

Babel

Version 3.61.2419
2021/06/30

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	15
1.12	The base option	17
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	33
1.18	Dates	34
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Transforms	38
1.22	Selection based on BCP 47 tags	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	49
1.31	Current and future work	50
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	51
3	The interface between the core of babel and the language definition files	52
3.1	Guidelines for contributed languages	53
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	57
4	Changes	61
4.1	Changes in babel version 3.9	61

II	Source code	62
5	Identification and loading of required files	62
6	locale directory	62
7	Tools	63
7.1	Multiple languages	67
7.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	67
7.3	base	69
7.4	Conditional loading of shorthands	72
7.5	Cross referencing macros	73
7.6	Marks	76
7.7	Preventing clashes with other packages	77
7.7.1	ifthen	77
7.7.2	varioref	77
7.7.3	hhline	78
7.7.4	hyperref	78
7.7.5	fancyhdr	78
7.8	Encoding and fonts	79
7.9	Basic bidi support	81
7.10	Local Language Configuration	86
7.11	Language options	86
8	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	90
8.1	Tools	90
9	Multiple languages	91
9.1	Selecting the language	93
9.2	Errors	102
9.3	Hooks	105
9.4	Setting up language files	107
9.5	Shorthands	109
9.6	Language attributes	118
9.7	Support for saving macro definitions	120
9.8	Short tags	121
9.9	Hyphens	122
9.10	Multiencoding strings	123
9.11	Macros common to a number of languages	130
9.12	Making glyphs available	130
9.12.1	Quotation marks	130
9.12.2	Letters	132
9.12.3	Shorthands for quotation marks	132
9.12.4	Umlauts and tremas	133
9.13	Layout	135
9.14	Load engine specific macros	135
9.15	Creating and modifying languages	135
10	Adjusting the Babel behavior	157
11	Loading hyphenation patterns	158
12	Font handling with fontspec	163

13	Hooks for XeTeX and LuaTeX	167
13.1	XeTeX	167
13.2	Layout	170
13.3	LuaTeX	171
13.4	Southeast Asian scripts	177
13.5	CJK line breaking	178
13.6	Arabic justification	181
13.7	Common stuff	185
13.8	Automatic fonts and ids switching	185
13.9	Layout	198
13.10	Auto bidi with basic and basic-r	202
14	Data for CJK	213
15	The ‘nil’ language	213
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	215
16.4	Encoding related macros	218
17	Acknowledgements	221

Troubleshooting

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

Part I

User guide

What is this document about? This user guide focuses on internationalization and localization with \LaTeX and `pdftex`, `xetex` and `luatex` with the `babel` package. There are also some notes on its use with `e-Plain` and `pdf-Plain` \TeX . Part II describes the code, and usually it can be ignored.

What if I'm interested only in the latest changes? Changes and new features with relation to version 3.8 are highlighted with `New X.XX`, and there are some notes for the latest versions in [the babel site](#). The most recent features can be still unstable.

Can I help? Sure! If you are interested in the \TeX multilingual support, please join the [kadingira mail list](#). You can follow the development of `babel` in [GitHub](#) and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

It doesn't work for me! You can ask for help in some forums like [tex.stackexchange](#), but if you have found a bug, I strongly beg you to report it in [GitHub](#), which is much better than just complaining on an e-mail list or a web forum. Remember *warnings are not errors* by themselves, they just warn about possible problems or incompatibilities.

How can I contribute a new language? See section 3.1 for contributing a language.

I only need learn the most basic features. The first subsections (1.1-1.3) describe the traditional way of loading a language (with `ldf` files), which is usually all you need. The alternative way based on `ini` files, which complements the previous one (it does *not* replace it, although it is still necessary in some languages), is described below; go to 1.13.

I don't like manuals. I prefer sample files. This manual contains lots of examples and tips, but in [GitHub](#) there are many [sample files](#).

1 The user interface

1.1 Monolingual documents

In most cases, a single language is required, and then all you need in \LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in \LaTeX for an option – in this case a language – to be recognized by several packages.

Many languages are compatible with `xetex` and `luatex`. With them you can use `babel` to localize the documents. When these engines are used, the Latin script is covered by default in current \LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

TROUBLESHOOTING A common source of trouble is a wrong setting of the input encoding. Depending on the \TeX version you can get the following somewhat cryptic error:

```
! Paragraph ended before \UTFviii@three@octets was complete.
```

Or the more explanatory:

```
! Package inputenc Error: Invalid UTF-8 byte ...
```

Make sure you set the encoding actually used by your editor.

NOTE Because of the way babel has evolved, “language” can refer to (1) a set of hyphenation patterns as preloaded into the format, (2) a package option, (3) an ldf file, and (4) a name used in the document to select a language or dialect. So, a package option refers to a language in a generic way – sometimes it is the actual language name used to select it, sometimes it is a file name loading a language with a different name, sometimes it is a file name loading several languages. Please, read the documentation for specific languages for further info.

TROUBLESHOOTING The following warning is about hyphenation patterns, which are not under the direct control of babel:

```
Package babel Warning: No hyphenation patterns were preloaded for
(babel)                the language `LANG' into the format.
(babel)                Please, configure your TeX system to add them and
(babel)                rebuild the format. Now I will use the patterns
(babel)                preloaded for \language=0 instead on input line 57.
```

The document will be typeset, but very likely the text will not be correctly hyphenated. Some languages may be raising this warning wrongly (because they are not hyphenated); it is a bug to be fixed – just ignore it. See the manual of your distribution (MacTeX, MikTeX, TeXLive, etc.) for further info about how to configure it.

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

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

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

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

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

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

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

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

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

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

PDF_{TEX}

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

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

LUATEX/XETEX

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

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

1.3 Mostly monolingual documents

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

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

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

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

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

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

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

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

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

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

\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 `bidirectional` 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 `other language*` does not.

