\latinencoding{OT1}}%
3785   \ifx\cf@encoding\bb1@t@one
3786     \xdef\latinencoding{\bb1@t@one}%
3787   \else
3788     \def\@elt#1{,#1,}%
3789     \edef\bb1@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3790     \let\@elt\relax
3791     \bb1@xin@{,T1,}\bb1@tempa
3792     \ifin@
3793       \xdef\latinencoding{\bb1@t@one}%
3794     \fi
3795   \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.

```

3796 \DeclareRobustCommand{\latintext}{%
3797   \fontencoding{\latinencoding}\selectfont
3798   \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.

```

3799 \ifx\@undefined\DeclareTextFontCommand
3800   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3801 \else
3802   \DeclareTextFontCommand{\textlatin}{\latintext}
3803 \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).

```
3804 \bbl@ifformatlater{2021-06-01}%
3805 {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
3806 {\def\bbl@patchfont#1{%
3807     \expandafter\bbl@add\csname selectfont \endcsname{#1}%
3808     \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.

```
3809 \bbl@trace{Loading basic (internal) bidi support}
3810 \ifodd\bbl@engine
3811 \else % TODO. Move to txtbabel
3812     \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3813         \bbl@error
3814         {The bidi method 'basic' is available only in\\%
3815         luatex. I'll continue with 'bidi=default', so\\%
3816         expect wrong results}%
3817         {See the manual for further details.}%
3818     \let\bbl@beforeforeign\leavevmode
3819     \AtEndOfPackage{%
3820         \EnableBabelHook{babel-bidi}%
3821         \bbl@xebidipar}
3822 \fi\fi
3823 \def\bbl@loadxebidi#1{%
3824     \ifx\RTLfootnotetext\@undefined
3825         \AtEndOfPackage{%
3826             \EnableBabelHook{babel-bidi}%
3827             \ifx\fontspec\@undefined
3828                 \bbl@loadfontspec % bidi needs fontspec
3829             \fi
3830             \usepackage#1{bidi}}%
3831     \fi}
3832 \ifnum\bbl@bidimode>200
3833     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3834         \bbl@tentative{bidi=bidi}
3835         \bbl@loadxebidi{}
```

```

3836 \or
3837 \bbl@loadxebidi{[rldocument]}
3838 \or
3839 \bbl@loadxebidi{}
3840 \fi
3841 \fi
3842 \fi
3843 % TODO? Separate:
3844 \ifnum\bbl@bidimode=\@ne
3845 \let\bbl@beforeforeign\leavevmode
3846 \ifodd\bbl@engine
3847 \newattribute\bbl@attr@dir
3848 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3849 \bbl@exp{\output{\bodydir\pagedir\the\output}}
3850 \fi
3851 \AtEndOfPackage{%
3852 \EnableBabelHook{babel-bidi}%
3853 \ifodd\bbl@engine\else
3854 \bbl@xebidipar
3855 \fi}
3856 \fi

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

3857 \bbl@trace{Macros to switch the text direction}
3858 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3859 \def\bbl@rscripts{% TODO. Base on codes ??
3860 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3861 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
3862 Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
3863 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3864 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3865 Old South Arabian,}%
3866 \def\bbl@provide@dirs#1{%
3867 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3868 \ifin@
3869 \global\bbl@csarg\chardef{wdir@#1}\@ne
3870 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3871 \ifin@
3872 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3873 \fi
3874 \else
3875 \global\bbl@csarg\chardef{wdir@#1}\z@
3876 \fi
3877 \ifodd\bbl@engine
3878 \bbl@csarg\ifcase{wdir@#1}%
3879 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3880 \or
3881 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3882 \or
3883 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3884 \fi
3885 \fi}
3886 \def\bbl@switchdir{%
3887 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%
3888 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}}%
3889 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
3890 \def\bbl@setdirs#1{% TODO - math
3891 \ifcase\bbl@select@type % TODO - strictly, not the right test

```

```

3892 \bbl@bodydir{#1}%
3893 \bbl@pardir{#1}%
3894 \fi
3895 \bbl@textdir{#1}}
3896 % TODO. Only if \bbl@bidimode > 0?:
3897 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3898 \DisableBabelHook{babel-bidi}

```

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

```

3899 \ifodd\bbl@engine % luatex=1
3900 \else % pdftex=0, xetex=2
3901 \newcount\bbl@dirlevel
3902 \chardef\bbl@thetextdir\z@
3903 \chardef\bbl@thepardir\z@
3904 \def\bbl@textdir#1{%
3905 \ifcase#1\relax
3906 \chardef\bbl@thetextdir\z@
3907 \bbl@textdir@i\beginL\endL
3908 \else
3909 \chardef\bbl@thetextdir\@ne
3910 \bbl@textdir@i\beginR\endR
3911 \fi}
3912 \def\bbl@textdir@i#1#2{%
3913 \ifhmode
3914 \ifnum\currentgrouplevel>\z@
3915 \ifnum\currentgrouplevel=\bbl@dirlevel
3916 \bbl@error{Multiple bidi settings inside a group}%
3917 {I'll insert a new group, but expect wrong results.}%
3918 \bgroup\aftergroup#2\aftergroup\egroup
3919 \else
3920 \ifcase\currentgroup\or % 0 bottom
3921 \aftergroup#2% 1 simple {}
3922 \or
3923 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3924 \or
3925 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
3926 \or\or\or % vbox vtop align
3927 \or
3928 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3929 \or\or\or\or\or\or % output math disc insert vcent mathchoice
3930 \or
3931 \aftergroup#2% 14 \begingroup
3932 \else
3933 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3934 \fi
3935 \fi
3936 \bbl@dirlevel\currentgrouplevel
3937 \fi
3938 #1%
3939 \fi}
3940 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
3941 \let\bbl@bodydir\@gobble
3942 \let\bbl@pagedir\@gobble
3943 \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).

```

3944 \def\bbl@xebidipar{%

```



```

3945 \let\bbl@xebidipar\relax
3946 \TeXeTstate\@ne
3947 \def\bbl@xeverypar{%
3948   \ifcase\bbl@thepardir
3949     \ifcase\bbl@thetextdir\else\beginR\fi
3950   \else
3951     {\setbox\z@\lastbox\beginR\box\z@}%
3952   \fi}%
3953 \let\bbl@severypar\everypar
3954 \newtoks\everypar
3955 \everypar=\bbl@severypar
3956 \bbl@severypar{\bbl@xeverypar\the\everypar}}
3957 \ifnum\bbl@bidimode>200
3958   \let\bbl@textdir\i@gobbletwo
3959   \let\bbl@xebidipar\@empty
3960   \AddBabelHook{bidi}{foreign}{%
3961     \def\bbl@tempa{\def\BabelText####1}%
3962     \ifcase\bbl@thetextdir
3963       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
3964     \else
3965       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
3966     \fi}
3967   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
3968 \fi
3969 \fi

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

3970 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
3971 \AtBeginDocument{%
3972   \ifx\pdfstringdefDisableCommands\@undefined\else
3973     \ifx\pdfstringdefDisableCommands\relax\else
3974       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
3975     \fi
3976   \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`.

```

3977 \bbl@trace{Local Language Configuration}
3978 \ifx\loadlocalcfg\@undefined
3979   \@ifpackagewith{babel}{noconfigs}%
3980     {\let\loadlocalcfg\@gobble}%
3981   {\def\loadlocalcfg#1{%
3982     \InputIfFileExists{#1.cfg}%
3983     {\typeout{*****^J%
3984               * Local config file #1.cfg used^^J%
3985               *}}}%
3986     \@empty}}
3987 \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).

```

3988 \bbl@trace{Language options}
3989 \let\bbl@afterlang\relax
3990 \let\BabelModifiers\relax
3991 \let\bbl@loaded@empty
3992 \def\bbl@load@language#1{%
3993   \InputIfFileExists{#1.ldf}%
3994   {\edef\bbl@loaded{\CurrentOption
3995     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
3996     \expandafter\let\expandafter\bbl@afterlang
3997       \csname\CurrentOption.ldf-h@k\endcsname
3998     \expandafter\let\expandafter\BabelModifiers
3999       \csname bbl@mod@\CurrentOption\endcsname}%
4000   {\bbl@error{%
4001     Unknown option '\CurrentOption'. Either you misspelled it\\%
4002     or the language definition file \CurrentOption.ldf was not found}}{%
4003     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4004     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4005     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.

```

4006 \def\bbl@try@load@lang#1#2#3{%
4007   \IfFileExists{\CurrentOption.ldf}%
4008   {\bbl@load@language{\CurrentOption}}}%
4009   {#1\bbl@load@language{#2}#3}}
4010 %
4011 \DeclareOption{hebrew}{%
4012   \input{rlbabel.def}}%
4013   \bbl@load@language{hebrew}}
4014 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4015 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4016 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4017 \DeclareOption{polutonikogreek}{%
4018   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4019 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4020 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4021 \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.

```

4022 \ifx\bbl@opt@config@nnil
4023   \@ifpackagewith{babel}{noconfigs}{}%
4024   {\InputIfFileExists{bblopts.cfg}%
4025     {\typeout{*****^J%
4026       * Local config file bblopts.cfg used^^J%
4027       *}}}%
4028     {}}%
4029 \else
4030   \InputIfFileExists{\bbl@opt@config.cfg}%
4031   {\typeout{*****^J%
4032     * Local config file \bbl@opt@config.cfg used^^J%
4033     *}}}%
4034   {\bbl@error{%
4035     Local config file '\bbl@opt@config.cfg' not found}}{%
4036     Perhaps you misspelled it.}}%

```

4037 \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.

```

4038 \let\bbl@tempc\relax
4039 \bbl@foreach\bbl@language@opts{%
4040   \ifcase\bbl@iniflag % Default
4041     \bbl@ifunset{ds@#1}%
4042     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4043   }%
4044   \or % provide=*
4045     \@gobble % case 2 same as 1
4046   \or % provide+=*
4047     \bbl@ifunset{ds@#1}%
4048     {\IfFileExists{#1.ldf}{}%
4049      {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}}%
4050   }%
4051   \bbl@ifunset{ds@#1}%
4052   {\def\bbl@tempc{#1}%
4053    \DeclareOption{#1}{%
4054      \ifnum\bbl@iniflag>\@ne
4055        \bbl@ldfinit
4056        \babelprovide[import]{#1}%
4057        \bbl@afterldf}%
4058      \else
4059        \bbl@load@language{#1}%
4060      \fi}%
4061   }%
4062   \or % provide*=*
4063     \def\bbl@tempc{#1}%
4064     \bbl@ifunset{ds@#1}%
4065     {\DeclareOption{#1}{%
4066       \bbl@ldfinit
4067       \babelprovide[import]{#1}%
4068       \bbl@afterldf}}%
4069     }%
4070   \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.

```

4071 \let\bbl@tempb\@nnil
4072 \let\bbl@clsopstlst\@classoptionslist
4073 \bbl@foreach\@classoptionslist{%
4074   \bbl@ifunset{ds@#1}%
4075   {\IfFileExists{#1.ldf}%
4076    {\def\bbl@tempb{#1}%
4077     \DeclareOption{#1}{%
4078       \ifnum\bbl@iniflag>\@ne
4079         \bbl@ldfinit
4080         \babelprovide[import]{#1}%
4081         \bbl@afterldf}%
4082       \else
4083         \bbl@load@language{#1}%
4084       \fi}%
4085     {\IfFileExists{babel-#1.tex}%
4086      {\def\bbl@tempb{#1}%

```

```

4087 \ifnum\bbl@iniflag>\z@
4088 \DeclareOption{#1}{%
4089 \ifnum\bbl@iniflag>\@ne
4090 \bbl@ldfinit
4091 \babelprovide[import]{#1}%
4092 \bbl@afterldf}%
4093 \fi}%
4094 \fi}%
4095 {}%
4096 {}

```

If a main language has been set, store it for the third pass.

```

4097 \ifnum\bbl@iniflag=\z@\else
4098 \ifx\bbl@opt@main\@nnil
4099 \ifx\bbl@tempc\relax
4100 \let\bbl@opt@main\bbl@tempb
4101 \else
4102 \let\bbl@opt@main\bbl@tempc
4103 \fi
4104 \fi
4105 \fi
4106 \ifx\bbl@opt@main\@nnil\else
4107 \expandafter
4108 \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
4109 \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
4110 \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 \LaTeX processes before):

```

4111 \def\AfterBabelLanguage#1{%
4112 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4113 \DeclareOption*{}
4114 \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.

```

4115 \bbl@trace{Option 'main'}
4116 \ifx\bbl@opt@main\@nnil
4117 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4118 \let\bbl@tempc\@empty
4119 \bbl@for\bbl@tempb\bbl@tempa{%
4120 \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4121 \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4122 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4123 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4124 \ifx\bbl@tempb\bbl@tempc\else
4125 \bbl@warning{%
4126 Last declared language option is '\bbl@tempc',\%
4127 but the last processed one was '\bbl@tempb'.\%
4128 The main language can't be set as both a global\%
4129 and a package option. Use 'main=\bbl@tempc' as\%
4130 option. Reported}%
4131 \fi
4132 \else
4133 \ifodd\bbl@iniflag % case 1,3

```

```

4134 \bbl@ldfinit
4135 \let\CurrentOption\bbl@opt@main
4136 \ifx\bbl@opt@provide\@nnil
4137 \bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}%
4138 \else
4139 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}%
4140 \bbl@xin@{,provide,}{, #1,}%
4141 \ifin@
4142 \def\bbl@opt@provide{#2}%
4143 \bbl@replace\bbl@opt@provide{;}{,}%
4144 \fi}%
4145 \bbl@exp{%
4146 \bbl@babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4147 \fi
4148 \bbl@afterldf{}%
4149 \else % case 0,2
4150 \chardef\bbl@iniflag\z@ % Force ldf
4151 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
4152 \ExecuteOptions{\bbl@opt@main}
4153 \DeclareOption*{}%
4154 \ProcessOptions*
4155 \fi
4156 \fi
4157 \def\AfterBabelLanguage{%
4158 \bbl@error
4159 {Too late for \string\AfterBabelLanguage}%
4160 {Languages have been loaded, so I can do nothing}}

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

4161 \ifx\bbl@main@language\@undefined
4162 \bbl@info{%
4163 You haven't specified a language. I'll use 'nil'%%
4164 as the main language. Reported}
4165 \bbl@load@language{nil}
4166 \fi
4167 \</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

```

4168 \<*kernel>
4169 \let\bbl@onlyswitch\@empty
4170 \input babel.def
4171 \let\bbl@onlyswitch\@undefined
4172 \</kernel>
4173 \<*patterns>

```

11 Loading hyphenation patterns

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

```
4174 <<Make sure ProvidesFile is defined>>
4175 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4176 \xdef\bbl@format{\jobname}
4177 \def\bbl@version{\<<version>>}
4178 \def\bbl@date{\<<date>>}
4179 \ifx\AtBeginDocument\@undefined
4180   \def\@empty{}
4181 \fi
4182 <<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.

```
4183 \def\process@line#1#2 #3 #4 {%
4184   \ifx=#1%
4185     \process@synonym{#2}%
4186   \else
4187     \process@language{#1#2}{#3}{#4}%
4188   \fi
4189   \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.

```
4190 \toks@{}
4191 \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.