1.9 More on selection

`\babeltags` $\langle tag1 \rangle = \langle language1 \rangle, \langle tag2 \rangle = \langle language2 \rangle, \dots$

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{\langle language1 \rangle}\{<text>\}`, and `\begin{\langle tag1 \rangle}` to be `\begin{other language*}\{\langle language1 \rangle\}`, and so on. Note `\langle tag1 \rangle` 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 \LaTeX and conflicts with existing macros may arise (`\textlatin`, `\textbar`, `\textit`, `\textcolor` and many others). The same applies to environments, because `arabic` conflicts with `\arabic`. 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` $[\text{include}=\langle commands \rangle, \text{exclude}=\langle commands \rangle, \text{fontenc}=\langle encoding \rangle]\{\langle language \rangle\}$

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 breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

`\shorthandoff` `*{\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).

`\useshorthands` `*{\char}`

The command `\useshorthands` 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 `\useshorthands*{\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 `\useshorthands`. 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:

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

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

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

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

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

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

`\languageshorthands` $\langle language \rangle$

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

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

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

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

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

`\babelshorthand` $\langle shorthand \rangle$

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

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

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

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

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

⁶Thanks to Enrico Gregorio

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

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

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

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

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

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

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

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

1.11 Package options

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

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

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=	<p>$\langle char \rangle \langle char \rangle \dots$ off</p> <p>The only language shorthands activated are those given, like, eg:</p> <pre>\usepackage[esperanto,french,shorthands=;!]{babel}</pre> <p>If ' is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \string (otherwise they will be expanded by \LaTeX before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.</p>
safe=	<p>none ref bib</p> <p>Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibtex 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).</p>
math=	<p>active normal</p> <p>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.</p>
config=	<p>$\langle file \rangle$</p> <p>Load $\langle file \rangle$.cfg instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).</p>
main=	<p>$\langle language \rangle$</p> <p>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.</p>
headfoot=	<p>$\langle language \rangle$</p> <p>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.</p>

- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoiled by an unexpected .cfg file. However, if the key config is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** **New 3.9l** Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.
- silent** **New 3.9l** No warnings and no *infos* are written to the log file.⁸
- strings=** generic | unicode | encoded | *<label>* | **
 Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional T_EX, L^AT_EX 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 L^AT_EX tools, so use it only as a last resort).
- hyphenmap=** off | first | select | other | other*
New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.⁹ It can take the following values:
off deactivates this feature and no case mapping is applied;
first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it's the default if a single language option has been stated;¹⁰
select sets it only at \selectlanguage;
other also sets it at otherlanguage;
other* also sets it at otherlanguage* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other* for monolingual documents.¹¹
- bidi=** default | basic | basic-r | bidi-l | bidi-r
New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.24.
- layout=** **New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.24.

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

⁸You can use alternatively the package silence.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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

last language passed as option (by its name in `language.dat`). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

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

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

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

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

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

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

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 T_EX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the `...name` strings).

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them by means of `\babelprovide`. In other words, `\babelprovide` is mainly meant for auxiliary tasks, and as alternative when the `ldf`, for some reason, does not work as expected.

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

LUATEX/XETEX

```
\documentclass{book}

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

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

```

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

```

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

```

Or also:

```

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

```

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

```

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

```

Arabic Monolingual documents mostly work in `luatex`, but it must be fine tuned, particularly 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]{lᦺᦑ ᦺᦑ ᦺᦑ ᦺᦑ ᦺᦑ ᦺᦑ} % Random
```

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

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

Latin, Greek, Cyrillic Combining chars with the default luatex font renderer might be wrong; on the other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenation points are discarded (this bug 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}	bo	Tibetan ^u
agq	Aghem	brx	Bodo
ak	Akan	bs-Cyrl	Bosnian
am	Amharic ^{ul}	bs-Latn	Bosnian ^{ul}
ar	Arabic ^{ul}	bs	Bosnian ^{ul}
ar-DZ	Arabic ^{ul}	ca	Catalan ^{ul}
ar-MA	Arabic ^{ul}	ce	Chechen
ar-SY	Arabic ^{ul}	cgg	Chiga
as	Assamese	chr	Cherokee
asa	Asu	ckb	Central Kurdish
ast	Asturian ^{ul}	cop	Coptic
az-Cyrl	Azerbaijani	cs	Czech ^{ul}
az-Latn	Azerbaijani	cu	Church Slavic
az	Azerbaijani ^{ul}	cu-Cyrs	Church Slavic
bas	Basaa	cu-Glag	Church Slavic
be	Belarusian ^{ul}	cy	Welsh ^{ul}
bem	Bemba	da	Danish ^{ul}
bez	Bena	dav	Taita
bg	Bulgarian ^{ul}	de-AT	German ^{ul}
bm	Bambara	de-CH	German ^{ul}
bn	Bangla ^{ul}	de	German ^{ul}

dje	Zarma	ii	Sichuan Yi
dsb	Lower Sorbian ^{ul}	is	Icelandic ^{ul}
dua	Duala	it	Italian ^{ul}
dyo	Jola-Fonyi	ja	Japanese
dz	Dzongkha	jgo	Ngomba
ebu	Embu	jmc	Machame
ee	Ewe	ka	Georgian ^{ul}
el	Greek ^{ul}	kab	Kabyle
el-polyton	Polytonic Greek ^{ul}	kam	Kamba
en-AU	English ^{ul}	kde	Makonde
en-CA	English ^{ul}	kea	Kabuverdianu
en-GB	English ^{ul}	khq	Koyra Chiini
en-NZ	English ^{ul}	ki	Kikuyu
en-US	English ^{ul}	kk	Kazakh
en	English ^{ul}	kkj	Kako
eo	Esperanto ^{ul}	kl	Kalaallisut
es-MX	Spanish ^{ul}	kln	Kalenjin
es	Spanish ^{ul}	km	Khmer
et	Estonian ^{ul}	kn	Kannada ^{ul}
eu	Basque ^{ul}	ko	Korean
ewo	Ewondo	kok	Konkani
fa	Persian ^{ul}	ks	Kashmiri
ff	Fulah	ksb	Shambala
fi	Finnish ^{ul}	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French ^{ul}	ky	Kyrgyz
fr-BE	French ^{ul}	lag	Langi
fr-CA	French ^{ul}	lb	Luxembourgish
fr-CH	French ^{ul}	lg	Ganda
fr-LU	French ^{ul}	lkt	Lakota
fur	Friulian ^{ul}	ln	Lingala
fy	Western Frisian	lo	Lao ^{ul}
ga	Irish ^{ul}	lrc	Northern Luri
gd	Scottish Gaelic ^{ul}	lt	Lithuanian ^{ul}
gl	Galician ^{ul}	lu	Luba-Katanga
grc	Ancient Greek ^{ul}	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian ^{ul}
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa ¹	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ¹
hy	Armenian ^u	ms-SG	Malay ¹
ia	Interlingua ^{ul}	ms	Malay ^{ul}
id	Indonesian ^{ul}	mt	Maltese
ig	Igbo	mua	Mundang

my	Burmese	sn	Shona
mzn	Mazanderani	so	Somali
naq	Nama	sq	Albanian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Cyrl-BA	Serbian ^{ul}
nd	North Ndebele	sr-Cyrl-ME	Serbian ^{ul}
ne	Nepali	sr-Cyrl-XK	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Cyrl	Serbian ^{ul}
nmg	Kwasio	sr-Latn-BA	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sr-Latn-ME	Serbian ^{ul}
nnh	Ngiemboon	sr-Latn-XK	Serbian ^{ul}
nus	Nuer	sr-Latn	Serbian ^{ul}
nyn	Nyankole	sr	Serbian ^{ul}
om	Oromo	sv	Swedish ^{ul}
or	Odia	sw	Swahili
os	Ossetic	ta	Tamil ^u
pa-Arab	Punjabi	te	Telugu ^{ul}
pa-Guru	Punjabi	teo	Teso
pa	Punjabi	th	Thai ^{ul}
pl	Polish ^{ul}	ti	Tigrinya
pms	Piedmontese ^{ul}	tk	Turkmen ^{ul}
ps	Pashto	to	Tongan
pt-BR	Portuguese ^{ul}	tr	Turkish ^{ul}
pt-PT	Portuguese ^{ul}	twq	Tasawaq
pt	Portuguese ^{ul}	tzm	Central Atlas Tamazight
qu	Quechua	ug	Uyghur
rm	Romansh ^{ul}	uk	Ukrainian ^{ul}
rn	Rundi	ur	Urdu ^{ul}
ro	Romanian ^{ul}	uz-Arab	Uzbek
rof	Rombo	uz-Cyrl	Uzbek
ru	Russian ^{ul}	uz-Latn	Uzbek
rw	Kinyarwanda	uz	Uzbek
rwk	Rwa	vai-Latn	Vai
sa-Beng	Sanskrit	vai-Vaii	Vai
sa-Deva	Sanskrit	vai	Vai
sa-Gujr	Sanskrit	vi	Vietnamese ^{ul}
sa-Knda	Sanskrit	vun	Vunjo
sa-Mlym	Sanskrit	wae	Walser
sa-Telu	Sanskrit	xog	Soga
sa	Sanskrit	yav	Yangben
sah	Sakha	yi	Yiddish
saq	Samburu	yo	Yoruba
sbp	Sangu	yue	Cantonese
se	Northern Sami ^{ul}	zgh	Standard Moroccan Tamazight
seh	Sena		
ses	Koyraboro Senni	zh-Hans-HK	Chinese
sg	Sango	zh-Hans-MO	Chinese
shi-Latn	Tachelhit	zh-Hans-SG	Chinese
shi-Tfng	Tachelhit	zh-Hans	Chinese
shi	Tachelhit	zh-Hant-HK	Chinese
si	Sinhala	zh-Hant-MO	Chinese
sk	Slovak ^{ul}	zh-Hant	Chinese
sl	Slovenian ^{ul}	zh	Chinese
smn	Inari Sami	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	cantonese
akan	catalan
albanian	centralatlastamazight
american	centralkurdish
amharic	chechen
ancientgreek	cherokee
arabic	chiga
arabic-algeria	chinese-hans-hk
arabic-DZ	chinese-hans-mo
arabic-morocco	chinese-hans-sg
arabic-MA	chinese-hans
arabic-syria	chinese-hant-hk
arabic-SY	chinese-hant-mo
armenian	chinese-hant
assamese	chinese-simplified-hongkongsarchina
asturian	chinese-simplified-macausarchina
asu	chinese-simplified-singapore
australian	chinese-simplified
austrian	chinese-traditional-hongkongsarchina
azerbaijani-cyrillic	chinese-traditional-macausarchina
azerbaijani-cyrl	chinese-traditional
azerbaijani-latin	chinese
azerbaijani-latn	churchslavic
azerbaijani	churchslavic-cyrs
bafia	churchslavic-oldcyrillic ¹²
bambara	churchsslavic-glag
basaa	churchsslavic-glagolitic
basque	cognian
belarusian	cornish
bemba	croatian
bena	czech
bengali	danish
bodo	duala
bosnian-cyrillic	dutch
bosnian-cyrl	dzongkha
bosnian-latin	embu
bosnian-latn	english-au
bosnian	english-australia
brazilian	english-ca
breton	english-canada
british	english-gb
bulgarian	english-newzealand
burmese	english-nz
canadian	english-unitedkingdom

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

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

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

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

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

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

Modifying and adding values to ini files

New 3.39 There is a way to modify the values of ini files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghijklj`. Keys may be added, too. Without `import` you may modify the identification keys.

This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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.

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

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

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

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

import= *<language-tag>*

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

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

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

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

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

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

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

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

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

Loads only the strings. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remember there is an alternative syntax for the latter:

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

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

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

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

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

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

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

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

Same for `\Alph`.

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

onchar= `ids` | `fonts`

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

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

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

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

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

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

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

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

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

mapfont= `direction`

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

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

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

1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

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

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

The styles are:

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

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

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

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

\localeinfo `{\field}`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.21 Transforms

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

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

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

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

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

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for dad (simple and T _E X-friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	<code>oneletter.nobreak</code>	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: <i>! ? ; .</i>
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zsz</i> as <i>cs-cs, dz-dz</i> , etc.

¹⁵They are similar in concept, but not the same, as those in Unicode.

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 $\{\langle hyphenrules-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

New 3.37-3.39 With *luatex* it is now 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. Only a few rules are currently provided (see below), 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 $([\text{t}\acute{u}])$, the replacement could be $\{1|\text{t}\acute{u}|\text{t}\acute{o}\}$, which maps $\text{t}\acute{u}$ to $\text{t}\acute{o}$, and \acute{u} to \acute{o} , 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 `\babelfont`. The *transforms* mechanism supplements rather than replaces OTF features.

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

1.22 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{
  autoloader.bcp47 = on,
  autoloader.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

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.

LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

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

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection{<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.

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:

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

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

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

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

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

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

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

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

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

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

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}\{()\}
```