```
4192 \def\process@synonym#1{%
4193   \ifnum\last@language=\m@ne
4194     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4195   \else
4196     \expandafter\chardef\csname l@#1\endcsname\last@language
4197     \wlog{\string\l@#1=\string\language\the\last@language}%
4198     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4199       \csname\language\hyphenmins\endcsname
4200     \let\bbl@elt\relax
4201     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4202   \fi}
```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting. Some pattern files contain changes to the `\lccode` 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{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

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

```

4203 \def\process@language#1#2#3{%
4204   \expandafter\addlanguage\csname l@#1\endcsname
4205   \expandafter\language\csname l@#1\endcsname
4206   \edef\language#1{%
4207     \bbl@hook@everylanguage{#1}%
4208     % > luatex
4209     \bbl@get@enc#1::@@@
4210     \begingroup
4211       \lefthyphenmin\m@ne
4212       \bbl@hook@loadpatterns{#2}%
4213       % > luatex
4214       \ifnum\lefthyphenmin=\m@ne
4215       \else
4216         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4217           \the\lefthyphenmin\the\righthyphenmin}%
4218       \fi
4219     \endgroup
4220     \def\bbl@tempa{#3}%
4221     \ifx\bbl@tempa\@empty\else
4222       \bbl@hook@loadexceptions{#3}%
4223       % > luatex
4224     \fi
4225     \let\bbl@elt\relax
4226     \edef\bbl@languages{%
4227       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4228     \ifnum\the\language=\z@
4229       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4230         \set@hyphenmins\tw@\thr@@\relax
4231       \else
4232         \expandafter\expandafter\expandafter\set@hyphenmins
4233         \csname #1hyphenmins\endcsname
4234       \fi
4235       \the\toks@
4236       \toks@{}%
4237     \fi}

```

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

```

4238 \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.

```

4239 \def\bbl@hook@everylanguage#1{
4240 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4241 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4242 \def\bbl@hook@loadkernel#1{%
4243   \def\addlanguage{\csname newlanguage\endcsname}%
4244   \def\adddialect##1##2{%
4245     \global\chardef##1##2\relax
4246     \wlog{\string##1 = a dialect from \string\language##2}}%
4247   \def\iflanguage##1{%
4248     \expandafter\ifx\csname l@##1\endcsname\relax
4249       \@nolanerr{##1}%
4250     \else
4251       \ifnum\csname l@##1\endcsname=\language
4252         \expandafter\expandafter\expandafter\@firstoftwo
4253       \else
4254         \expandafter\expandafter\expandafter\@secondoftwo
4255       \fi
4256     \fi}%
4257   \def\providehyphenmins##1##2{%
4258     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4259       \@namedef{##1hyphenmins}{##2}%
4260     \fi}%
4261   \def\set@hyphenmins##1##2{%
4262     \lefthyphenmin##1\relax
4263     \righthyphenmin##2\relax}%
4264   \def\selectlanguage{%
4265     \errhelp{Selecting a language requires a package supporting it}%
4266     \errmessage{Not loaded}}%
4267   \let\foreignlanguage\selectlanguage
4268   \let\otherlanguage\selectlanguage
4269   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4270   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4271     \def\setlocale{%
4272       \errhelp{Find an armchair, sit down and wait}%
4273       \errmessage{Not yet available}}%
4274     \let\uselocale\setlocale
4275     \let\locale\setlocale
4276     \let\selectlocale\setlocale
4277     \let\localename\setlocale
4278     \let\textlocale\setlocale
4279     \let\textlanguage\setlocale
4280     \let\languagetext\setlocale}
4281 \begingroup
4282 \def\AddBabelHook#1#2{%
4283   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4284     \def\next{\toks1}%
4285   \else
4286     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4287   \fi
4288   \next}
4289 \ifx\directlua\@undefined
4290   \ifx\XeTeXinputencoding\@undefined\else
4291     \input xebabel.def
4292   \fi
4293 \else
4294   \input luababel.def
4295 \fi
4296 \openin1 = babel-\bbl@format.cfg
4297 \ifeof1

```



```

4298 \else
4299 \input babel-\bbl@format.cfg\relax
4300 \fi
4301 \closein1
4302 \endgroup
4303 \bbl@hook@loadkernel{switch.def}

```

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

```

4304 \openin1 = language.dat

```

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

```

4305 \def\language{english}%
4306 \ifeof1
4307 \message{I couldn't find the file language.dat,\space
4308           I will try the file hyphen.tex}
4309 \input hyphen.tex\relax
4310 \chardef\l@english\z@
4311 \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.

```

4312 \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.

```

4313 \loop
4314 \endlinechar\m@ne
4315 \read1 to \bbl@line
4316 \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.

```

4317 \if T\ifeof1F\fi T\relax
4318 \ifx\bbl@line\@empty\else
4319 \edef\bbl@line{\bbl@line\space\space\space}%
4320 \expandafter\process@line\bbl@line\relax
4321 \fi
4322 \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.

```

4323 \begingroup
4324 \def\bbl@elt#1#2#3#4{%
4325 \global\language=#2\relax
4326 \gdef\language{#1}%
4327 \def\bbl@elt##1##2##3##4{}}%
4328 \bbl@languages
4329 \endgroup
4330 \fi
4331 \closein1

```

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

```

4332 \if/\the\toks@\else

```

```

4333 \errhelp{language.dat loads no language, only synonyms}
4334 \errmessage{Orphan language synonym}
4335 \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.

```

4336 \let\bbl@line\@undefined
4337 \let\process@line\@undefined
4338 \let\process@synonym\@undefined
4339 \let\process@language\@undefined
4340 \let\bbl@get@enc\@undefined
4341 \let\bbl@hyph@enc\@undefined
4342 \let\bbl@tempa\@undefined
4343 \let\bbl@hook@loadkernel\@undefined
4344 \let\bbl@hook@everylanguage\@undefined
4345 \let\bbl@hook@loadpatterns\@undefined
4346 \let\bbl@hook@loadexceptions\@undefined
4347 \</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].

```

4348 <<(*More package options)>> ≡
4349 \chardef\bbl@bidimode\z@
4350 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4351 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4352 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4353 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4354 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4355 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4356 <</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.

```

4357 <<(*Font selection)>> ≡
4358 \bbl@trace{Font handling with fontspec}
4359 \ifx\ExplSyntaxOn\@undefined\else
4360   \ExplSyntaxOn
4361   \catcode`\ =10
4362   \def\bbl@loadfontspec{%
4363     \usepackage{fontspec}% TODO. Apply patch always
4364     \expandafter
4365     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4366       Font '\l_fontspec_fontname_tl' is using the\\%
4367       default features for language '##1'.\\%
4368       That's usually fine, because many languages\\%
4369       require no specific features, but if the output is\\%
4370       not as expected, consider selecting another font.}
4371     \expandafter
4372     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4373       Font '\l_fontspec_fontname_tl' is using the\\%

```

```

4374     default features for script '##2'.\\%
4375     That's not always wrong, but if the output is\\%
4376     not as expected, consider selecting another font.}}
4377 \ExplSyntaxOff
4378 \fi
4379 \@onlypreamble\babelfont
4380 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4381 \bbl@foreach{#1}{\%
4382 \expandafter\ifx\csname date##1\endcsname\relax
4383 \IfFileExists{babel-##1.tex}%
4384 {\babelprovide{##1}}%
4385 }%
4386 \fi}%
4387 \edef\bbl@tempa{#1}%
4388 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4389 \ifx\fontspec\undefined
4390 \bbl@loadfontspec
4391 \fi
4392 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4393 \bbl@bblfont}
4394 \newcommand\bbl@bblfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
4395 \bbl@ifunset{\bbl@tempb family}%
4396 {\bbl@providedefam{\bbl@tempb}}%
4397 }%
4398 % For the default font, just in case:
4399 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4400 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4401 {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4402 \bbl@exp{%
4403 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4404 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4405 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4406 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4407 \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:

```

4408 \def\bbl@providedefam#1{%
4409 \bbl@exp{%
4410 \\\newcommand<#1default>{}% Just define it
4411 \\\bbl@add@list\\bbl@font@fams{#1}%
4412 \\\DeclareRobustCommand<#1family>{%
4413 \\\not@math@alphabet<#1family>\relax
4414 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4415 \\\fontfamily<#1default>%
4416 \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}<\fi>%
4417 \\\selectfont}%
4418 \\\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.

```

4419 \def\bbl@nostdfont#1{%
4420 \bbl@ifunset{bbl@WFF@\f@family}%
4421 {\bbl@csarg\gdef{WFF@\f@family}}% Flag, to avoid dupl warns
4422 \bbl@infowarn{The current font is not a babel standard family:\\%
4423 #1%
4424 \fontname\font\\%
4425 There is nothing intrinsically wrong with this warning, and\\%
4426 you can ignore it altogether if you do not need these\\%
4427 families. But if they are used in the document, you should be\\%

```

```

4428     aware 'babel' will no set Script and Language for them, so\\%
4429     you may consider defining a new family with \string\babelfont.\\%
4430     See the manual for further details about \string\babelfont.\\%
4431     Reported}}
4432     {}}%
4433 \gdef\bbl@switchfont{%
4434   \bbl@ifunset{\bbl@sys@\language}\bbl@provide@sys{\language}}{}%
4435   \bbl@exp{%   eg Arabic -> arabic
4436     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4437   \bbl@foreach\bbl@font@fams{%
4438     \bbl@ifunset{\bbl@##1dflt@\language}%      (1) language?
4439     {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}%    (2) from script?
4440     {\bbl@ifunset{\bbl@##1dflt@}%              2=F - (3) from generic?
4441     {}%                                          123=F - nothing!
4442     {\bbl@exp{%                                3=T - from generic
4443       \global\let<\bbl@##1dflt@\language>%
4444       \<\bbl@##1dflt@>}}}%
4445     {\bbl@exp{%                                2=T - from script
4446       \global\let<\bbl@##1dflt@\language>%
4447       \<\bbl@##1dflt@*\bbl@tempa>}}}%
4448     {}%                                          1=T - language, already defined
4449   \def\bbl@tempa{\bbl@nostdfont{}}}%
4450   \bbl@foreach\bbl@font@fams{%   don't gather with prev for
4451     \bbl@ifunset{\bbl@##1dflt@\language}%
4452     {\bbl@cs{famrst@##1}%
4453     \global\bbl@csarg\let{famrst@##1}\relax}%
4454     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4455       \\bbl@add\\originalTeX{%
4456       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4457       \<##1default>\<##1family>{##1}}}%
4458       \\bbl@font@set<\bbl@##1dflt@\language>% the main part!
4459       \<##1default>\<##1family>}}}%
4460   \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.

```

4461 \ifx\f@family\undefined\else   % if latex
4462 \ifcase\bbl@engine              % if pdftex
4463   \let\bbl@ckeckstdfonts\relax
4464 \else
4465   \def\bbl@ckeckstdfonts{%
4466     \begingroup
4467     \global\let\bbl@ckeckstdfonts\relax
4468     \let\bbl@tempa\@empty
4469     \bbl@foreach\bbl@font@fams{%
4470       \bbl@ifunset{\bbl@##1dflt@}%
4471       {\@nameuse{##1family}%
4472       \bbl@csarg\gdef{WFF@f@family}}}% Flag
4473       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4474       \space\space\fontname\font\\}%
4475       \bbl@csarg\xdef{##1dflt@}{\f@family}%
4476       \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4477     {}}%
4478   \ifx\bbl@tempa\@empty\else
4479     \bbl@infowarn{The following font families will use the default\\%
4480       settings for all or some languages:\\%
4481       \bbl@tempa
4482       There is nothing intrinsically wrong with it, but\\%
4483       'babel' will no set Script and Language, which could\\%

```

```

4484         be relevant in some languages. If your document uses\\%
4485         these families, consider redefining them with \string\babelfont.\\%
4486         Reported}%
4487     \fi
4488 \endgroup}
4489 \fi
4490 \fi

```

Now the macros defining the font with fontspec.

When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.

```

4491 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4492   \bbl@xin@{<>}{#1}%
4493   \ifin@
4494     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4495   \fi
4496   \bbl@exp{%
4497     \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4498     \\bbl@ifsamestring{#2}{\f@family}%
4499     {\\#3%
4500       \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}}%
4501     \let\\bbl@tempa\relax}%
4502   {}}
4503 % TODO - next should be global?, but even local does its job. I'm
4504 % still not sure -- must investigate:
4505 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4506   \let\bbl@tempe\bbl@mapselect
4507   \let\bbl@mapselect\relax
4508   \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4509   \let#4\@empty % Make sure \renewfontfamily is valid
4510   \bbl@exp{%
4511     \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4512     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4513     {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4514     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4515     {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4516     \\renewfontfamily\\#4%
4517     [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{..}{#3}
4518   \begingroup
4519     #4%
4520     \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4521   \endgroup
4522   \let#4\bbl@temp@fam
4523   \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4524   \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.

```

4525 \def\bbl@font@rst#1#2#3#4{%
4526   \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.

```

4527 \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 :-).

```

4528 \newcommand\babelFSstore[2][{}]{%

```

```

4529 \bbl@ifblank{#1}%
4530 {\bbl@csarg\def{sname@#2}{Latin}}%
4531 {\bbl@csarg\def{sname@#2}{#1}}%
4532 \bbl@provide@dirs{#2}%
4533 \bbl@csarg\ifnum{wdir@#2}>\z@
4534 \let\bbl@beforeforeign\leavevmode
4535 \EnableBabelHook{babel-bidi}%
4536 \fi
4537 \bbl@foreach{#2}{%
4538 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4539 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4540 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4541 \def\bbl@FSstore#1#2#3#4{%
4542 \bbl@csarg\edef{#2default#1}{#3}%
4543 \expandafter\addto\csname extras#1\endcsname{%
4544 \let#4#3%
4545 \ifx#3\family
4546 \edef#3{\csname bbl@#2default#1\endcsname}%
4547 \fontfamily{#3}\selectfont
4548 \else
4549 \edef#3{\csname bbl@#2default#1\endcsname}%
4550 \fi}%
4551 \expandafter\addto\csname noextras#1\endcsname{%
4552 \ifx#3\family
4553 \fontfamily{#4}\selectfont
4554 \fi
4555 \let#3#4}}
4556 \let\bbl@langfeatures\empty
4557 \def\babelFSfeatures{% make sure \fontspec is redefined once
4558 \let\bbl@ori@fontspec\fontspec
4559 \renewcommand\fontspec[1][{}%
4560 \bbl@ori@fontspec[\bbl@langfeatures##1]}
4561 \let\babelFSfeatures\bbl@FSfeatures
4562 \babelFSfeatures}
4563 \def\bbl@FSfeatures#1#2{%
4564 \expandafter\addto\csname extras#1\endcsname{%
4565 \babel@save\bbl@langfeatures
4566 \edef\bbl@langfeatures{#2,}}
4567 <</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.

```

4568 <<(*Footnote changes)>> ≡
4569 \bbl@trace{Bidi footnotes}
4570 \ifnum\bbl@bidimode>\z@
4571 \def\bbl@footnote#1#2#3{%
4572 \@ifnextchar[%
4573 {\bbl@footnote@o{#1}{#2}{#3}}%
4574 {\bbl@footnote@x{#1}{#2}{#3}}}
4575 \long\def\bbl@footnote@x#1#2#3#4{%
4576 \bgroup
4577 \select@language@x{\bbl@main@language}%
4578 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4579 \egroup}

```

```

4580 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4581   \bgroup
4582     \select@language@x{\bbl@main@language}%
4583     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4584   \egroup}
4585 \def\bbl@footnotetext#1#2#3{%
4586   \@ifnextchar[%
4587     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4588     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4589 \long\def\bbl@footnotetext@x#1#2#3#4{%
4590   \bgroup
4591     \select@language@x{\bbl@main@language}%
4592     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4593   \egroup}
4594 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4595   \bgroup
4596     \select@language@x{\bbl@main@language}%
4597     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4598   \egroup}
4599 \def\BabelFootnote#1#2#3#4{%
4600   \ifx\bbl@fn@footnote\@undefined
4601     \let\bbl@fn@footnote\footnote
4602   \fi
4603   \ifx\bbl@fn@footnotetext\@undefined
4604     \let\bbl@fn@footnotetext\footnotetext
4605   \fi
4606   \bbl@ifblank{#2}%
4607   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4608     \@namedef{\bbl@stripslash#1text}%
4609     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4610   {\def#1{\bbl@exp{\bbl@footnote{\@foreignlanguage{#2}}}{#3}{#4}}%
4611     \@namedef{\bbl@stripslash#1text}%
4612     {\bbl@exp{\bbl@footnotetext{\@foreignlanguage{#2}}}{#3}{#4}}}%
4613 \fi
4614 <</Footnote changes>>

```

Now, the code.

```

4615 (*xetex)
4616 \def\BabelStringsDefault{unicode}
4617 \let\xebbl@stop\relax
4618 \AddBabelHook{xetex}{encodedcommands}{%
4619   \def\bbl@tempa{#1}%
4620   \ifx\bbl@tempa\@empty
4621     \XeTeXinputencoding"bytes"%
4622   \else
4623     \XeTeXinputencoding"#1"%
4624   \fi
4625   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4626 \AddBabelHook{xetex}{stopcommands}{%
4627   \xebbl@stop
4628   \let\xebbl@stop\relax}
4629 \def\bbl@intraspace#1 #2 #3\@@{%
4630   \bbl@csarg\gdef{xeisp@\languagename}%
4631   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4632 \def\bbl@intrapenalty#1\@@{%
4633   \bbl@csarg\gdef{xeipn@\languagename}%
4634   {\XeTeXlinebreakpenalty #1\relax}}
4635 \def\bbl@provide@intraspace{%
4636   \bbl@xin@{/s}{/\bbl@c1{lnbrk}}%

```

```

4637 \ifin@else\bblexin@{/c}/{/bbl@cl{lnbrk}}\fi
4638 \ifin@
4639 \bbl@ifunset{bbl@intsp@language}{}%
4640 {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
4641 \ifx\bbl@KVP@intraspace\@nil
4642 \bbl@exp{%
4643 \bbl@intraspace\bbl@cl{intsp}\@}%
4644 \fi
4645 \ifx\bbl@KVP@intrapenalty\@nil
4646 \bbl@intrapenalty0\@
4647 \fi
4648 \fi
4649 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4650 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
4651 \fi
4652 \ifx\bbl@KVP@intrapenalty\@nil\else
4653 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
4654 \fi
4655 \bbl@exp{%
4656 % TODO. Execute only once (but redundant):
4657 \bbl@add\<extras\language>{%
4658 \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4659 \<bbl@xeisp@language>%
4660 \<bbl@xeipn@language>%
4661 \bbl@toglobal\<extras\language>%
4662 \bbl@add\<noextras\language>{%
4663 \XeTeXlinebreaklocale "en"%
4664 \bbl@toglobal\<noextras\language>}%
4665 \ifx\bbl@ispace\@undefined
4666 \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4667 \ifx\AtBeginDocument\@notprerr
4668 \expandafter\@secondoftwo % to execute right now
4669 \fi
4670 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4671 \fi}%
4672 \fi}
4673 \ifx\DisableBabelHook\@undefined\endinput\fi
4674 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4675 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4676 \DisableBabelHook{babel-fontspec}
4677 <<Font selection>>
4678 \input txtbabel.def
4679 </xetex>

```

13.2 Layout

In progress.

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

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

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

```

4680 <texxet>
4681 \providecommand\bbl@provide@intraspace{}
4682 \bbl@trace{Redefinitions for bidi layout}
4683 \def\bbl@sspre@caption{%
4684 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}

```



```

4685 \ifx\bbbl@opt@layout\@nnil\endinput\fi % No layout
4686 \def\bbbl@startskip{\ifcase\bbbl@thepardir\leftskip\else\rightskip\fi}
4687 \def\bbbl@endskip{\ifcase\bbbl@thepardir\rightskip\else\leftskip\fi}
4688 \ifx\bbbl@beforeforeign\leavevmode % A poor test for bidi=
4689   \def\@hangfrom#1{%
4690     \setbox\@tempboxa\hbox{#1}%
4691     \hangindent\ifcase\bbbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4692     \noindent\box\@tempboxa}
4693 \def\raggedright{%
4694   \let\@centercr
4695   \bbbl@startskip\z@skip
4696   \@rightskip\@flushglue
4697   \bbbl@endskip\@rightskip
4698   \parindent\z@
4699   \parfillskip\bbbl@startskip}
4700 \def\raggedleft{%
4701   \let\@centercr
4702   \bbbl@startskip\@flushglue
4703   \bbbl@endskip\z@skip
4704   \parindent\z@
4705   \parfillskip\bbbl@endskip}
4706 \fi
4707 \IfBabelLayout{lists}
4708   {\bbbl@sreplace\list
4709     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbbl@listleftmargin}%
4710     \def\bbbl@listleftmargin{%
4711       \ifcase\bbbl@thepardir\leftmargin\else\rightmargin\fi}%
4712     \ifcase\bbbl@engine
4713       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4714       \def\p@enumiii{\p@enumii}\theenumii{}\fi
4715     \bbbl@sreplace\@verbatim
4716       {\leftskip\@totalleftmargin}%
4717       {\bbbl@startskip\textwidth
4718         \advance\bbbl@startskip-\linewidth}%
4719     \bbbl@sreplace\@verbatim
4720       {\rightskip\z@skip}%
4721       {\bbbl@endskip\z@skip}}%
4722   {}
4723 {}
4724 \IfBabelLayout{contents}
4725   {\bbbl@sreplace\@dottedtocline{\leftskip}{\bbbl@startskip}%
4726     \bbbl@sreplace\@dottedtocline{\rightskip}{\bbbl@endskip}}
4727 {}
4728 \IfBabelLayout{columns}
4729   {\bbbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbbl@outputbox}%
4730     \def\bbbl@outputbox#1{%
4731       \hb@xt@\textwidth{%
4732         \hskip\columnwidth
4733         \hfil
4734         {\normalcolor\vrule \@width\columnseprule}%
4735         \hfil
4736         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4737         \hskip-\textwidth
4738         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4739         \hskip\columnsep
4740         \hskip\columnwidth}}}%
4741   {}
4742 <<Footnote changes>>
4743 \IfBabelLayout{footnotes}%

```

```

4744 {\BabelFootnote\footnote\language\language{}{}}%
4745 \BabelFootnote\localfootnote\language\language{}{}}%
4746 \BabelFootnote\mainfootnote{}{}}{}{}
4747 {}

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.

4748 \IfBabelLayout{counters}%
4749 {\let\bbl@latinarabic=\@arabic
4750 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4751 \let\bbl@asciroman=\@roman
4752 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4753 \let\bbl@asciiRoman=\@Roman
4754 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4755 \</texet>

```

13.3 LuaTeX

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

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

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

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

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

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

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

This 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).

```

4756 \<!*luatex>
4757 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4758 \bbl@trace{Read language.dat}
4759 \ifx\bbl@readstream\undefined
4760 \csname newread\endcsname\bbl@readstream
4761 \fi
4762 \begingroup
4763 \toks@{}

```

```

4764 \count@ \z@ % 0=start, 1=0th, 2=normal
4765 \def\bb1@process@line#1#2 #3 #4 {%
4766   \ifx=#1%
4767     \bb1@process@synonym{#2}%
4768   \else
4769     \bb1@process@language{#1#2}{#3}{#4}%
4770   \fi
4771   \ignorespaces}
4772 \def\bb1@manylang{%
4773   \ifnum\bb1@last>\@ne
4774     \bb1@info{Non-standard hyphenation setup}%
4775   \fi
4776   \let\bb1@manylang\relax}
4777 \def\bb1@process@language#1#2#3{%
4778   \ifcase\count@
4779     \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4780   \or
4781     \count@\tw@
4782   \fi
4783   \ifnum\count@=\tw@
4784     \expandafter\addlanguage\csname l@#1\endcsname
4785     \language\allocationnumber
4786     \chardef\bb1@last\allocationnumber
4787     \bb1@manylang
4788     \let\bb1@elt\relax
4789     \xdef\bb1@languages{%
4790       \bb1@languages\bb1@elt{#1}{\the\language}{#2}{#3}}%
4791   \fi
4792   \the\toks@
4793   \toks@{}}
4794 \def\bb1@process@synonym@aux#1#2{%
4795   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4796   \let\bb1@elt\relax
4797   \xdef\bb1@languages{%
4798     \bb1@languages\bb1@elt{#1}{#2}{}}}%
4799 \def\bb1@process@synonym#1{%
4800   \ifcase\count@
4801     \toks@\expandafter{\the\toks@\relax\bb1@process@synonym{#1}}%
4802   \or
4803     \ifundefined{zth@#1}{\bb1@process@synonym@aux{#1}{0}}{%
4804   \else
4805     \bb1@process@synonym@aux{#1}{\the\bb1@last}%
4806   \fi}
4807 \ifx\bb1@languages\@undefined % Just a (sensible?) guess
4808   \chardef\l@english\z@
4809   \chardef\l@USenglish\z@
4810   \chardef\bb1@last\z@
4811   \global\@namedef{\bb1@hyphendata@0}{\hyphen.tex}{}
4812   \gdef\bb1@languages{%
4813     \bb1@elt{english}{0}{\hyphen.tex}{}%
4814     \bb1@elt{USenglish}{0}{}}
4815 \else
4816   \global\let\bb1@languages@format\bb1@languages
4817   \def\bb1@elt#1#2#3#4{% Remove all except language 0
4818     \ifnum#2>\z@ \else
4819       \noexpand\bb1@elt{#1}{#2}{#3}{#4}%
4820     \fi}%
4821   \xdef\bb1@languages{\bb1@languages}%
4822 \fi

```

```

4823 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4824 \bbl@languages
4825 \openin\bbl@readstream=language.dat
4826 \ifeof\bbl@readstream
4827 \bbl@warning{I couldn't find language.dat. No additional\\%
4828             patterns loaded. Reported}%
4829 \else
4830 \loop
4831 \endlinechar\m@ne
4832 \read\bbl@readstream to \bbl@line
4833 \endlinechar\^^M
4834 \if T\ifeof\bbl@readstream F\fi T\relax
4835 \ifx\bbl@line\empty\else
4836 \edef\bbl@line{\bbl@line\space\space\space}%
4837 \expandafter\bbl@process@line\bbl@line\relax
4838 \fi
4839 \repeat
4840 \fi
4841 \endgroup
4842 \bbl@trace{Macros for reading patterns files}
4843 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4844 \ifx\babelcatcodetablenum\undefined
4845 \ifx\newcatcodetable\undefined
4846 \def\babelcatcodetablenum{5211}
4847 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4848 \else
4849 \newcatcodetable\babelcatcodetablenum
4850 \newcatcodetable\bbl@pattcodes
4851 \fi
4852 \else
4853 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4854 \fi
4855 \def\bbl@luapatterns#1#2{%
4856 \bbl@get@enc#1::\@@@
4857 \setbox\z@\hbox\bgroup
4858 \begingroup
4859 \savecatcodetable\babelcatcodetablenum\relax
4860 \initcatcodetable\bbl@pattcodes\relax
4861 \catcodetable\bbl@pattcodes\relax
4862 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4863 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\-=13
4864 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4865 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4866 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4867 \catcode\'=12 \catcode\'=12 \catcode\'=12
4868 \input #1\relax
4869 \catcodetable\babelcatcodetablenum\relax
4870 \endgroup
4871 \def\bbl@tempa{#2}%
4872 \ifx\bbl@tempa\empty\else
4873 \input #2\relax
4874 \fi
4875 \egroup}%
4876 \def\bbl@patterns@lua#1{%
4877 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4878 \csname l@#1\endcsname
4879 \edef\bbl@tempa{#1}%
4880 \else
4881 \csname l@#1:\f@encoding\endcsname

```

```

4882 \edef\bbl@tempa{#1:\f@encoding}%
4883 \fi\relax
4884 \@namedef{luatexhyphen@loaded@the\language}{}% Temp
4885 \@ifundefined{bbl@hyphendata@the\language}%
4886 {\def\bbl@elt##1##2##3##4{%
4887 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4888 \def\bbl@tempb{##3}%
4889 \ifx\bbl@tempb\empty\else % if not a synonymous
4890 \def\bbl@tempc{##3}{##4}}%
4891 \fi
4892 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4893 \fi}%
4894 \bbl@languages
4895 \@ifundefined{bbl@hyphendata@the\language}%
4896 {\bbl@info{No hyphenation patterns were set for\%
4897 language '\bbl@tempa'. Reported}}%
4898 {\expandafter\expandafter\expandafter\bbl@luapatterns
4899 \csname bbl@hyphendata@the\language\endcsname}}}%
4900 \endinput\fi
4901 % Here ends \ifx\AddBabelHook\@undefined
4902 % A few lines are only read by hyphen.cfg
4903 \ifx\DisableBabelHook\@undefined
4904 \AddBabelHook{luatex}{everylanguage}{%
4905 \def\process@language##1##2##3{%
4906 \def\process@line####1####2 ####3 ####4 {}}}
4907 \AddBabelHook{luatex}{loadpatterns}{%
4908 \input #1\relax
4909 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4910 {##1}{}}}
4911 \AddBabelHook{luatex}{loadexceptions}{%
4912 \input #1\relax
4913 \def\bbl@tempb##1##2{##1}{##1}}%
4914 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4915 {\expandafter\expandafter\expandafter\bbl@tempb
4916 \csname bbl@hyphendata@the\language\endcsname}}
4917 \endinput\fi
4918 % Here stops reading code for hyphen.cfg
4919 % The following is read the 2nd time it's loaded
4920 \begingroup % TODO - to a lua file
4921 \catcode`\%=12
4922 \catcode`\'=12
4923 \catcode`\%=12
4924 \catcode`\:=12
4925 \directlua{
4926 Babel = Babel or {}
4927 function Babel.bytes(line)
4928 return line:gsub("(.)",
4929 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4930 end
4931 function Babel.begin_process_input()
4932 if luatexbase and luatexbase.add_to_callback then
4933 luatexbase.add_to_callback('process_input_buffer',
4934 Babel.bytes, 'Babel.bytes')
4935 else
4936 Babel.callback = callback.find('process_input_buffer')
4937 callback.register('process_input_buffer', Babel.bytes)
4938 end
4939 end
4940 function Babel.end_process_input ()

```

```

4941   if luatexbase and luatexbase.remove_from_callback then
4942     luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4943   else
4944     callback.register('process_input_buffer',Babel.callback)
4945   end
4946 end
4947 function Babel.addpatterns(pp, lg)
4948   local lg = lang.new(lg)
4949   local pats = lang.patterns(lg) or ''
4950   lang.clear_patterns(lg)
4951   for p in pp:gmatch('[^%s]+') do
4952     ss = ''
4953     for i in string.utfcharacters(p:gsub('%d', '')) do
4954       ss = ss .. '%d?' .. i
4955     end
4956     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4957     ss = ss:gsub('%.%%d%?$', '%%.')
4958     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4959     if n == 0 then
4960       tex.sprint(
4961         [[\string\csname\space bbl@info\endcsname{New pattern: }
4962         .. p .. [{}]]])
4963       pats = pats .. ' ' .. p
4964     else
4965       tex.sprint(
4966         [[\string\csname\space bbl@info\endcsname{Renew pattern: }
4967         .. p .. [{}]]])
4968     end
4969   end
4970   lang.patterns(lg, pats)
4971 end
4972 }
4973 \endgroup
4974 \ifx\newattribute\@undefined\else
4975   \newattribute\bbl@attr@locale
4976   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
4977   \AddBabelHook{luatex}{beforeextras}{%
4978     \setattribute\bbl@attr@locale\localeid}
4979 \fi
4980 \def\BabelStringsDefault{unicode}
4981 \let\luabbl@stop\relax
4982 \AddBabelHook{luatex}{encodedcommands}{%
4983   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4984   \ifx\bbl@tempa\bbl@tempb\else
4985     \directlua{Babel.begin_process_input()}%
4986     \def\luabbl@stop{%
4987       \directlua{Babel.end_process_input()}}%
4988   \fi}%
4989 \AddBabelHook{luatex}{stopcommands}{%
4990   \luabbl@stop
4991   \let\luabbl@stop\relax}
4992 \AddBabelHook{luatex}{patterns}{%
4993   \@ifundefined{bbl@hyphendata@the\language}%
4994     {\def\bbl@elt##1##2##3##4{%
4995       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
4996       \def\bbl@tempb{##3}%
4997       \ifx\bbl@tempb\empty\else % if not a synonymous
4998         \def\bbl@tempc{##3}##4}%
4999       \fi

```

```

5000      \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5001      \fi}%
5002      \bbl@languages
5003      \@ifundefined{bbl@hyphendata@the\language}%
5004      {\bbl@info{No hyphenation patterns were set for\%
5005        language '#2'. Reported}}%
5006      {\expandafter\expandafter\expandafter\bbl@luapatterns
5007        \csname bbl@hyphendata@the\language\endcsname}}}%
5008      \@ifundefined{bbl@patterns@}{}%
5009      \begingroup
5010      \bbl@xin@{\, \number\language,}{, \bbl@pttnlist}%
5011      \ifin@else
5012        \ifx\bbl@patterns@\@empty\else
5013          \directlua{ Babel.addpatterns(
5014            [[\bbl@patterns@]], \number\language) }%
5015          \fi
5016          \@ifundefined{bbl@patterns@#1}%
5017          \@empty
5018          {\directlua{ Babel.addpatterns(
5019            [[\space\csname bbl@patterns@#1\endcsname]],
5020            \number\language) }}%
5021          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5022          \fi
5023        \endgroup}%
5024      \bbl@exp{%
5025        \bbl@ifunset{bbl@prehc@\languagename}{}%
5026        {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5027        {\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.