(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 may be enabled and disabled for all defined events with `\EnableBabelHook{\name}`, `\DisableBabelHook{\name}`.

Names containing the string `babel` are reserved (they are used, for example, by `\usesshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.
write This event comes just after the switching commands are written to the aux file.
beforeextras Just before executing `\extras⟨language⟩`. This event and the next one should not contain language-dependent code (for that, add it to `\extras⟨language⟩`).
afterextras Just after executing `\extras⟨language⟩`. For example, the following deactivates shorthands in all languages:

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

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

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

initiateactive (char as active, char as other, original char) **New 3.9i** Executed just after a shorthand has been ‘initiated’. The three parameters are the same character with different catcodes: active, other (`\string’ed`) and the original one.
afterreset **New 3.9i** Executed when selecting a language just after `\originalTeX` is run and reset to its base value, before executing `\captions⟨language⟩` and `\date⟨language⟩`.

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

everylanguage (language) Executed before every language patterns are loaded.
loadkernel (file) By default just defines a few basic commands. It can be used to define different versions of them or to load a file.
loadpatterns (patterns file) Loads the patterns file. Used by `luababel.def`.
loadexceptions (exceptions file) Loads the exceptions file. Used by `luababel.def`.

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

1.27 Languages supported by babel with ldf files

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

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton
Bulgarian bulgarian
Catalan catalan
Croatian croatian
Czech czech
Danish danish

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

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

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

```

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

```

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

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

1.28 Unicode character properties in luatex

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

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

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

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

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

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

1.29 Tweaking some features

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

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

1.30 Tips, workarounds, known issues and notes

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

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

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because T_EX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of T_EX, not of babel. Alternatively, you may use `\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 T_EX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²¹ But that is the easy part, because they don't require modifying the L^AT_EX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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

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

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with `language.dat`

\TeX and most engines based on it (pdf \TeX , xetex, ϵ - \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:

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.

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

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

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

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

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

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

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 `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras⟨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, sent 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.

²⁶But not removed, for backward compatibility.

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

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

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

<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 \TeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \TeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions{<lang>}</code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

```

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

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

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

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

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

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

\StartBabelCommands*{<language>}{date}

```



```

\SetString\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char` The internal macro `\initiate@active@char` is used in language definition files to instruct \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, `\csname`, the control sequence for which the meaning has to be saved.

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

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

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

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

`\save@sf@q` Sometimes it is necessary to preserve the `\spacefactor`. For this purpose the macro `\save@sf@q` is available. It takes one argument, saves the current `spacefactor`, executes the argument, and restores the `spacefactor`.

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

3.8 Encoding-dependent strings

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

Furthermore, it must be activated explicitly, with the package option `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (and except if forced as described

²⁷This mechanism was introduced by Bernd Raichle.

below). In other words, the old way of defining/switching strings still works and it's used by default.

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

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

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

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

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

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

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

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

²⁸In future releases further categories may be added.

```

\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\monthiiiname{März}

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

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


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

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

\EndBabelCommands

```

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

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

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

$\backslash EndBabelCommands$ 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 $\backslash EndBabelCommands$.

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

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

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

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

There are three helper macros to be used inside `\SetHyphenMap`:

- `\BabelLower{⟨ucode⟩}{⟨lcode⟩}` is similar to `\lcode` but it's ignored if the char has been set and saves the original lcode to restore it when switching the language (except with `hyphenmap=first`).
- `\BabelLowerMM{⟨ucode-from⟩}{⟨ucode-to⟩}{⟨step⟩}{⟨lcode-from⟩}` loops through the given uppercase codes, using the step, and assigns them the lcode, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{⟨ucode-from⟩}{⟨ucode-to⟩}{⟨step⟩}{⟨lcode⟩}` loops through the given uppercase codes, using the step, and assigns them the lcode, 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 an uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a lowercase case). There is an exception, however: the section counter s has been devised to have arbitrary keys, so you can add lowercased keys if you want.

7 Tools

```
1 <<version=3.61.2419>>
2 <<date=2021/06/30>>
```

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in \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` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³⁰. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```
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   \def\<##1>{\expandafter\<\noexpand\csname##1\endcsname}%
33   \edef\bbl@exp@aux{\endgroup#1}%
34   \bbl@exp@aux}
```

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

`\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.

```

35 \def\bbl@tempa#1{%
36   \long\def\bbl@trim##1##2{%
37     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38   \def\bbl@trim@c{%
39     \ifx\bbl@trim@a\@sptoken
40       \expandafter\bbl@trim@b
41     \else
42       \expandafter\bbl@trim@b\expandafter#1%
43     \fi}%
44   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
45 \bbl@tempa{ }
46 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}

```

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

```

48 \begingroup
49   \gdef\bbl@ifunset#1{%
50     \expandafter\ifx\csname#1\endcsname\relax
51       \expandafter\@firstoftwo
52     \else
53       \expandafter\@secondoftwo
54     \fi}
55 \bbl@ifunset{ifcsname}%
56 {}%
57 {\gdef\bbl@ifunset#1{%
58   \ifcsname#1\endcsname
59     \expandafter\ifx\csname#1\endcsname\relax
60       \bbl@afterelse\expandafter\@firstoftwo
61     \else
62       \bbl@afterfi\expandafter\@secondoftwo
63     \fi
64   \else
65     \expandafter\@firstoftwo
66   \fi}}
67 \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,

```

68 \def\bbl@ifblank#1{%
69   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bbl@ifset#1#2#3{%
72   \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).

```

73 \def\bbl@forkv#1#2{%
74   \def\bbl@kvcmd##1##2##3{#2}%
75   \bbl@kvnext#1,\@nil,}
76 \def\bbl@kvnext#1,{%

```

```

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

```

84 \def\bbl@vforeach#1#2{%
85 \def\bbl@forcmd##1{#2}%
86 \bbl@fornext#1,\@nil,}
87 \def\bbl@fornext#1,{%
88 \ifx\@nil#1\relax\else
89 \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
90 \expandafter\bbl@fornext
91 \fi}
92 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

\bbl@replace

```

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

```

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

```

105 \ifx\detokenize\@undefined\else % Unused macros if old Plain TeX
106 \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
107 \def\bbl@tempa{#1}%
108 \def\bbl@tempb{#2}%
109 \def\bbl@tempe{#3}}
110 \def\bbl@sreplace#1#2#3{%
111 \begingroup
112 \expandafter\bbl@parsedef\meaning#1\relax
113 \def\bbl@tempc{#2}%
114 \def\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
115 \def\bbl@tempd{#3}%
116 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
117 \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
118 \ifin@
119 \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
120 \def\bbl@tempc{% Expanded an executed below as 'uplevel'
121 \\makeatletter % "internal" macros with @ are assumed
122 \\scantokens{%
123 \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
124 \catcode64=\the\catcode64\relax}% Restore @

```

```

125     \else
126       \let\bbl@tempc\@empty % Not \relax
127     \fi
128     \bbl@exp{%      For the 'uplevel' assignments
129   \endgroup
130     \bbl@tempc}} % empty or expand to set #1 with changes
131 \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.

```

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

```

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

```

154 \def\bbl@bspack{%
155   \ifhmode
156     \hskip\z@skip
157     \def\bbl@espack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158   \else
159     \let\bbl@espack\@empty
160   \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`.

```

161 \def\bbl@cased{%
162   \ifx\oe\OE
163     \expandafter\in@\expandafter
164       {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166       \bbl@afterelse\expandafter\MakeUppercase
167     \else
168       \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170   \else
171     \expandafter\@firstofone
172   \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.

```

173 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
174   \toks@{\expandafter\expandafter\expandafter{%
175     \csname extras\language\endcsname}%
176     \bbl@exp{\in@{#1}{\the\toks@}}}%
177   \ifin\else
178     \@temptokena{#2}%
179     \edef\bbl@tempc{\the\@temptokena\the\toks@}%
180     \toks@\expandafter{\bbl@tempc#3}%
181     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
182   \fi}
183 <</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 .

```

184 <<*Make sure ProvidesFile is defined>> ≡
185 \ifx\ProvidesFile\@undefined
186   \def\ProvidesFile#1[#2 #3 #4]{%
187     \wlog{File: #1 #4 #3 <#2>}%
188     \let\ProvidesFile\@undefined}
189 \fi
190 <</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.

```

191 <<*Define core switching macros>> ≡
192 \ifx\language\@undefined
193   \csname newcount\endcsname\language
194 \fi
195 <</Define core switching macros>>

```

\last@language Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated.

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

```

196 <<*Define core switching macros>> ≡
197 <<*Define core switching macros>> ≡
198 \countdef\last@language=19 % TODO. why? remove?
199 \def\addlanguage{\csname newlanguage\endcsname}
200 <</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 or \LaTeX 2.09. 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`)