```

5028 \@onlypreamble\babelpatterns
5029 \AtEndOfPackage{%
5030   \newcommand\babelpatterns[2][\@empty]{%
5031     \ifx\bbl@patterns@\relax
5032       \let\bbl@patterns@\@empty
5033     \fi
5034     \ifx\bbl@pttnlist@\@empty\else
5035       \bbl@warning{%
5036         You must not intermingle \string\selectlanguage\space and\%
5037         \string\babelpatterns\space or some patterns will not\%
5038         be taken into account. Reported}%
5039       \fi
5040       \ifx\@empty#1%
5041         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5042       \else
5043         \edef\bbl@tempb{\zap@space#1 \@empty}%
5044         \bbl@for\bbl@tempa\bbl@tempb{%
5045           \bbl@fixname\bbl@tempa
5046           \bbl@iflanguage\bbl@tempa{%
5047             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5048               \@ifundefined{bbl@patterns@\bbl@tempa}%
5049               \@empty
5050               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5051             #2}}}%
5052         \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.

```
5053% TODO - to a lua file
5054 \directlua{
5055   Babel = Babel or {}
5056   Babel.linebreaking = Babel.linebreaking or {}
5057   Babel.linebreaking.before = {}
5058   Babel.linebreaking.after = {}
5059   Babel.locale = {} % Free to use, indexed by \localeid
5060   function Babel.linebreaking.add_before(func)
5061     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5062     table.insert(Babel.linebreaking.before, func)
5063   end
5064   function Babel.linebreaking.add_after(func)
5065     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5066     table.insert(Babel.linebreaking.after, func)
5067   end
5068 }
5069 \def\bbl@intraspace#1 #2 #3\@@{%
5070   \directlua{
5071     Babel = Babel or {}
5072     Babel.intraspaces = Babel.intraspaces or {}
5073     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5074       {b = #1, p = #2, m = #3}
5075     Babel.locale_props[\the\localeid].intraspace = %
5076       {b = #1, p = #2, m = #3}
5077   }}
5078 \def\bbl@intrapenalty#1\@@{%
5079   \directlua{
5080     Babel = Babel or {}
5081     Babel.intrapenalties = Babel.intrapenalties or {}
5082     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5083     Babel.locale_props[\the\localeid].intrapenalty = #1
5084   }}
5085 \begingroup
5086 \catcode`\%=12
5087 \catcode`\^=14
5088 \catcode`\'=12
5089 \catcode`\~=12
5090 \gdef\bbl@seaintraspace{^
5091   \let\bbl@seaintraspace\relax
5092   \directlua{
5093     Babel = Babel or {}
5094     Babel.sea_enabled = true
5095     Babel.sea_ranges = Babel.sea_ranges or {}
5096     function Babel.set_chranges (script, chrng)
5097       local c = 0
5098       for s, e in string.gmatch(chrng..' ', '(-)%%.(-)%s') do
5099         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5100         c = c + 1
5101       end
5102     end
5103     function Babel.sea_disc_to_space (head)
5104       local sea_ranges = Babel.sea_ranges
5105       local last_char = nil
```



```

5106     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5107     for item in node.traverse(head) do
5108         local i = item.id
5109         if i == node.id'glyph' then
5110             last_char = item
5111         elseif i == 7 and item.subtype == 3 and last_char
5112             and last_char.char > 0x0C99 then
5113             quad = font.getfont(last_char.font).size
5114             for lg, rg in pairs(sea_ranges) do
5115                 if last_char.char > rg[1] and last_char.char < rg[2] then
5116                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5117                     local intraspace = Babel.intraspaces[lg]
5118                     local intrapenalty = Babel.intrapenalties[lg]
5119                     local n
5120                     if intrapenalty ~= 0 then
5121                         n = node.new(14, 0)      ^% penalty
5122                         n.penalty = intrapenalty
5123                         node.insert_before(head, item, n)
5124                     end
5125                     n = node.new(12, 13)        ^% (glue, spaceskip)
5126                     node.setglue(n, intraspace.b * quad,
5127                                   intraspace.p * quad,
5128                                   intraspace.m * quad)
5129                     node.insert_before(head, item, n)
5130                     node.remove(head, item)
5131                 end
5132             end
5133         end
5134     end
5135 end
5136 }^^
5137 \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.

```

5138 \catcode`\%=14
5139 \gdef\bbl@ckintraspace{%
5140     \let\bbl@ckintraspace\relax
5141     \directlua{
5142         Babel = Babel or {}
5143         require('babel-data-cjk.lua')
5144         Babel.cjk_enabled = true
5145         function Babel.cjk_linebreak(head)
5146             local GLYPH = node.id'glyph'
5147             local last_char = nil
5148             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5149             local last_class = nil
5150             local last_lang = nil
5151
5152             for item in node.traverse(head) do
5153                 if item.id == GLYPH then
5154

```

```

5155         local lang = item.lang
5156
5157         local LOCALE = node.get_attribute(item,
5158             Babel.attr_locale)
5159         local props = Babel.locale_props[LOCALE]
5160
5161         local class = Babel.cjk_class[item.char].c
5162
5163         if props.cjk_quotes and props.cjk_quotes[item.char] then
5164             class = props.cjk_quotes[item.char]
5165         end
5166
5167         if class == 'cp' then class = 'cl' end % ]) as CL
5168         if class == 'id' then class = 'I' end
5169
5170         local br = 0
5171         if class and last_class and Babel.cjk_breaks[last_class][class] then
5172             br = Babel.cjk_breaks[last_class][class]
5173         end
5174
5175         if br == 1 and props.linebreak == 'c' and
5176             lang ~= \the\l@nohyphenation\space and
5177             last_lang ~= \the\l@nohyphenation then
5178             local intrapenalty = props.intrapenalty
5179             if intrapenalty ~= 0 then
5180                 local n = node.new(14, 0) % penalty
5181                 n.penalty = intrapenalty
5182                 node.insert_before(head, item, n)
5183             end
5184             local intraspace = props.intraspace
5185             local n = node.new(12, 13) % (glue, spaceskip)
5186             node.setglue(n, intraspace.b * quad,
5187                 intraspace.p * quad,
5188                 intraspace.m * quad)
5189             node.insert_before(head, item, n)
5190         end
5191
5192         if font.getfont(item.font) then
5193             quad = font.getfont(item.font).size
5194         end
5195         last_class = class
5196         last_lang = lang
5197     else % if penalty, glue or anything else
5198         last_class = nil
5199     end
5200 end
5201 lang.hyphenate(head)
5202 end
5203 }%
5204 \bbl@luahyphenate}
5205 \gdef\bbl@luahyphenate{%
5206 \let\bbl@luahyphenate\relax
5207 \directlua{
5208     luatexbase.add_to_callback('hyphenate',
5209     function (head, tail)
5210         if Babel.linebreaking.before then
5211             for k, func in ipairs(Babel.linebreaking.before) do
5212                 func(head)
5213             end

```

```

5214     end
5215     if Babel.cjk_enabled then
5216         Babel.cjk_linebreak(head)
5217     end
5218     lang.hyphenate(head)
5219     if Babel.linebreaking.after then
5220         for k, func in ipairs(Babel.linebreaking.after) do
5221             func(head)
5222         end
5223     end
5224     if Babel.sea_enabled then
5225         Babel.sea_disc_to_space(head)
5226     end
5227 end,
5228 'Babel.hyphenate')
5229 }
5230 }
5231 \endgroup
5232 \def\bbl@provide@intraspace{%
5233   \bbl@ifunset{\bbl@intsp@language\name}{}%
5234   {\expandafter\ifx\csname\bbl@intsp@language\name\endcsname\@empty\else
5235     \bbl@xin@{/c}{/\bbl@cl{lncr}}}%
5236     \ifin@           % cjk
5237     \bbl@cjk@intraspace
5238     \directlua{
5239       Babel = Babel or {}
5240       Babel.locale_props = Babel.locale_props or {}
5241       Babel.locale_props[\the\localeid].linebreak = 'c'
5242     }%
5243     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@%
5244     \ifx\bbl@KVP@intrapenalty\@nil
5245       \bbl@intrapenalty0\@
5246     \fi
5247   \else           % sea
5248     \bbl@sea@intraspace
5249     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@%
5250     \directlua{
5251       Babel = Babel or {}
5252       Babel.sea_ranges = Babel.sea_ranges or {}
5253       Babel.set_chranges('\bbl@cl{sbc}',
5254         '\bbl@cl{chrng}')
5255     }%
5256     \ifx\bbl@KVP@intrapenalty\@nil
5257       \bbl@intrapenalty0\@
5258     \fi
5259   \fi
5260 \fi
5261 \ifx\bbl@KVP@intrapenalty\@nil\else
5262   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5263 \fi}}

```

13.6 Arabic justification

```

5264 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5265 \def\bblar@chars{%
5266   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5267   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5268   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5269 \def\bblar@elongated{%

```

```

5270 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5271 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5272 0649,064A}
5273 \begingroup
5274 \catcode\_ =11 \catcode\`:=11
5275 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5276 \endgroup
5277 \gdef\bbl@arabicjust{%
5278 \let\bbl@arabicjust\relax
5279 \newattribute\bblar@kashida
5280 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5281 \bblar@kashida=\z@
5282 \bbl@patchfont{{\bbl@parsejalt}}}%
5283 \directlua{
5284 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5285 Babel.arabic.elong_map[\the\localeid] = {}
5286 luatexbase.add_to_callback('post_linebreak_filter',
5287 Babel.arabic.justify, 'Babel.arabic.justify')
5288 luatexbase.add_to_callback('hpack_filter',
5289 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5290 }}%
5291 % Save both node lists to make replacement. TODO. Save also widths to
5292 % make computations
5293 \def\bblar@fetchjalt#1#2#3#4{%
5294 \bbl@exp{\bbl@foreach{#1}}}%
5295 \bbl@ifunset\bblar@JE@##1}%
5296 {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5297 {\setbox\z@\hbox{^^^^200d\char"@nameuse\bblar@JE@##1#2}}%
5298 \directlua{%
5299 local last = nil
5300 for item in node.traverse(tex.box[0].head) do
5301 if item.id == node.id'glyph' and item.char > 0x600 and
5302 not (item.char == 0x200D) then
5303 last = item
5304 end
5305 end
5306 Babel.arabic.#3['##1#4'] = last.char
5307 }}}
5308 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5309 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5310 % positioning?
5311 \gdef\bbl@parsejalt{%
5312 \ifx\addfontfeature\undefined\else
5313 \bbl@xin{/e}{/\bbl@c1\lnbrk}}%
5314 \ifin@
5315 \directlua{%
5316 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5317 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5318 tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5319 end
5320 }%
5321 \fi
5322 \fi}
5323 \gdef\bbl@parsejalti{%
5324 \begingroup
5325 \let\bbl@parsejalt\relax % To avoid infinite loop
5326 \edef\bbl@tempb{\fontid\font}%
5327 \bblar@nofswarn
5328 \bblar@fetchjalt\bblar@elongated{{from}}}%

```

```

5329 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5330 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5331 \addfontfeature{RawFeature+=jalt}%
5332 % \namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5333 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5334 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5335 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5336 \directlua{%
5337     for k, v in pairs(Babel.arabic.from) do
5338         if Babel.arabic.dest[k] and
5339             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5340             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5341                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5342         end
5343     end
5344 }%
5345 \endgroup}
5346 %
5347 \begingroup
5348 \catcode`#=11
5349 \catcode`~=11
5350 \directlua{
5351
5352 Babel.arabic = Babel.arabic or {}
5353 Babel.arabic.from = {}
5354 Babel.arabic.dest = {}
5355 Babel.arabic.justify_factor = 0.95
5356 Babel.arabic.justify_enabled = true
5357
5358 function Babel.arabic.justify(head)
5359     if not Babel.arabic.justify_enabled then return head end
5360     for line in node.traverse_id(node.id'hlist', head) do
5361         Babel.arabic.justify_hlist(head, line)
5362     end
5363     return head
5364 end
5365
5366 function Babel.arabic.justify_hbox(head, gc, size, pack)
5367     local has_inf = false
5368     if Babel.arabic.justify_enabled and pack == 'exactly' then
5369         for n in node.traverse_id(12, head) do
5370             if n.stretch_order > 0 then has_inf = true end
5371         end
5372         if not has_inf then
5373             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5374         end
5375     end
5376     return head
5377 end
5378
5379 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5380     local d, new
5381     local k_list, k_item, pos_inline
5382     local width, width_new, full, k_curr, wt_pos, goal, shift
5383     local subst_done = false
5384     local elong_map = Babel.arabic.elong_map
5385     local last_line
5386     local GLYPH = node.id'glyph'
5387     local KASHIDA = Babel.attr_kashida

```

```

5388 local LOCALE = Babel.attr_locale
5389
5390 if line == nil then
5391     line = {}
5392     line.glue_sign = 1
5393     line.glue_order = 0
5394     line.head = head
5395     line.shift = 0
5396     line.width = size
5397 end
5398
5399 % Exclude last line. todo. But-- it discards one-word lines, too!
5400 % ? Look for glue = 12:15
5401 if (line.glue_sign == 1 and line.glue_order == 0) then
5402     elongs = {}      % Stores elongated candidates of each line
5403     k_list = {}      % And all letters with kashida
5404     pos_inline = 0   % Not yet used
5405
5406     for n in node.traverse_id(GLYPH, line.head) do
5407         pos_inline = pos_inline + 1 % To find where it is. Not used.
5408
5409         % Elongated glyphs
5410         if elong_map then
5411             local locale = node.get_attribute(n, LOCALE)
5412             if elong_map[locale] and elong_map[locale][n.font] and
5413                 elong_map[locale][n.font][n.char] then
5414                 table.insert(elongs, {node = n, locale = locale} )
5415                 node.set_attribute(n.prev, KASHIDA, 0)
5416             end
5417         end
5418
5419         % Tatwil
5420         if Babel.kashida_wts then
5421             local k_wt = node.get_attribute(n, KASHIDA)
5422             if k_wt > 0 then % todo. parameter for multi inserts
5423                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5424             end
5425         end
5426
5427     end % of node.traverse_id
5428
5429     if #elongs == 0 and #k_list == 0 then goto next_line end
5430     full = line.width
5431     shift = line.shift
5432     goal = full * Babel.arabic.justify_factor % A bit crude
5433     width = node.dimensions(line.head)      % The 'natural' width
5434
5435     % == Elongated ==
5436     % Original idea taken from 'chickenize'
5437     while (#elongs > 0 and width < goal) do
5438         subst_done = true
5439         local x = #elongs
5440         local curr = elongs[x].node
5441         local oldchar = curr.char
5442         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5443         width = node.dimensions(line.head) % Check if the line is too wide
5444         % Substitute back if the line would be too wide and break:
5445         if width > goal then
5446             curr.char = oldchar

```

```

5447         break
5448     end
5449     % If continue, pop the just substituted node from the list:
5450     table.remove(elongs, x)
5451 end
5452
5453 % == Tatwil ==
5454 if #k_list == 0 then goto next_line end
5455
5456 width = node.dimensions(line.head)    % The 'natural' width
5457 k_curr = #k_list
5458 wt_pos = 1
5459
5460 while width < goal do
5461     subst_done = true
5462     k_item = k_list[k_curr].node
5463     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5464         d = node.copy(k_item)
5465         d.char = 0x0640
5466         line.head, new = node.insert_after(line.head, k_item, d)
5467         width_new = node.dimensions(line.head)
5468         if width > goal or width == width_new then
5469             node.remove(line.head, new) % Better compute before
5470             break
5471         end
5472         width = width_new
5473     end
5474     if k_curr == 1 then
5475         k_curr = #k_list
5476         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5477     else
5478         k_curr = k_curr - 1
5479     end
5480 end
5481
5482 ::next_line::
5483
5484 % Must take into account marks and ins, see luatex manual.
5485 % Have to be executed only if there are changes. Investigate
5486 % what's going on exactly.
5487 if subst_done and not gc then
5488     d = node.hpack(line.head, full, 'exactly')
5489     d.shift = shift
5490     node.insert_before(head, line, d)
5491     node.remove(head, line)
5492 end
5493 end % if process line
5494 end
5495 }
5496 \endgroup
5497 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5498 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5499 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
5500 \DisableBabelHook{babel-fontspec}
5501 <<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 from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5502% TODO - to a lua file
5503 \directlua{
5504 Babel.script_blocks = {
5505   ['dflt'] = {},
5506   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5507              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5508   ['Armn'] = {{0x0530, 0x058F}},
5509   ['Beng'] = {{0x0980, 0x09FF}},
5510   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5511   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5512   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5513              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5514   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5515   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5516              {0xAB00, 0xAB2F}},
5517   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5518   % Don't follow strictly Unicode, which places some Coptic letters in
5519   % the 'Greek and Coptic' block
5520   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5521   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5522              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5523              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5524              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5525              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5526              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5527   ['Hebr'] = {{0x0590, 0x05FF}},
5528   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5529              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5530   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5531   ['Knda'] = {{0x0C80, 0x0CFF}},
5532   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5533              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5534              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5535   ['Lao0'] = {{0x0E80, 0x0EFF}},
5536   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5537              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5538              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5539   ['Mahj'] = {{0x11150, 0x1117F}},
5540   ['Mlym'] = {{0x0D00, 0x0D7F}},
5541   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5542   ['Orya'] = {{0x0B00, 0x0B7F}},
5543   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5544   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5545   ['Taml'] = {{0x0B80, 0x0BFF}},
5546   ['Telu'] = {{0x0C00, 0x0C7F}},
5547   ['Tfng'] = {{0x2D30, 0x2D7F}},
5548   ['Thai'] = {{0x0E00, 0x0E7F}},
5549   ['Tibt'] = {{0x0F00, 0x0FFF}},
5550   ['Vaii'] = {{0xA500, 0xA63F}},
5551   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
```