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. The first two options are for debugging.

```

201 (*package)
202 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
203 \ProvidesPackage{babel}[\langle\date\rangle\langle\version\rangle] The Babel package]
204 \@ifpackagewith{babel}{debug}
205   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
206   \let\bbl@debug\@firstofone
207   \ifx\directlua\@undefined\else
208     \directlua{ Babel = Babel or {}
209       Babel.debug = true }%
210   \fi}
211 {\providecommand\bbl@trace[1]{}%
212   \let\bbl@debug\@gobble
213   \ifx\directlua\@undefined\else
214     \directlua{ Babel = Babel or {}
215       Babel.debug = false }%
216   \fi}
217 \langle\Basic macros\rangle
218 % Temporarily repeat here the code for errors. TODO.
219 \def\bbl@error#1#2{%
220   \begingroup
221     \def\{\MessageBreak}%
222     \PackageError{babel}{#1}{#2}%
223   \endgroup}
224 \def\bbl@warning#1{%
225   \begingroup
226     \def\{\MessageBreak}%
227     \PackageWarning{babel}{#1}%
228   \endgroup}
229 \def\bbl@infowarn#1{%
230   \begingroup
231     \def\{\MessageBreak}%
232     \GenericWarning
233       {(babel) \@spaces\@spaces\@spaces}%
234       {Package babel Info: #1}%
235   \endgroup}
236 \def\bbl@info#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \PackageInfo{babel}{#1}%
240   \endgroup}
241 \def\bbl@nocaption{\protect\bbl@nocaption@i}
242 % TODO - Wrong for \today !!! Must be a separate macro.
243 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
244   \global\@namedef{#2}{\textbf{?#1?}}%
245   \@nameuse{#2}%
246   \edef\bbl@tempa{#1}%
247   \bbl@sreplace\bbl@tempa{name}{}}%
248   \bbl@warning{%
249     \@backslashchar#1 not set for '\language'. Please,\%
250     define it after the language has been loaded\%
251     (typically in the preamble) with\%
252     \string\setlocalecaption{\language}{\bbl@tempa}{..\}%
253     Reported}}
254 \def\bbl@tentative{\protect\bbl@tentative@i}
255 \def\bbl@tentative@i#1{%

```

```

256 \bbl@warning{%
257   Some functions for '#1' are tentative.\\%
258   They might not work as expected and their behavior\\%
259   may change in the future.\\%
260   Reported}}
261 \def\nolanerr#1{%
262   \bbl@error
263   {You haven't defined the language '#1' yet.\\%
264     Perhaps you misspelled it or your installation\\%
265     is not complete}%
266   {Your command will be ignored, type <return> to proceed}}
267 \def\nopatterns#1{%
268   \bbl@warning
269   {No hyphenation patterns were preloaded for\\%
270     the language '#1' into the format.\\%
271     Please, configure your TeX system to add them and\\%
272     rebuild the format. Now I will use the patterns\\%
273     preloaded for \bbl@nulllanguage\space instead}}
274   % End of errors
275 \@ifpackagewith{babel}{silent}
276   {\let\bbl@info@gobble
277    \let\bbl@infowarn@gobble
278    \let\bbl@warning@gobble}
279   {}
280 %
281 \def\AfterBabelLanguage#1{%
282   \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.

```

283 \ifx\bbl@languages\undefined\else
284   \begingroup
285     \catcode\^^I=12
286     \@ifpackagewith{babel}{showlanguages}{%
287       \begingroup
288         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
289         \wlog{<*languages>}%
290         \bbl@languages
291         \wlog{</languages>}%
292       \endgroup}{%
293     \endgroup
294     \def\bbl@elt#1#2#3#4{%
295       \ifnum#2=\z@
296         \gdef\bbl@nulllanguage{#1}%
297         \def\bbl@elt##1##2##3##4{}}%
298     \fi}%
299   \bbl@languages
300 \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`.

```

301 \bbl@trace{Defining option 'base'}
302 \@ifpackagewith{babel}{base}{%

```

```

303 \let\bbl@onlyswitch\@empty
304 \let\bbl@provide@locale\relax
305 \input babel.def
306 \let\bbl@onlyswitch\@undefined
307 \ifx\directlua\@undefined
308   \DeclareOption*{\bbl@patterns{\CurrentOption}}%
309 \else
310   \input luababel.def
311   \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
312 \fi
313 \DeclareOption{base}{}%
314 \DeclareOption{showlanguages}{}%
315 \ProcessOptions
316 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
317 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
318 \global\let\@ifl@ter@\@ifl@ter
319 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@}%
320 \endinput{}%
321% \end{macrocode}
322%
323% \subsection{\texttt{key=value} options and other general option}
324%
325%   The following macros extract language modifiers, and only real
326%   package options are kept in the option list. Modifiers are saved
327%   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
328%   no modifiers have been given, the former is |\relax|. How
329%   modifiers are handled are left to language styles; they can use
330%   |\in@|, loop them with |\@for| or load |keyval|, for example.
331%
332%   \begin{macrocode}
333\bbl@trace{key=value and another general options}
334\bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
335\def\bbl@tempb#1.#2{% Remove trailing dot
336  #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
337\def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
338  \ifx\@empty#2%
339    \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
340  \else
341    \in@{,provide=}{, #1}%
342    \ifin@
343      \edef\bbl@tempc{%
344        \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
345    \else
346      \in@{=}{ #1}%
347      \ifin@
348        \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
349      \else
350        \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
351        \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
352      \fi
353    \fi
354  \fi}
355\let\bbl@tempc\@empty
356\bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
357\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.

```

358 \DeclareOption{KeepShorthandsActive}{}
359 \DeclareOption{activeacute}{}
360 \DeclareOption{activegrave}{}
361 \DeclareOption{debug}{}
362 \DeclareOption{noconfigs}{}
363 \DeclareOption{showlanguages}{}
364 \DeclareOption{silent}{}
365 % \DeclareOption{mono}{}
366 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
367 \chardef\bbl@iniflag\z@
368 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
369 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
370 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
371 % A separate option
372 \let\bbl@autoload@options\@empty
373 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
374 % Don't use. Experimental. TODO.
375 \newif\ifbbl@single
376 \DeclareOption{selectors=off}{\bbl@singletrue}
377 <<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.

```

378 \let\bbl@opt@shorthands\@nnil
379 \let\bbl@opt@config\@nnil
380 \let\bbl@opt@main\@nnil
381 \let\bbl@opt@headfoot\@nnil
382 \let\bbl@opt@layout\@nnil
383 \let\bbl@opt@provide\@nnil

```

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

```

384 \def\bbl@tempa#1=#2\bbl@tempa{%
385   \bbl@csarg\ifx{opt@#1}\@nnil
386   \bbl@csarg\edef{opt@#1}{#2}%
387   \else
388   \bbl@error
389   {Bad option '#1=#2'. Either you have misspelled the\\%
390    key or there is a previous setting of '#1'. Valid\\%
391    keys are, among others, 'shorthands', 'main', 'bidi',\\%
392    'strings', 'config', 'headfoot', 'safe', 'math'.}%
393   {See the manual for further details.}
394   \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.

```

395 \let\bbl@language@opts\@empty
396 \DeclareOption*{%
397   \bbl@xin@{\string=}{\CurrentOption}%
398   \ifin@
399   \expandafter\bbl@tempa\CurrentOption\bbl@tempa
400   \else
401   \bbl@add@list\bbl@language@opts{\CurrentOption}%
402   \fi}

```

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

```

403 \ProcessOptions*

```



```

404 \ifx\bbbl@opt@provide\@nnil\else % Tests. Ignore.
405   \chardef\bbbl@iniflag\@ne
406   \bbbl@replace\bbbl@opt@provide{;}{,}
407   \bbbl@add\bbbl@opt@provide{,import}
408   \show\bbbl@opt@provide
409 \fi
410 %

```

7.4 Conditional loading of shorthands

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

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

```

411 \bbbl@trace{Conditional loading of shorthands}
412 \def\bbbl@sh@string#1{%
413   \ifx#1\@empty\else
414     \ifx#1t\string~%
415     \else\ifx#1c\string,%
416     \else\string#1%
417     \fi\fi
418   \expandafter\bbbl@sh@string
419   \fi}
420 \ifx\bbbl@opt@shorthands\@nnil
421   \def\bbbl@ifshorthand#1#2#3{#2}%
422 \else\ifx\bbbl@opt@shorthands\@empty
423   \def\bbbl@ifshorthand#1#2#3{#3}%
424 \else

```

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

```

425   \def\bbbl@ifshorthand#1{%
426     \bbbl@xin@\string#1}{\bbbl@opt@shorthands}%
427     \ifin@
428     \expandafter\@firstoftwo
429     \else
430     \expandafter\@secondoftwo
431     \fi}

```

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

```

432   \edef\bbbl@opt@shorthands{%
433     \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

434   \bbbl@ifshorthand{'}%
435   {\PassOptionsToPackage{activeacute}{babel}}{}
436   \bbbl@ifshorthand{`}%
437   {\PassOptionsToPackage{activegrave}{babel}}{}
438 \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.

```

439 \ifx\bbbl@opt@headfoot\@nnil\else
440   \g@addto@macro\@resetactivechars{%
441     \set@typeset@protect
442     \expandafter\select@language@x\expandafter{\bbbl@opt@headfoot}%
443     \let\protect\noexpand}
444 \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.

```

445 \ifx\bbl@opt@safe\undefined
446   \def\bbl@opt@safe{BR}
447 \fi
448 \ifx\bbl@opt@main\nil\else
449   \edef\bbl@language@opts{%
450     \ifx\bbl@language@opts\empty\else\bbl@language@opts,\fi
451     \bbl@opt@main}
452 \fi

```

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

```

453 \bbl@trace{Defining IfBabelLayout}
454 \ifx\bbl@opt@layout\nil
455   \newcommand\IfBabelLayout[3]{#3}%
456 \else
457   \newcommand\IfBabelLayout[1]{%
458     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
459     \ifin@
460       \expandafter\@firstoftwo
461     \else
462       \expandafter\@secondoftwo
463     \fi}
464 \fi

```

Common definitions. *In progress.* Still based on `babel.def`, but the code should be moved here.

```

465 \input babel.def

```

7.5 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.

```

466 <<More package options>> ≡
467 \DeclareOption{safe=none}{\let\bbl@opt@safe\empty}
468 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
469 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
470 <</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.

```

471 \bbl@trace{Cross referencing macros}
472 \ifx\bbl@opt@safe\empty\else
473   \def\@newl@bel#1#2#3{%
474     {\@safe@activestrue
475       \bbl@ifunset{#1@#2}%
476       \relax
477       {\gdef\@multiplelabels{%
478         \latex@warning@no@line{There were multiply-defined labels}}}%

```

```

479      \@latex@warning@no@line{Label `#2' multiply defined}}}%
480      \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.

```

481  \CheckCommand*\@testdef[3]{%
482    \def\reserved@a{#3}%
483    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
484      \else
485        \@tempswattrue
486        \fi}

```

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

```

487  \def\@testdef#1#2#3{%  TODO. With @samestring?
488    \@safe@activetrue
489    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
490    \def\bbl@tempb{#3}%
491    \@safe@activesfalse
492    \ifx\bbl@tempa\relax
493      \else
494        \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
495        \fi
496        \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
497        \ifx\bbl@tempa\bbl@tempb
498          \else
499            \@tempswattrue
500            \fi}
501  \fi

```

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

```

502  \bbl@xin@{R}\bbl@opt@safe
503  \ifin@
504    \bbl@redefineroast\ref#1{%
505      \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
506    \bbl@redefineroast\pageref#1{%
507      \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
508  \else
509    \let\org@ref\ref
510    \let\org@pageref\pageref
511  \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.

```

512  \bbl@xin@{B}\bbl@opt@safe
513  \ifin@
514    \bbl@redefine\@citex[#1]#2{%
515      \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
516      \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.