```

5552 }
5553
5554 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5555 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5556 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5557
5558 function Babel.locale_map(head)
5559   if not Babel.locale_mapped then return head end
5560
5561   local LOCALE = Babel.attr_locale
5562   local GLYPH = node.id('glyph')
5563   local inmath = false
5564   local toloc_save
5565   for item in node.traverse(head) do
5566     local toloc
5567     if not inmath and item.id == GLYPH then
5568       % Optimization: build a table with the chars found
5569       if Babel.chr_to_loc[item.char] then
5570         toloc = Babel.chr_to_loc[item.char]
5571       else
5572         for lc, maps in pairs(Babel.loc_to_scr) do
5573           for _, rg in pairs(maps) do
5574             if item.char >= rg[1] and item.char <= rg[2] then
5575               Babel.chr_to_loc[item.char] = lc
5576               toloc = lc
5577               break
5578             end
5579           end
5580         end
5581       end
5582       % Now, take action, but treat composite chars in a different
5583       % fashion, because they 'inherit' the previous locale. Not yet
5584       % optimized.
5585       if not toloc and
5586         (item.char >= 0x0300 and item.char <= 0x036F) or
5587         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5588         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5589         toloc = toloc_save
5590       end
5591       if toloc and toloc > -1 then
5592         if Babel.locale_props[toloc].lg then
5593           item.lang = Babel.locale_props[toloc].lg
5594           node.set_attribute(item, LOCALE, toloc)
5595         end
5596         if Babel.locale_props[toloc]['/'..item.font] then
5597           item.font = Babel.locale_props[toloc]['/'..item.font]
5598         end
5599         toloc_save = toloc
5600       end
5601     elseif not inmath and item.id == 7 then
5602       item.replace = item.replace and Babel.locale_map(item.replace)
5603       item.pre = item.pre and Babel.locale_map(item.pre)
5604       item.post = item.post and Babel.locale_map(item.post)
5605     elseif item.id == node.id'math' then
5606       inmath = (item.subtype == 0)
5607     end
5608   end
5609   return head
5610 end

```

5611 }

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

```

5612 \newcommand\babelcharproperty[1]{%
5613   \count@=#1\relax
5614   \ifvmode
5615     \expandafter\babel@chprop
5616   \else
5617     \babel@error{\string\babelcharproperty\space can be used only in\%
5618       vertical mode (preamble or between paragraphs)}%
5619     {See the manual for futher info}%
5620   \fi}
5621 \newcommand\babel@chprop[3][\the\count@]{%
5622   \@tempcnta=#1\relax
5623   \babel@ifunset{\babel@chprop@#2}%
5624   {\babel@error{No property named '#2'. Allowed values are\%
5625     direction (bc), mirror (bmg), and linebreak (lb)}%
5626     {See the manual for futher info}}%
5627   }%
5628   \loop
5629     \babel@cs{chprop@#2}{#3}%
5630   \ifnum\count@<\@tempcnta
5631     \advance\count@\@ne
5632   \repeat}
5633 \def\babel@chprop@direction#1{%
5634   \directlua{
5635     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5636     Babel.characters[\the\count@]['d'] = '#1'
5637   }}
5638 \let\babel@chprop@bc\babel@chprop@direction
5639 \def\babel@chprop@mirror#1{%
5640   \directlua{
5641     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5642     Babel.characters[\the\count@]['m'] = '\number#1'
5643   }}
5644 \let\babel@chprop@bmg\babel@chprop@mirror
5645 \def\babel@chprop@linebreak#1{%
5646   \directlua{
5647     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5648     Babel.cjk_characters[\the\count@]['c'] = '#1'
5649   }}
5650 \let\babel@chprop@lb\babel@chprop@linebreak
5651 \def\babel@chprop@locale#1{%
5652   \directlua{
5653     Babel.chr_to_loc = Babel.chr_to_loc or {}
5654     Babel.chr_to_loc[\the\count@] =
5655       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@@#1}}\space
5656   }}

```

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.

```

5657 \directlua{
5658   Babel.nohyphenation = \the\l@nohyphenation
5659 }

```

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, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to

function(m) return Babel.capt_map(m[1],1) end, where the last argument identifies the mapping to be applied to m[1]. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

```

5660 \begingroup
5661 \catcode`\~ = 12
5662 \catcode`\% = 12
5663 \catcode`\& = 14
5664 \gdef\babelposthyphenation#1#2#3{&%
5665   \bbl@activateposthyphen
5666   \begingroup
5667     \def\babeltempa{\bbl@add@list\babeltempb}&%
5668     \let\babeltempb\@empty
5669     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5670     \bbl@replace\bbl@tempa{,}{ ,}&%
5671     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5672       \bbl@ifsamestring{##1}{remove}&%
5673       {\bbl@add@list\babeltempb{nil}}&%
5674       {\directlua{
5675         local rep = {[##1]=}
5676         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5677         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5678         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5679         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5680         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5681         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5682         tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
5683       }}&%
5684     \directlua{
5685       local lbr = Babel.linebreaking.replacements[1]
5686       local u = unicode.utf8
5687       local id = \the\csname l@#1\endcsname
5688       &% Convert pattern:
5689       local patt = string.gsub(==[#2]==, '%s', '')
5690       if not u.find(patt, '()', nil, true) then
5691         patt = '()' .. patt .. '()'
5692       end
5693       patt = string.gsub(patt, '%(%)%^', '^()')
5694       patt = string.gsub(patt, '%$%(%)', '()$')
5695       patt = u.gsub(patt, '{(.)}',
5696         function (n)
5697           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5698         end)
5699       patt = u.gsub(patt, '{(%x%x%x%x+)}',
5700         function (n)
5701           return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5702         end)
5703       lbr[id] = lbr[id] or {}
5704       table.insert(lbr[id], { pattern = patt, replace = { \babeltempb } })
5705     }&%
5706   \endgroup}
5707 % TODO. Copy paste pattern.
5708 \gdef\babelprehyphenation#1#2#3{&%
5709   \bbl@activateprehyphen
5710   \begingroup
5711     \def\babeltempa{\bbl@add@list\babeltempb}&%
5712     \let\babeltempb\@empty

```

```

5713 \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5714 \bbl@replace\bbl@tempa{,}{ ,}&%
5715 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5716   \bbl@ifsamestring{##1}{remove}&%
5717   {\bbl@add@list\babeltempb{nil}}&%
5718   {\directlua{
5719     local rep = [=[#1]=]
5720     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5721     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5722     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5723     rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5724       'space = {' .. '%2, %3, %4' .. '}')
5725     rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5726       'spacefactor = {' .. '%2, %3, %4' .. '}')
5727     rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5728     tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
5729   }}&%
5730 \directlua{
5731   local lbkr = Babel.linebreaking.replacements[0]
5732   local u = unicode.utf8
5733   local id = \the\csname bbl@id@#1\endcsname
5734   &% Convert pattern:
5735   local patt = string.gsub([==[#2]==], '%s', '')
5736   local patt = string.gsub(patt, '|', ' ')
5737   if not u.find(patt, '()', nil, true) then
5738     patt = '()' .. patt .. '()'
5739   end
5740   &% patt = string.gsub(patt, '%(%)^', '^()')
5741   &% patt = string.gsub(patt, '([^\%])%$%$', '%1()$')
5742   patt = u.gsub(patt, '{(.)}',
5743     function (n)
5744       return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5745     end)
5746   patt = u.gsub(patt, '{(%x%x%x%x+)}',
5747     function (n)
5748       return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5749     end)
5750   lbkr[id] = lbkr[id] or {}
5751   table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5752 }&%
5753 \endgroup}
5754 \endgroup
5755 \def\bbl@activateposthyphen{%
5756   \let\bbl@activateposthyphen\relax
5757   \directlua{
5758     require('babel-transforms.lua')
5759     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5760   }}
5761 \def\bbl@activateprehyphen{%
5762   \let\bbl@activateprehyphen\relax
5763   \directlua{
5764     require('babel-transforms.lua')
5765     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5766   }}

```

13.9 Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before luaoftload is applied, which is loaded by default by \LaTeX . Just in case, consider the possibility it has

not been loaded.

```

5767 \def\bbl@activate@preotf{%
5768 \let\bbl@activate@preotf\relax % only once
5769 \directlua{
5770   Babel = Babel or {}
5771   %
5772   function Babel.pre_otfload_v(head)
5773     if Babel.numbers and Babel.digits_mapped then
5774       head = Babel.numbers(head)
5775     end
5776     if Babel.bidi_enabled then
5777       head = Babel.bidi(head, false, dir)
5778     end
5779     return head
5780   end
5781   %
5782   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5783     if Babel.numbers and Babel.digits_mapped then
5784       head = Babel.numbers(head)
5785     end
5786     if Babel.bidi_enabled then
5787       head = Babel.bidi(head, false, dir)
5788     end
5789     return head
5790   end
5791   %
5792   luatexbase.add_to_callback('pre_linebreak_filter',
5793     Babel.pre_otfload_v,
5794     'Babel.pre_otfload_v',
5795     luatexbase.priority_in_callback('pre_linebreak_filter',
5796       'luaotfload.node_processor') or nil)
5797   %
5798   luatexbase.add_to_callback('hpack_filter',
5799     Babel.pre_otfload_h,
5800     'Babel.pre_otfload_h',
5801     luatexbase.priority_in_callback('hpack_filter',
5802       'luaotfload.node_processor') or nil)
5803 }

```

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=`.

```

5804 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5805 \let\bbl@beforeforeign\leavevmode
5806 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5807 \RequirePackage{luatexbase}
5808 \bbl@activate@preotf
5809 \directlua{
5810   require('babel-data-bidi.lua')
5811   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5812     require('babel-bidi-basic.lua')
5813   \or
5814     require('babel-bidi-basic-r.lua')
5815   \fi}
5816 % TODO - to locale_props, not as separate attribute
5817 \newattribute\bbl@attr@dir
5818 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5819 % TODO. I don't like it, hackish:
5820 \bbl@exp{\output{\bodydir\pagedir\the\output}}

```

```

5821 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5822 \fi\fi
5823 \chardef\bbl@thetextdir\z@
5824 \chardef\bbl@thepardir\z@
5825 \def\bbl@getluadir#1{%
5826   \directlua{
5827     if tex.#1dir == 'TLT' then
5828       tex.sprint('0')
5829     elseif tex.#1dir == 'TRT' then
5830       tex.sprint('1')
5831     end}}
5832 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5833   \ifcase#3\relax
5834     \ifcase\bbl@getluadir{#1}\relax\else
5835       #2 TLT\relax
5836     \fi
5837   \else
5838     \ifcase\bbl@getluadir{#1}\relax
5839       #2 TRT\relax
5840     \fi
5841   \fi}
5842 \def\bbl@textdir#1{%
5843   \bbl@setluadir{text}\textdir{#1}%
5844   \chardef\bbl@thetextdir#1\relax
5845   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5846 \def\bbl@pardir#1{%
5847   \bbl@setluadir{par}\pardir{#1}%
5848   \chardef\bbl@thepardir#1\relax}
5849 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5850 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5851 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
5852 %
5853 \ifnum\bbl@bidimode>\z@
5854   \def\bbl@mathboxdir{%
5855     \ifcase\bbl@thetextdir\relax
5856       \everyhbox{\bbl@mathboxdir@aux L}%
5857     \else
5858       \everyhbox{\bbl@mathboxdir@aux R}%
5859     \fi}
5860   \def\bbl@mathboxdir@aux#1{%
5861     \@ifnextchar\egroup{}\{\textdir T#1T\relax}}
5862   \frozen@everymath\expandafter{%
5863     \expandafter\bbl@mathboxdir\the\frozen@everymath}
5864   \frozen@everydisplay\expandafter{%
5865     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5866 \fi

```

13.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least

in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```

5867 \bbl@trace{Redefinitions for bidi layout}
5868 \ifx\@eqnnum\undefined\else
5869   \ifx\bbl@attr@dir\undefined\else
5870     \edef\@eqnnum{%
5871       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5872       \unexpanded\expandafter{\@eqnnum}}%
5873   \fi
5874 \fi
5875 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5876 \ifnum\bbl@bidimode>\z@
5877   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5878     \bbl@exp{%
5879       \mathdir\the\bodydir
5880       #1%           Once entered in math, set boxes to restore values
5881       \<ifmmode>%
5882       \everyvbox{%
5883         \the\everyvbox
5884         \bodydir\the\bodydir
5885         \mathdir\the\mathdir
5886         \everyhbox{\the\everyhbox}%
5887         \everyvbox{\the\everyvbox}}%
5888       \everyhbox{%
5889         \the\everyhbox
5890         \bodydir\the\bodydir
5891         \mathdir\the\mathdir
5892         \everyhbox{\the\everyhbox}%
5893         \everyvbox{\the\everyvbox}}%
5894       \<fi>}}%
5895   \def\@hangfrom#1{%
5896     \setbox\@tempboxa\hbox{{#1}}%
5897     \hangindent\wd\@tempboxa
5898     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5899       \shapemode\@ne
5900     \fi
5901     \noindent\box\@tempboxa}
5902 \fi
5903 \IfBabelLayout{tabular}
5904   {\let\bbl@OL@tabular\@tabular
5905     \bbl@replace\@tabular{$}\{\bbl@nextfake$\}%
5906     \let\bbl@NL@tabular\@tabular
5907     \AtBeginDocument{%
5908       \ifx\bbl@NL@tabular\@tabular\else
5909         \bbl@replace\@tabular{$}\{\bbl@nextfake$\}%
5910         \let\bbl@NL@tabular\@tabular
5911       \fi}}
5912   {}
5913 \IfBabelLayout{lists}
5914   {\let\bbl@OL@list\list
5915     \bbl@sreplace\list{\parshape}\{\bbl@listparshape}%
5916     \let\bbl@NL@list\list
5917     \def\bbl@listparshape#1#2#3{%
5918       \parshape #1 #2 #3 %
5919       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5920         \shapemode\tw@
5921       \fi}}
5922   {}

```

```

5923 \IfBabelLayout{graphics}
5924 {\let\bbl@pictresetdir\relax
5925 \def\bbl@pictsetdir#1{%
5926 \ifcase\bbl@thetextdir
5927 \let\bbl@pictresetdir\relax
5928 \else
5929 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5930 \or\textdir TLT
5931 \else\bodydir TLT \textdir TLT
5932 \fi
5933 % \(\text|par)dir required in pgf:
5934 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5935 \fi}%
5936 \ifx\AddToHook\undefined\else
5937 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
5938 \directlua{
5939 Babel.get_picture_dir = true
5940 Babel.picture_has_bidi = 0
5941 function Babel.picture_dir (head)
5942 if not Babel.get_picture_dir then return head end
5943 for item in node.traverse(head) do
5944 if item.id == node.id'glyph' then
5945 local itemchar = item.char
5946 % TODO. Copypaste pattern from Babel.bidi (-r)
5947 local chardata = Babel.characters[itemchar]
5948 local dir = chardata and chardata.d or nil
5949 if not dir then
5950 for nn, et in ipairs(Babel.ranges) do
5951 if itemchar < et[1] then
5952 break
5953 elseif itemchar <= et[2] then
5954 dir = et[3]
5955 break
5956 end
5957 end
5958 end
5959 if dir and (dir == 'al' or dir == 'r') then
5960 Babel.picture_has_bidi = 1
5961 end
5962 end
5963 end
5964 return head
5965 end
5966 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5967 "Babel.picture_dir")
5968 }%
5969 \AtBeginDocument{%
5970 \long\def\put(#1,#2)#3{%
5971 \@killglue
5972 % Try:
5973 \ifx\bbl@pictresetdir\relax
5974 \def\bbl@tempc{0}%
5975 \else
5976 \directlua{
5977 Babel.get_picture_dir = true
5978 Babel.picture_has_bidi = 0
5979 }%
5980 \setbox\z@\hb@xt@\z@{%
5981 \@defaultunitsset\@tempdimc{#1}\unitlength

```



```

5982         \kern\@tempdimc
5983         #3\hss}%
5984         \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
5985     \fi
5986     % Do:
5987     \@defaultunitsset\@tempdimc{#2}\unitlength
5988     \raise\@tempdimc\hb@xt@\z@{%
5989         \@defaultunitsset\@tempdimc{#1}\unitlength
5990         \kern\@tempdimc
5991         {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
5992     \ignorespaces}%
5993     \MakeRobust\put}%
5994 \fi
5995 \AtBeginDocument
5996 {\ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
5997     \ifx\AddToHook\@undefined
5998         \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
5999         {\bbl@pictsetdir\z@\pgfpicturetrue}%
6000     \else
6001         \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6002     \fi
6003     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6004     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6005 \fi
6006 \ifx\tikzpicture\@undefined\else
6007     \ifx\AddToHook\@undefined\else
6008         \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6009     \fi
6010     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6011     \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6012 \fi
6013 \ifx\AddToHook\@undefined\else
6014     \ifx\tcolorbox\@undefined\else
6015         \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6016         \bbl@sreplace\tcb@savebox
6017         {\ignorespaces}{\ignorespaces\bbl@pictresetdir}%
6018         \ifx\tikzpicture\tcb@hooked\@undefined\else
6019             \bbl@sreplace\tikzpicture\tcb@hooked{\noexpand\tikzpicture}%
6020             {\textdir TLT\noexpand\tikzpicture}%
6021         \fi
6022     \fi
6023 \fi
6024 }}
6025 {}

```

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.

```

6026 \IfBabelLayout{counters}%
6027 {\let\bbl@OL@@textsuperscript\textsuperscript
6028  \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6029  \let\bbl@latinarabic=\@arabic
6030  \let\bbl@OL@@arabic\@arabic
6031  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
6032 \@ifpackagewith{babel}{bidi=default}%
6033 {\let\bbl@asciroman=\@roman
6034  \let\bbl@OL@@roman\@roman
6035  \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6036  \let\bbl@asciiRoman=\@Roman

```

```

6037 \let\bbl@OL@@roman\@Roman
6038 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6039 \let\bbl@OL@labelenumii\labelenumii
6040 \def\labelenumii{}\theenumii{}%
6041 \let\bbl@OL@p@enumiii\p@enumiii
6042 \def\p@enumiii{\p@enumii}\theenumii{}\{\}\{\}
6043 <<Footnote changes>>
6044 \IfBabelLayout{footnotes}%
6045 {\let\bbl@OL@footnote\footnote
6046 \BabelFootnote\footnote\languagename{}\{\}%
6047 \BabelFootnote\localfootnote\languagename{}\{\}%
6048 \BabelFootnote\mainfootnote{}\{\}\{\}}
6049 {}

```

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.

```

6050 \IfBabelLayout{extras}%
6051 {\let\bbl@OL@underline\underline
6052 \bbl@sreplace\underline{\$@@underline}\bbl@nextfake$@@underline}%
6053 \let\bbl@OL@LaTeX2e\LaTeX2e
6054 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6055 \if b\expandafter\@car\@f@series\@nil\boldmath\fi
6056 \babelsublr{%
6057 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
6058 {}
6059 </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.

```

6060 <(*transforms)
6061 Babel.linebreaking.replacements = {}
6062 Babel.linebreaking.replacements[0] = {} -- pre
6063 Babel.linebreaking.replacements[1] = {} -- post
6064
6065 -- Discretionaries contain strings as nodes
6066 function Babel.str_to_nodes(fn, matches, base)
6067 local n, head, last
6068 if fn == nil then return nil end
6069 for s in string.utfvalues(fn(matches)) do
6070   if base.id == 7 then
6071     base = base.replace
6072   end
6073   n = node.copy(base)
6074   n.char = s
6075   if not head then
6076     head = n
6077   else

```

```

6078     last.next = n
6079     end
6080     last = n
6081 end
6082 return head
6083 end
6084
6085 Babel.fetch_subtext = {}
6086
6087 Babel.ignore_pre_char = function(node)
6088     return (node.lang == Babel.nohyphenation)
6089 end
6090
6091 -- Merging both functions doesn't seem feasible, because there are too
6092 -- many differences.
6093 Babel.fetch_subtext[0] = function(head)
6094     local word_string = ''
6095     local word_nodes = {}
6096     local lang
6097     local item = head
6098     local inmath = false
6099
6100     while item do
6101
6102         if item.id == 11 then
6103             inmath = (item.subtype == 0)
6104         end
6105
6106         if inmath then
6107             -- pass
6108
6109         elseif item.id == 29 then
6110             local locale = node.get_attribute(item, Babel.attr_locale)
6111
6112             if lang == locale or lang == nil then
6113                 lang = lang or locale
6114                 if Babel.ignore_pre_char(item) then
6115                     word_string = word_string .. Babel.us_char
6116                 else
6117                     word_string = word_string .. unicode.utf8.char(item.char)
6118                 end
6119                 word_nodes[#word_nodes+1] = item
6120             else
6121                 break
6122             end
6123
6124         elseif item.id == 12 and item.subtype == 13 then
6125             word_string = word_string .. ' '
6126             word_nodes[#word_nodes+1] = item
6127
6128             -- Ignore leading unrecognized nodes, too.
6129             elseif word_string ~= '' then
6130                 word_string = word_string .. Babel.us_char
6131                 word_nodes[#word_nodes+1] = item -- Will be ignored
6132             end
6133
6134         item = item.next
6135     end
6136

```

```

6137 -- Here and above we remove some trailing chars but not the
6138 -- corresponding nodes. But they aren't accessed.
6139 if word_string:sub(-1) == ' ' then
6140     word_string = word_string:sub(1,-2)
6141 end
6142 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6143 return word_string, word_nodes, item, lang
6144 end
6145
6146 Babel.fetch_subtext[1] = function(head)
6147     local word_string = ''
6148     local word_nodes = {}
6149     local lang
6150     local item = head
6151     local inmath = false
6152
6153     while item do
6154
6155         if item.id == 11 then
6156             inmath = (item.subtype == 0)
6157         end
6158
6159         if inmath then
6160             -- pass
6161
6162         elseif item.id == 29 then
6163             if item.lang == lang or lang == nil then
6164                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6165                     lang = lang or item.lang
6166                     word_string = word_string .. unicode.utf8.char(item.char)
6167                     word_nodes[#word_nodes+1] = item
6168                 end
6169             else
6170                 break
6171             end
6172
6173         elseif item.id == 7 and item.subtype == 2 then
6174             word_string = word_string .. '='
6175             word_nodes[#word_nodes+1] = item
6176
6177         elseif item.id == 7 and item.subtype == 3 then
6178             word_string = word_string .. '|'
6179             word_nodes[#word_nodes+1] = item
6180
6181         -- (1) Go to next word if nothing was found, and (2) implicitly
6182         -- remove leading USs.
6183         elseif word_string == '' then
6184             -- pass
6185
6186         -- This is the responsible for splitting by words.
6187         elseif (item.id == 12 and item.subtype == 13) then
6188             break
6189
6190         else
6191             word_string = word_string .. Babel.us_char
6192             word_nodes[#word_nodes+1] = item -- Will be ignored
6193         end
6194
6195         item = item.next

```

```

6196 end
6197
6198 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6199 return word_string, word_nodes, item, lang
6200 end
6201
6202 function Babel.pre_hyphenate_replace(head)
6203   Babel.hyphenate_replace(head, 0)
6204 end
6205
6206 function Babel.post_hyphenate_replace(head)
6207   Babel.hyphenate_replace(head, 1)
6208 end
6209
6210 Babel.us_char = string.char(31)
6211
6212 function Babel.hyphenate_replace(head, mode)
6213   local u = unicode.utf8
6214   local lbkr = Babel.linebreaking.replacements[mode]
6215
6216   local word_head = head
6217
6218   while true do -- for each subtext block
6219
6220     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6221
6222     if Babel.debug then
6223       print()
6224       print((mode == 0) and '@@@<' or '@@@>', w)
6225     end
6226
6227     if nw == nil and w == '' then break end
6228
6229     if not lang then goto next end
6230     if not lbkr[lang] then goto next end
6231
6232     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6233     -- loops are nested.
6234     for k=1, #lbkr[lang] do
6235       local p = lbkr[lang][k].pattern
6236       local r = lbkr[lang][k].replace
6237
6238       if Babel.debug then
6239         print('*****', p, mode)
6240       end
6241
6242       -- This variable is set in some cases below to the first *byte*
6243       -- after the match, either as found by u.match (faster) or the
6244       -- computed position based on sc if w has changed.
6245       local last_match = 0
6246       local step = 0
6247
6248       -- For every match.
6249       while true do
6250         if Babel.debug then
6251           print('====')
6252         end
6253         local new -- used when inserting and removing nodes
6254

```

```

6255     local matches = { u.match(w, p, last_match) }
6256
6257     if #matches < 2 then break end
6258
6259     -- Get and remove empty captures (with ()'s, which return a
6260     -- number with the position), and keep actual captures
6261     -- (from (...)), if any, in matches.
6262     local first = table.remove(matches, 1)
6263     local last = table.remove(matches, #matches)
6264     -- Non re-fetched substrings may contain \31, which separates
6265     -- subsubstrings.
6266     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6267
6268     local save_last = last -- with A()BC()D, points to D
6269
6270     -- Fix offsets, from bytes to unicode. Explained above.
6271     first = u.len(w:sub(1, first-1)) + 1
6272     last = u.len(w:sub(1, last-1)) -- now last points to C
6273
6274     -- This loop stores in a small table the nodes
6275     -- corresponding to the pattern. Used by 'data' to provide a
6276     -- predictable behavior with 'insert' (w_nodes is modified on
6277     -- the fly), and also access to 'remove'd nodes.
6278     local sc = first-1 -- Used below, too
6279     local data_nodes = {}
6280
6281     for q = 1, last-first+1 do
6282         data_nodes[q] = w_nodes[sc+q]
6283     end
6284
6285     -- This loop traverses the matched substring and takes the
6286     -- corresponding action stored in the replacement list.
6287     -- sc = the position in substr nodes / string
6288     -- rc = the replacement table index
6289     local rc = 0
6290
6291     while rc < last-first+1 do -- for each replacement
6292         if Babel.debug then
6293             print('.....', rc + 1)
6294         end
6295         sc = sc + 1
6296         rc = rc + 1
6297
6298         if Babel.debug then
6299             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6300             local ss = ''
6301             for itt in node.traverse(head) do
6302                 if itt.id == 29 then
6303                     ss = ss .. unicode.utf8.char(itt.char)
6304                 else
6305                     ss = ss .. '{' .. itt.id .. '}'
6306                 end
6307             end
6308             print('*****', ss)
6309         end
6310
6311         local crep = r[rc]
6312         local item = w_nodes[sc]

```

```

6314     local item_base = item
6315     local placeholder = Babel.us_char
6316     local d
6317
6318     if crep and crep.data then
6319         item_base = data_nodes[crep.data]
6320     end
6321
6322     if crep then
6323         step = crep.step or 0
6324     end
6325
6326     if crep and next(crep) == nil then -- = {}
6327         last_match = save_last    -- Optimization
6328         goto next
6329
6330     elseif crep == nil or crep.remove then
6331         node.remove(head, item)
6332         table.remove(w_nodes, sc)
6333         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6334         sc = sc - 1 -- Nothing has been inserted.
6335         last_match = utf8.offset(w, sc+1+step)
6336         goto next
6337
6338     elseif crep and crep.kashida then -- Experimental
6339         node.set_attribute(item,
6340             Babel.attr_kashida,
6341             crep.kashida)
6342         last_match = utf8.offset(w, sc+1+step)
6343         goto next
6344
6345     elseif crep and crep.string then
6346         local str = crep.string(matches)
6347         if str == '' then -- Gather with nil
6348             node.remove(head, item)
6349             table.remove(w_nodes, sc)
6350             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6351             sc = sc - 1 -- Nothing has been inserted.
6352         else
6353             local loop_first = true
6354             for s in string.utfvalues(str) do
6355                 d = node.copy(item_base)
6356                 d.char = s
6357                 if loop_first then
6358                     loop_first = false
6359                     head, new = node.insert_before(head, item, d)
6360                     if sc == 1 then
6361                         word_head = head
6362                     end
6363                     w_nodes[sc] = d
6364                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6365                 else
6366                     sc = sc + 1
6367                     head, new = node.insert_before(head, item, d)
6368                     table.insert(w_nodes, sc, new)
6369                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6370                 end
6371             end
6372             if Babel.debug then
6373                 print('.....', 'str')

```

```

6373         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6374     end
6375     end -- for
6376     node.remove(head, item)
6377 end -- if ''
6378 last_match = utf8.offset(w, sc+1+step)
6379 goto next
6380
6381 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6382     d = node.new(7, 0) -- (disc, discretionary)
6383     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6384     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6385     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6386     d.attr = item_base.attr
6387     if crep.pre == nil then -- TeXbook p96
6388         d.penalty = crep.penalty or tex.hyphenpenalty
6389     else
6390         d.penalty = crep.penalty or tex.exhyphenpenalty
6391     end
6392     placeholder = '|'
6393     head, new = node.insert_before(head, item, d)
6394
6395 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6396     -- ERROR
6397
6398 elseif crep and crep.penalty then
6399     d = node.new(14, 0) -- (penalty, userpenalty)
6400     d.attr = item_base.attr
6401     d.penalty = crep.penalty
6402     head, new = node.insert_before(head, item, d)
6403
6404 elseif crep and crep.space then
6405     -- 655360 = 10 pt = 10 * 65536 sp
6406     d = node.new(12, 13) -- (glue, spaceskip)
6407     local quad = font.getfont(item_base.font).size or 655360
6408     node.setglue(d, crep.space[1] * quad,
6409                 crep.space[2] * quad,
6410                 crep.space[3] * quad)
6411     if mode == 0 then
6412         placeholder = ' '
6413     end
6414     head, new = node.insert_before(head, item, d)
6415
6416 elseif crep and crep.spacefactor then
6417     d = node.new(12, 13) -- (glue, spaceskip)
6418     local base_font = font.getfont(item_base.font)
6419     node.setglue(d,
6420                 crep.spacefactor[1] * base_font.parameters['space'],
6421                 crep.spacefactor[2] * base_font.parameters['space_stretch'],
6422                 crep.spacefactor[3] * base_font.parameters['space_shrink'])
6423     if mode == 0 then
6424         placeholder = ' '
6425     end
6426     head, new = node.insert_before(head, item, d)
6427
6428 elseif mode == 0 and crep and crep.space then
6429     -- ERROR
6430
6431 end -- ie replacement cases

```



```

6432
6433     -- Shared by disc, space and penalty.
6434     if sc == 1 then
6435         word_head = head
6436     end
6437     if crep.insert then
6438         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6439         table.insert(w_nodes, sc, new)
6440         last = last + 1
6441     else
6442         w_nodes[sc] = d
6443         node.remove(head, item)
6444         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6445     end
6446
6447     last_match = utf8.offset(w, sc+1+step)
6448
6449     ::next::
6450
6451     end -- for each replacement
6452
6453     if Babel.debug then
6454         print('.....', '/')
6455         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6456     end
6457
6458     end -- for match
6459
6460     end -- for patterns
6461
6462     ::next::
6463     word_head = nw
6464 end -- for substring
6465 return head
6466 end
6467
6468 -- This table stores capture maps, numbered consecutively
6469 Babel.capture_maps = {}
6470
6471 -- The following functions belong to the next macro
6472 function Babel.capture_func(key, cap)
6473     local ret = "[" .. cap:gsub('{{[0-9]}}', "].m[%1].[" .. "]"
6474     local cnt
6475     local u = unicode.utf8
6476     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
6477     if cnt == 0 then
6478         ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6479             function (n)
6480                 return u.char(tonumber(n, 16))
6481             end)
6482     end
6483     ret = ret:gsub("%[%[%]]%.%", '')
6484     ret = ret:gsub("%.%.%[%[%]]%", '')
6485     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6486 end
6487
6488 function Babel.capt_map(from, mapno)
6489     return Babel.capture_maps[mapno][from] or from
6490 end

```

```

6491
6492 -- Handle the {n|abc|ABC} syntax in captures
6493 function Babel.capture_func_map(capno, from, to)
6494     local u = unicode.utf8
6495     from = u.gsub(from, '{(%x%x%x%x+)}',
6496         function (n)
6497             return u.char(tonumber(n, 16))
6498         end)
6499     to = u.gsub(to, '{(%x%x%x%x+)}',
6500         function (n)
6501             return u.char(tonumber(n, 16))
6502         end)
6503     local froms = {}
6504     for s in string.utfcharacters(from) do
6505         table.insert(froms, s)
6506     end
6507     local cnt = 1
6508     table.insert(Babel.capture_maps, {})
6509     local mlen = table.getn(Babel.capture_maps)
6510     for s in string.utfcharacters(to) do
6511         Babel.capture_maps[mlen][froms[cnt]] = s
6512         cnt = cnt + 1
6513     end
6514     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6515         (mlen) .. ").." .. "[["
6516 end
6517
6518 -- Create/Extend reversed sorted list of kashida weights:
6519 function Babel.capture_kashida(key, wt)
6520     wt = tonumber(wt)
6521     if Babel.kashida_wts then
6522         for p, q in ipairs(Babel.kashida_wts) do
6523             if wt == q then
6524                 break
6525             elseif wt > q then
6526                 table.insert(Babel.kashida_wts, p, wt)
6527                 break
6528             elseif table.getn(Babel.kashida_wts) == p then
6529                 table.insert(Babel.kashida_wts, wt)
6530             end
6531         end
6532     else
6533         Babel.kashida_wts = { wt }
6534     end
6535     return 'kashida = ' .. wt
6536 end
6537 </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.

```
6538 (*basic-r)
6539 Babel = Babel or {}
6540
6541 Babel.bidi_enabled = true
6542
6543 require('babel-data-bidi.lua')
6544
6545 local characters = Babel.characters
6546 local ranges = Babel.ranges
6547
6548 local DIR = node.id("dir")
6549
6550 local function dir_mark(head, from, to, outer)
6551   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6552   local d = node.new(DIR)
6553   d.dir = '+' .. dir
6554   node.insert_before(head, from, d)
6555   d = node.new(DIR)
6556   d.dir = '-' .. dir
6557   node.insert_after(head, to, d)
6558 end
6559
6560 function Babel.bidi(head, ispar)
6561   local first_n, last_n      -- first and last char with nums
6562   local last_es              -- an auxiliary 'last' used with nums
6563   local first_d, last_d      -- first and last char in L/R block
6564   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):

```

6565 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6566 local strong_lr = (strong == 'l') and 'l' or 'r'
6567 local outer = strong
6568
6569 local new_dir = false
6570 local first_dir = false
6571 local inmath = false
6572
6573 local last_lr
6574
6575 local type_n = ''
6576
6577 for item in node.traverse(head) do
6578
6579   -- three cases: glyph, dir, otherwise
6580   if item.id == node.id'glyph'
6581     or (item.id == 7 and item.subtype == 2) then
6582
6583     local itemchar
6584     if item.id == 7 and item.subtype == 2 then
6585       itemchar = item.replace.char
6586     else
6587       itemchar = item.char
6588     end
6589     local chardata = characters[itemchar]
6590     dir = chardata and chardata.d or nil
6591     if not dir then
6592       for nn, et in ipairs(ranges) do
6593         if itemchar < et[1] then
6594           break
6595         elseif itemchar <= et[2] then
6596           dir = et[3]
6597           break
6598         end
6599       end
6600     end
6601     dir = dir or 'l'
6602     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).

```

6603   if new_dir then
6604     attr_dir = 0
6605     for at in node.traverse(item.attr) do
6606       if at.number == Babel.attr_dir then
6607         attr_dir = at.value % 3
6608       end
6609     end
6610     if attr_dir == 1 then
6611       strong = 'r'
6612     elseif attr_dir == 2 then
6613       strong = 'al'
6614     else
6615       strong = 'l'
6616     end
6617     strong_lr = (strong == 'l') and 'l' or 'r'

```

```

6618         outer = strong_lr
6619         new_dir = false
6620     end
6621
6622     if dir == 'nsm' then dir = strong end          -- W1

```

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

```

6623         dir_real = dir          -- We need dir_real to set strong below
6624         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:

```

6625         if strong == 'al' then
6626             if dir == 'en' then dir = 'an' end          -- W2
6627             if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6628             strong_lr = 'r'          -- W3
6629         end

```

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

```

6630     elseif item.id == node.id'dir' and not inmath then
6631         new_dir = true
6632         dir = nil
6633     elseif item.id == node.id'math' then
6634         inmath = (item.subtype == 0)
6635     else
6636         dir = nil          -- Not a char
6637     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>.

```

6638     if dir == 'en' or dir == 'an' or dir == 'et' then
6639         if dir ~= 'et' then
6640             type_n = dir
6641         end
6642         first_n = first_n or item
6643         last_n = last_es or item
6644         last_es = nil
6645     elseif dir == 'es' and last_n then -- W3+W6
6646         last_es = item
6647     elseif dir == 'cs' then          -- it's right - do nothing
6648     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6649         if strong_lr == 'r' and type_n ~= '' then
6650             dir_mark(head, first_n, last_n, 'r')
6651         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6652             dir_mark(head, first_n, last_n, 'r')
6653             dir_mark(head, first_d, last_d, outer)
6654             first_d, last_d = nil, nil
6655         elseif strong_lr == 'l' and type_n ~= '' then
6656             last_d = last_n
6657         end
6658         type_n = ''
6659         first_n, last_n = nil, nil
6660     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:

```
6661   if dir == 'l' or dir == 'r' then
6662       if dir ~= outer then
6663           first_d = first_d or item
6664           last_d = item
6665       elseif first_d and dir ~= strong_lr then
6666           dir_mark(head, first_d, last_d, outer)
6667           first_d, last_d = nil, nil
6668       end
6669   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.

```
6670   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6671       item.char = characters[item.char] and
6672           characters[item.char].m or item.char
6673   elseif (dir or new_dir) and last_lr ~= item then
6674       local mir = outer .. strong_lr .. (dir or outer)
6675       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6676           for ch in node.traverse(node.next(last_lr)) do
6677               if ch == item then break end
6678               if ch.id == node.id'glyph' and characters[ch.char] then
6679                   ch.char = characters[ch.char].m or ch.char
6680               end
6681           end
6682       end
6683   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).

```
6684   if dir == 'l' or dir == 'r' then
6685       last_lr = item
6686       strong = dir_real           -- Don't search back - best save now
6687       strong_lr = (strong == 'l') and 'l' or 'r'
6688   elseif new_dir then
6689       last_lr = nil
6690   end
6691   end
```

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

```
6692   if last_lr and outer == 'r' then
6693       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6694           if characters[ch.char] then
6695               ch.char = characters[ch.char].m or ch.char
6696           end
6697       end
6698   end
6699   if first_n then
6700       dir_mark(head, first_n, last_n, outer)
6701   end
6702   if first_d then
6703       dir_mark(head, first_d, last_d, outer)
6704   end
```

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

```

6705 return node.prev(head) or head
6706 end
6707  $\langle$ /basic-r $\rangle$ 

And here the Lua code for bidi=basic:

6708  $\langle$ *basic $\rangle$ 
6709 Babel = Babel or {}
6710
6711 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6712
6713 Babel.fontmap = Babel.fontmap or {}
6714 Babel.fontmap[0] = {}      -- l
6715 Babel.fontmap[1] = {}      -- r
6716 Babel.fontmap[2] = {}      -- al/an
6717
6718 Babel.bidi_enabled = true
6719 Babel.mirroring_enabled = true
6720
6721 require('babel-data-bidi.lua')
6722
6723 local characters = Babel.characters
6724 local ranges = Babel.ranges
6725
6726 local DIR = node.id('dir')
6727 local GLYPH = node.id('glyph')
6728
6729 local function insert_implicit(head, state, outer)
6730     local new_state = state
6731     if state.sim and state.eim and state.sim ~= state.eim then
6732         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6733         local d = node.new(DIR)
6734         d.dir = '+' .. dir
6735         node.insert_before(head, state.sim, d)
6736         local d = node.new(DIR)
6737         d.dir = '-' .. dir
6738         node.insert_after(head, state.eim, d)
6739     end
6740     new_state.sim, new_state.eim = nil, nil
6741     return head, new_state
6742 end
6743
6744 local function insert_numeric(head, state)
6745     local new
6746     local new_state = state
6747     if state.san and state.ean and state.san ~= state.ean then
6748         local d = node.new(DIR)
6749         d.dir = '+TLT'
6750         _, new = node.insert_before(head, state.san, d)
6751         if state.san == state.sim then state.sim = new end
6752         local d = node.new(DIR)
6753         d.dir = '-TLT'
6754         _, new = node.insert_after(head, state.ean, d)
6755         if state.ean == state.eim then state.eim = new end
6756     end
6757     new_state.san, new_state.ean = nil, nil
6758     return head, new_state
6759 end
6760
6761 -- TODO - \hbox with an explicit dir can lead to wrong results

```

```

6762 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6763 -- was s made to improve the situation, but the problem is the 3-dir
6764 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6765 -- well.
6766
6767 function Babel.bidi(head, ispar, hdir)
6768   local d    -- d is used mainly for computations in a loop
6769   local prev_d = ''
6770   local new_d = false
6771
6772   local nodes = {}
6773   local outer_first = nil
6774   local inmath = false
6775
6776   local glue_d = nil
6777   local glue_i = nil
6778
6779   local has_en = false
6780   local first_et = nil
6781
6782   local ATDIR = Babel.attr_dir
6783
6784   local save_outer
6785   local temp = node.get_attribute(head, ATDIR)
6786   if temp then
6787     temp = temp % 3
6788     save_outer = (temp == 0 and 'l') or
6789                  (temp == 1 and 'r') or
6790                  (temp == 2 and 'al')
6791   elseif ispar then -- Or error? Shouldn't happen
6792     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6793   else -- Or error? Shouldn't happen
6794     save_outer = ('TRT' == hdir) and 'r' or 'l'
6795   end
6796   -- when the callback is called, we are just _after_ the box,
6797   -- and the textdir is that of the surrounding text
6798   -- if not ispar and hdir ~= tex.textdir then
6799   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6800   -- end
6801   local outer = save_outer
6802   local last = outer
6803   -- 'al' is only taken into account in the first, current loop
6804   if save_outer == 'al' then save_outer = 'r' end
6805
6806   local fontmap = Babel.fontmap
6807
6808   for item in node.traverse(head) do
6809
6810     -- In what follows, #node is the last (previous) node, because the
6811     -- current one is not added until we start processing the neutrals.
6812
6813     -- three cases: glyph, dir, otherwise
6814     if item.id == GLYPH
6815        or (item.id == 7 and item.subtype == 2) then
6816
6817       local d_font = nil
6818       local item_r
6819       if item.id == 7 and item.subtype == 2 then
6820         item_r = item.replace -- automatic discs have just 1 glyph

```



```

6821     else
6822         item_r = item
6823     end
6824     local chardata = characters[item_r.char]
6825     d = chardata and chardata.d or nil
6826     if not d or d == 'nsm' then
6827         for nn, et in ipairs(ranges) do
6828             if item_r.char < et[1] then
6829                 break
6830             elseif item_r.char <= et[2] then
6831                 if not d then d = et[3]
6832                 elseif d == 'nsm' then d_font = et[3]
6833                 end
6834                 break
6835             end
6836         end
6837     end
6838     d = d or 'l'
6839
6840     -- A short 'pause' in bidi for mapfont
6841     d_font = d_font or d
6842     d_font = (d_font == 'l' and 0) or
6843             (d_font == 'nsm' and 0) or
6844             (d_font == 'r' and 1) or
6845             (d_font == 'al' and 2) or
6846             (d_font == 'an' and 2) or nil
6847     if d_font and fontmap and fontmap[d_font][item_r.font] then
6848         item_r.font = fontmap[d_font][item_r.font]
6849     end
6850
6851     if new_d then
6852         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6853         if inmath then
6854             attr_d = 0
6855         else
6856             attr_d = node.get_attribute(item, ATDIR)
6857             attr_d = attr_d % 3
6858         end
6859         if attr_d == 1 then
6860             outer_first = 'r'
6861             last = 'r'
6862         elseif attr_d == 2 then
6863             outer_first = 'r'
6864             last = 'al'
6865         else
6866             outer_first = 'l'
6867             last = 'l'
6868         end
6869         outer = last
6870         has_en = false
6871         first_et = nil
6872         new_d = false
6873     end
6874
6875     if glue_d then
6876         if (d == 'l' and 'l' or 'r') ~= glue_d then
6877             table.insert(nodes, {glue_i, 'on', nil})
6878         end
6879         glue_d = nil

```

```

6880         glue_i = nil
6881     end
6882
6883     elseif item.id == DIR then
6884         d = nil
6885         new_d = true
6886
6887     elseif item.id == node.id'glue' and item.subtype == 13 then
6888         glue_d = d
6889         glue_i = item
6890         d = nil
6891
6892     elseif item.id == node.id'math' then
6893         inmath = (item.subtype == 0)
6894
6895     else
6896         d = nil
6897     end
6898
6899     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6900     if last == 'al' and d == 'en' then
6901         d = 'an'          -- W3
6902     elseif last == 'al' and (d == 'et' or d == 'es') then
6903         d = 'on'          -- W6
6904     end
6905
6906     -- EN + CS/ES + EN      -- W4
6907     if d == 'en' and #nodes >= 2 then
6908         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6909             and nodes[#nodes-1][2] == 'en' then
6910             nodes[#nodes][2] = 'en'
6911         end
6912     end
6913
6914     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6915     if d == 'an' and #nodes >= 2 then
6916         if (nodes[#nodes][2] == 'cs')
6917             and nodes[#nodes-1][2] == 'an' then
6918             nodes[#nodes][2] = 'an'
6919         end
6920     end
6921
6922     -- ET/EN                  -- W5 + W7->l / W6->on
6923     if d == 'et' then
6924         first_et = first_et or (#nodes + 1)
6925     elseif d == 'en' then
6926         has_en = true
6927         first_et = first_et or (#nodes + 1)
6928     elseif first_et then      -- d may be nil here !
6929         if has_en then
6930             if last == 'l' then
6931                 temp = 'l'    -- W7
6932             else
6933                 temp = 'en'   -- W5
6934             end
6935         else
6936             temp = 'on'       -- W6
6937         end
6938         for e = first_et, #nodes do

```

```

6939         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6940     end
6941     first_et = nil
6942     has_en = false
6943 end
6944
6945 -- Force mathdir in math if ON (currently works as expected only
6946 -- with 'l')
6947 if inmath and d == 'on' then
6948     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6949 end
6950
6951 if d then
6952     if d == 'al' then
6953         d = 'r'
6954         last = 'al'
6955     elseif d == 'l' or d == 'r' then
6956         last = d
6957     end
6958     prev_d = d
6959     table.insert(nodes, {item, d, outer_first})
6960 end
6961
6962 outer_first = nil
6963
6964 end
6965
6966 -- TODO -- repeated here in case EN/ET is the last node. Find a
6967 -- better way of doing things:
6968 if first_et then -- dir may be nil here !
6969     if has_en then
6970         if last == 'l' then
6971             temp = 'l' -- W7
6972         else
6973             temp = 'en' -- W5
6974         end
6975     else
6976         temp = 'on' -- W6
6977     end
6978     for e = first_et, #nodes do
6979         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6980     end
6981 end
6982
6983 -- dummy node, to close things
6984 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6985
6986 ----- NEUTRAL -----
6987
6988 outer = save_outer
6989 last = outer
6990
6991 local first_on = nil
6992
6993 for q = 1, #nodes do
6994     local item
6995
6996     local outer_first = nodes[q][3]
6997     outer = outer_first or outer

```

```

6998     last = outer_first or last
6999
7000     local d = nodes[q][2]
7001     if d == 'an' or d == 'en' then d = 'r' end
7002     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7003
7004     if d == 'on' then
7005         first_on = first_on or q
7006     elseif first_on then
7007         if last == d then
7008             temp = d
7009         else
7010             temp = outer
7011         end
7012         for r = first_on, q - 1 do
7013             nodes[r][2] = temp
7014             item = nodes[r][1] -- MIRRORING
7015             if Babel.mirroring_enabled and item.id == GLYPH
7016                 and temp == 'r' and characters[item.char] then
7017                 local font_mode = font.fonts[item.font].properties.mode
7018                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7019                     item.char = characters[item.char].m or item.char
7020                 end
7021             end
7022         end
7023         first_on = nil
7024     end
7025
7026     if d == 'r' or d == 'l' then last = d end
7027 end
7028
7029 ----- IMPLICIT, REORDER -----
7030
7031 outer = save_outer
7032 last = outer
7033
7034 local state = {}
7035 state.has_r = false
7036
7037 for q = 1, #nodes do
7038
7039     local item = nodes[q][1]
7040
7041     outer = nodes[q][3] or outer
7042
7043     local d = nodes[q][2]
7044
7045     if d == 'nsm' then d = last end -- W1
7046     if d == 'en' then d = 'an' end
7047     local isdir = (d == 'r' or d == 'l')
7048
7049     if outer == 'l' and d == 'an' then
7050         state.san = state.san or item
7051         state.ean = item
7052     elseif state.san then
7053         head, state = insert_numeric(head, state)
7054     end
7055
7056     if outer == 'l' then

```

```

7057     if d == 'an' or d == 'r' then      -- im -> implicit
7058         if d == 'r' then state.has_r = true end
7059         state.sim = state.sim or item
7060         state.eim = item
7061     elseif d == 'l' and state.sim and state.has_r then
7062         head, state = insert_implicit(head, state, outer)
7063     elseif d == 'l' then
7064         state.sim, state.eim, state.has_r = nil, nil, false
7065     end
7066 else
7067     if d == 'an' or d == 'l' then
7068         if nodes[q][3] then -- nil except after an explicit dir
7069             state.sim = item -- so we move sim 'inside' the group
7070         else
7071             state.sim = state.sim or item
7072         end
7073         state.eim = item
7074     elseif d == 'r' and state.sim then
7075         head, state = insert_implicit(head, state, outer)
7076     elseif d == 'r' then
7077         state.sim, state.eim = nil, nil
7078     end
7079 end
7080
7081 if isdir then
7082     last = d      -- Don't search back - best save now
7083 elseif d == 'on' and state.san then
7084     state.san = state.san or item
7085     state.ean = item
7086 end
7087
7088 end
7089
7090 return node.prev(head) or head
7091 end
7092 </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.

```

7093 ⟨*nil⟩
7094 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7095 \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.

```

7096 \ifx\l@nil\undefined
7097   \newlanguage\l@nil
7098   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7099   \let\bbl@elt\relax
7100   \edef\bbl@languages{% Add it to the list of languages
7101     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
7102 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

7103 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
7104 \let\captionnil\@empty
7105 \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.

```

7106 \ldf@finish{nil}
7107 ⟨/nil⟩

```

16 Support for Plain T_EX (plain.def)

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```

7108 ⟨*bplain | blplain⟩
7109 \catcode`\{=1 % left brace is begin-group character
7110 \catcode`\}=2 % right brace is end-group character
7111 \catcode`\#=6 % hash mark is macro parameter character

```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```

7112 \openin 0 hyphen.cfg
7113 \ifeof0
7114 \else
7115   \let\input

```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that's done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
7116 \def\input #1 {%
7117     \let\input\
7118     \a hyphen.cfg
7119     \let\input\undefined
7120 }
7121 \fi
7122 </bplain | bplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
7123 <bplain>\a plain.tex
7124 <bplain>\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
7125 <bplain>\def\fmtname{babel-plain}
7126 <bplain>\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `lplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some \LaTeX features

The file `babel.def` expects some definitions made in the $\text{\LaTeX 2}_{\epsilon}$ style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading `babel`. `\BabelModifiers` can be set too (but not sure it works).

```
7127 <<*Emulate LaTeX>> ≡
7128 \def\@empty{}
7129 \def\loadlocalcfg#1{%
7130     \openin0#1.cfg
7131     \ifeof0
7132         \closein0
7133     \else
7134         \closein0
7135         {\immediate\write16{*****}%
7136             \immediate\write16{* Local config file #1.cfg used}%
7137             \immediate\write16{*}%
7138         }
7139         \input #1.cfg\relax
7140     \fi
7141     \@endofldf}
```

16.3 General tools

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

```
7142 \long\def\@firstofone#1{#1}
7143 \long\def\@firstoftwo#1#2{#1}
7144 \long\def\@secondoftwo#1#2{#2}
7145 \def\@nnil{\@nil}
7146 \def\@gobbletwo#1#2{}
7147 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7148 \def\@staror@long#1{%
7149     \@ifstar
7150     {\let\@ngrel@x\relax#1}%
7151 }
```

```

7151 {\let\l@ngrel@x\long#1}}
7152 \let\l@ngrel@x\relax
7153 \def\@car#1#2\@nil{#1}
7154 \def\@cdr#1#2\@nil{#2}
7155 \let\@typeset@protect\relax
7156 \let\protected@edef\edef
7157 \long\def\@gobble#1{}
7158 \edef\@backslashchar{\expandafter\@gobble\string\}
7159 \def\strip@prefix#1>{}
7160 \def\g@addto@macro#1#2{%
7161     \toks@\expandafter{#1#2}%
7162     \xdef#1{\the\toks@}}
7163 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7164 \def\@nameuse#1{\csname #1\endcsname}
7165 \def\@ifundefined#1{%
7166     \expandafter\ifx\csname#1\endcsname\relax
7167         \expandafter\@firstoftwo
7168     \else
7169         \expandafter\@secondoftwo
7170     \fi}
7171 \def\@expandtwoargs#1#2#3{%
7172     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7173 \def\zap@space#1 #2{%
7174     #1%
7175     \ifx#2\@empty\else\expandafter\zap@space\fi
7176     #2}
7177 \let\bbl@trace\@gobble
7178 \def\bbl@error#1#2{%
7179     \begingroup
7180         \newlinechar=`^^J
7181         \def\{^^J(babel) }%
7182         \errhelp{#2}\errmessage{\{#1}%
7183     \endgroup}
7184 \def\bbl@warning#1{%
7185     \begingroup
7186         \newlinechar=`^^J
7187         \def\{^^J(babel) }%
7188         \message{\{#1}%
7189     \endgroup}
7190 \let\bbl@infowarn\bbl@warning
7191 \def\bbl@info#1{%
7192     \begingroup
7193         \newlinechar=`^^J
7194         \def\{^^J}%
7195         \wlog{#1}%
7196     \endgroup}

```

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

```

7197 \ifx\@preamblecmds\@undefined
7198     \def\@preamblecmds{}
7199 \fi
7200 \def\@onlypreamble#1{%
7201     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7202         \@preamblecmds\do#1}}
7203 \@onlypreamble\@onlypreamble

```

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

```

7204 \def\begindocument{%

```



```

7205 \@begindocumenthook
7206 \global\let\@begindocumenthook\@undefined
7207 \def\do##1{\global\let##1\@undefined}%
7208 \@preamblecmds
7209 \global\let\do\noexpand}

7210 \ifx\@begindocumenthook\@undefined
7211 \def\@begindocumenthook{}
7212 \fi
7213 \@onlypreamble\@begindocumenthook
7214 \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`.

```

7215 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7216 \@onlypreamble\AtEndOfPackage
7217 \def\@endofldf{}
7218 \@onlypreamble\@endofldf
7219 \let\bbl@afterlang\@empty
7220 \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.

```

7221 \catcode`\&=\z@
7222 \ifx&\if@files\@undefined
7223 \expandafter\let\csname if@files\expandafter\endcsname
7224 \csname iffalse\endcsname
7225 \fi
7226 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7227 \def\newcommand{\@star@or@long\new@command}
7228 \def\new@command#1{%
7229 \@testopt{\@newcommand#1}0}
7230 \def\@newcommand#1[#2]{%
7231 \@ifnextchar [{\@xargdef#1[#2]}%
7232 {\@argdef#1[#2]}}
7233 \long\def\@argdef#1[#2]#3{%
7234 \@yargdef#1\@ne{#2}{#3}}
7235 \long\def\@xargdef#1[#2][#3]#4{%
7236 \expandafter\def\expandafter#1\expandafter{%
7237 \expandafter\@protected@testopt\expandafter #1%
7238 \csname\string#1\expandafter\endcsname{#3}}%
7239 \expandafter\@yargdef \csname\string#1\endcsname
7240 \tw@{#2}{#4}}
7241 \long\def\@yargdef#1#2#3{%
7242 \@tempcnta#3\relax
7243 \advance \@tempcnta \@ne
7244 \let\@hash@\relax
7245 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7246 \@tempcntb #2%
7247 \@whilenum\@tempcntb <\@tempcnta
7248 \do{%
7249 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7250 \advance\@tempcntb \@ne}%
7251 \let\@hash@###
7252 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7253 \def\providecommand{\@star@or@long\provide@command}
7254 \def\provide@command#1{%

```

```

7255 \begingroup
7256 \escapechar\m@ne\xdef\gtempa{\string#1}}%
7257 \endgroup
7258 \expandafter\ifundefined\gtempa
7259 {\def\reserved@a{\newcommand#1}}%
7260 {\let\reserved@a\relax
7261 \def\reserved@a{\newcommand\reserved@a}}%
7262 \reserved@a}%

7263 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7264 \def\declare@robustcommand#1{%
7265 \edef\reserved@a{\string#1}%
7266 \def\reserved@b{#1}%
7267 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7268 \edef#1{%
7269 \ifx\reserved@a\reserved@b
7270 \noexpand\x@protect
7271 \noexpand#1%
7272 \fi
7273 \noexpand\protect
7274 \expandafter\noexpand\csname
7275 \expandafter\@gobble\string#1 \endcsname
7276 }%
7277 \expandafter\newcommand\csname
7278 \expandafter\@gobble\string#1 \endcsname
7279 }
7280 \def\x@protect#1{%
7281 \ifx\protect\@typeset@protect\else
7282 \@x@protect#1%
7283 \fi
7284 }
7285 \catcode`\&=\z@ % Trick to hide conditionals
7286 \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`.

```

7287 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7288 \catcode`\&=4
7289 \ifx\in@\@undefined
7290 \def\in@#1#2{%
7291 \def\in@##1#1##2##3\in@{%
7292 \ifx\in@##2\in@false\else\in@true\fi}%
7293 \in@#2#1\in@\in@}
7294 \else
7295 \let\bbl@tempa\@empty
7296 \fi
7297 \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).

```

7298 \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.

```

7299 \def\ifl@aded#1#2#3#4{}

```

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

```
7300 \ifx\@tempcnta\@undefined
7301   \csname newcount\endcsname\@tempcnta\relax
7302 \fi
7303 \ifx\@tempcntb\@undefined
7304   \csname newcount\endcsname\@tempcntb\relax
7305 \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`).

```
7306 \ifx\bye\@undefined
7307   \advance\count10 by -2\relax
7308 \fi
7309 \ifx\@ifnextchar\@undefined
7310   \def\@ifnextchar#1#2#3{%
7311     \let\reserved@d=#1%
7312     \def\reserved@a{#2}\def\reserved@b{#3}%
7313     \futurelet\@let@token\@ifnch}
7314   \def\@ifnch{%
7315     \ifx\@let@token\@sptoken
7316       \let\reserved@c\@xifnch
7317     \else
7318       \ifx\@let@token\reserved@d
7319         \let\reserved@c\reserved@a
7320       \else
7321         \let\reserved@c\reserved@b
7322       \fi
7323     \fi
7324     \reserved@c}
7325   \def\:\let\@sptoken= } \: % this makes \@sptoken a space token
7326   \def\:\@xifnch \expandafter\def\: {\futurelet\@let@token\@ifnch}
7327 \fi
7328 \def\@testopt#1#2{%
7329   \@ifnextchar[#{#1}{#1[#2]}}
7330 \def\@protected@testopt#1{%
7331   \ifx\protect\@typeset@protect
7332     \expandafter\@testopt
7333   \else
7334     \@x@protect#1%
7335   \fi}
7336 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7337   #2\relax}\fi}
7338 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7339   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```
7340 \def\DeclareTextCommand{%
7341   \@dec@text@cmd\providecommand
7342 }
7343 \def\ProvideTextCommand{%
7344   \@dec@text@cmd\providecommand
7345 }
7346 \def\DeclareTextSymbol#1#2#3{%
7347   \@dec@text@cmd\chardef#1{#2}#3\relax
```

```

7348 }
7349 \def\@dec@text@cmd#1#2#3{%
7350   \expandafter\def\expandafter#2%
7351     \expandafter{%
7352       \csname#3-cmd\expandafter\endcsname
7353       \expandafter#2%
7354       \csname#3\string#2\endcsname
7355     }%
7356 %   \let\@ifdefinable\@rc@ifdefinable
7357   \expandafter#1\csname#3\string#2\endcsname
7358 }
7359 \def\@current@cmd#1{%
7360   \ifx\protect\@typeset@protect\else
7361     \noexpand#1\expandafter\@gobble
7362   \fi
7363 }
7364 \def\@changed@cmd#1#2{%
7365   \ifx\protect\@typeset@protect
7366     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7367       \expandafter\ifx\csname ?\string#1\endcsname\relax
7368         \expandafter\def\csname ?\string#1\endcsname{%
7369           \@changed@x@err{#1}%
7370         }%
7371       \fi
7372       \global\expandafter\let
7373         \csname\cf@encoding \string#1\expandafter\endcsname
7374         \csname ?\string#1\endcsname
7375     \fi
7376     \csname\cf@encoding\string#1%
7377     \expandafter\endcsname
7378   \else
7379     \noexpand#1%
7380   \fi
7381 }
7382 \def\@changed@x@err#1{%
7383   \errhelp{Your command will be ignored, type <return> to proceed}%
7384   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7385 \def\DeclareTextCommandDefault#1{%
7386   \DeclareTextCommand#1?%
7387 }
7388 \def\ProvideTextCommandDefault#1{%
7389   \ProvideTextCommand#1?%
7390 }
7391 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7392 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7393 \def\DeclareTextAccent#1#2#3{%
7394   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7395 }
7396 \def\DeclareTextCompositeCommand#1#2#3#4{%
7397   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7398   \edef\reserved@b{\string##1}%
7399   \edef\reserved@c{%
7400     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7401   \ifx\reserved@b\reserved@c
7402     \expandafter\expandafter\expandafter\ifx
7403       \expandafter\@car\reserved@a\relax\relax\@nil
7404       \@text@composite
7405     \else
7406       \edef\reserved@b##1{%

```

```

7407         \def\expandafter\noexpand
7408             \csname#2\string#1\endcsname###1{%
7409             \noexpand\@text@composite
7410             \expandafter\noexpand\csname#2\string#1\endcsname
7411             ###1\noexpand\@empty\noexpand\@text@composite
7412             {##1}%
7413         }%
7414     }%
7415     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7416 \fi
7417 \expandafter\def\csname\expandafter\string\csname
7418     #2\endcsname\string#1-\string#3\endcsname{##4}
7419 \else
7420     \errhelp{Your command will be ignored, type <return> to proceed}%
7421     \errmessage{\string\DeclareTextCompositeCommand\space used on
7422         inappropriate command \protect#1}
7423 \fi
7424 }
7425 \def\@text@composite#1#2#3\@text@composite{%
7426     \expandafter\@text@composite@x
7427         \csname\string#1-\string#2\endcsname
7428 }
7429 \def\@text@composite@x#1#2{%
7430     \ifx#1\relax
7431         #2%
7432     \else
7433         #1%
7434     \fi
7435 }
7436 %
7437 \def\@strip@args#1:#2-#3\@strip@args{#2}
7438 \def\DeclareTextComposite#1#2#3#4{%
7439     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7440     \bgroup
7441         \lcode`\@=#4%
7442         \lowercase{%
7443             \egroup
7444             \reserved@a \@%
7445         }%
7446 }
7447 %
7448 \def\UseTextSymbol#1#2{#2}
7449 \def\UseTextAccent#1#2#3{}
7450 \def\@use@text@encoding#1{}
7451 \def\DeclareTextSymbolDefault#1#2{%
7452     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7453 }
7454 \def\DeclareTextAccentDefault#1#2{%
7455     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7456 }
7457 \def\cf@encoding{OT1}

Currently we only use the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$  method for accents for those that are known to be made active in
some language definition file.

7458 \DeclareTextAccent{"}{OT1}{127}
7459 \DeclareTextAccent{'}{OT1}{19}
7460 \DeclareTextAccent{^}{OT1}{94}
7461 \DeclareTextAccent`{OT1}{18}
7462 \DeclareTextAccent{~}{OT1}{126}

```

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

```
7463 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7464 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7465 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
7466 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7467 \DeclareTextSymbol{\i}{OT1}{16}
7468 \DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```
7469 \ifx\scriptsize\undefined
7470   \let\scriptsize\sevenrm
7471 \fi
```

And a few more “dummy” definitions.

```
7472 \def\language{english}%
7473 \let\bbl@opt@shorthands\@nnil
7474 \def\bbl@ifshorthand#1#2#3{#2}%
7475 \let\bbl@language@opts\@empty
7476 \ifx\babeloptionstrings\undefined
7477   \let\bbl@opt@strings\@nnil
7478 \else
7479   \let\bbl@opt@strings\babeloptionstrings
7480 \fi
7481 \def\BabelStringsDefault{generic}
7482 \def\bbl@tempa{normal}
7483 \ifx\babeloptionmath\bbl@tempa
7484   \def\bbl@mathnormal{\noexpand\textormath}
7485 \fi
7486 \def\AfterBabelLanguage#1#2{}
7487 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
7488 \let\bbl@afterlang\relax
7489 \def\bbl@opt@safe{BR}
7490 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
7491 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
7492 \expandafter\newif\csname ifbbl@single\endcsname
7493 \chardef\bbl@bidimode\z@
7494 <</Emulate LaTeX>>
```

A proxy file:

```
7495 <*\plain>
7496 \input babel.def
7497 </\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).