```
517 \AtBeginDocument{%
518   \@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.)

```
519   \def\@citex[#1][#2]#3{%
520     \@safe@activetrue\edef\@tempa{#3}\@safe@activetruefalse
521     \org@citex[#1][#2]{\@tempa}}%
522   }{}}
```

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

```
523 \AtBeginDocument{%
524   \@ifpackageloaded{cite}{%
525     \def\@citex[#1]#2{%
526       \@safe@activetrue\org@citex[#1][#2]\@safe@activetruefalse}%
527     }{}}
```

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

```
528 \bbl@redefine\nocite#1{%
529   \@safe@activetrue\org@nocite{#1}\@safe@activetruefalse}
```

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

```
530 \bbl@redefine\bibcite{%
531   \bbl@cite@choice
532   \bibcite}
```

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

```
533 \def\bbl@bibcite#1#2{%
534   \org@bibcite{#1}{\@safe@activetruefalse#2}}
```

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

```
535 \def\bbl@cite@choice{%
536   \global\let\bibcite\bbl@bibcite
537   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}}%
538   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}}%
539   \global\let\bbl@cite@choice\relax}
```

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

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

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

```

541 \bbl@redefine\@bibitem#1{%
542   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
543 \else
544   \let\org@nocite\nocite
545   \let\org@@citex\@citex
546   \let\org@bibcite\bibcite
547   \let\org@bibitem\@bibitem
548 \fi

```

7.6 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.

```

549 \bbl@trace{Marks}
550 \IfBabelLayout{sectioning}
551   {\ifx\bbl@opt@headfoot\@nnil
552     \g@addto@macro\resetactivechars{%
553       \set@typeset@protect
554       \expandafter\select@language@x\expandafter{\bbl@main@language}%
555       \let\protect\noexpand
556       \ifcase\bbl@bidimode\else % Only with bidi. See also above
557         \edef\thepage{%
558           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
559       \fi}%
560   \fi}
561 {\ifbbl@single\else
562   \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust
563   \markright#1{%
564     \bbl@ifblank{#1}%
565     {\org@markright{}}%
566     {\toks@{#1}%
567       \bbl@exp{%
568         \\org@markright{\\protect\\foreignlanguage{\language}%
569           {\protect\\bbl@restore@actives\the\toks@}}}%

```

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

`\@mkboth`

```

570   \ifx\@mkboth\markboth
571     \def\bbl@tempc{\let\@mkboth\markboth}
572   \else
573     \def\bbl@tempc{}
574   \fi
575   \bbl@ifunset{markboth} \bbl@redefine\bbl@redefineroobust
576   \markboth#1#2{%
577     \protected@edef\bbl@tempb##1{%
578       \protect\foreignlanguage
579       {\language}{\protect\bbl@restore@actives##1}%
580     \bbl@ifblank{#1}%
581     {\toks@{}}%

```

```

582      {\toks@expandafter{\bbl@tempb{#1}}}%
583      \bbl@ifblank{#2}%
584      {\@temptokena{}}%
585      {\@temptokena\expandafter{\bbl@tempb{#2}}}%
586      \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}%
587      \bbl@tempc
588      \fi} % end ifbbl@single, end \IfBabelLayout

```

7.7 Preventing clashes with other packages

7.7.1 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.

```

589 \bbl@trace{Preventing clashes with other packages}
590 \bbl@xin@{R}\bbl@opt@safe
591 \ifin@
592   \AtBeginDocument{%
593     \@ifpackageloaded{ifthen}{%
594       \bbl@redefine@long\ifthenelse#1#2#3{%
595         \let\bbl@temp@pref\pageref
596         \let\pageref\org@pageref
597         \let\bbl@temp@ref\ref
598         \let\ref\org@ref
599         \@safe@activestrue
600         \org@ifthenelse{#1}%
601         {\let\pageref\bbl@temp@pref
602          \let\ref\bbl@temp@ref
603          \@safe@activesfalse
604          #2}%
605         {\let\pageref\bbl@temp@pref
606          \let\ref\bbl@temp@ref
607          \@safe@activesfalse
608          #3}%
609       }%
610     }{}%
611   }

```

7.7.2 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`.

```

612   \AtBeginDocument{%
613     \@ifpackageloaded{varioref}{%

```

```

614 \bbl@redefine\@@vpageref#1[#2]#3{%
615 \@safe@activestrue
616 \org@@vpageref{#1}[#2]#3}%
617 \@safe@activesfalse}%
618 \bbl@redefine\hrefpagenum#1#2{%
619 \@safe@activestrue
620 \org@hrefpagenum{#1}#2}%
621 \@safe@activesfalse}%

```

The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref_` to call `\org@ref` instead of `\ref`. The disadvantage of this solution is that whenever the definition of `\Ref` changes, this definition needs to be updated as well.

```

622 \expandafter\def\csname Ref \endcsname#1{%
623 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
624 }{}%
625 }
626 \fi

```

7.7.3 `hhline`

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the ‘:’ character which is made active by the french support in `babel`. Therefore we need to *reload* the package when the ‘:’ is an active character. Note that this happens *after* the category code of the @-sign has been changed to other, so we need to temporarily change it to letter again.

```

627 \AtEndOfPackage{%
628 \AtBeginDocument{%
629 \ifpackageloaded{hhline}%
630 {\expandafter\ifx\csname normal@char\string\endcsname\relax
631 \else
632 \makeatletter
633 \def\@currname{hhline}\input{hhline.sty}\makeatother
634 \fi}%
635 {}}}

```

7.7.4 `hyperref`

`\pdfstringdefDisableCommands` A number of interworking problems between `babel` and `hyperref` are tackled by `hyperref` itself. The following code was introduced to prevent some annoying warnings but it broke bookmarks. This was quickly fixed in `hyperref`, which essentially made it no-op. However, it will not be removed for the moment because `hyperref` is expecting it. TODO. Still true? Commented out in 2020/07/27.

```

636 % \AtBeginDocument{%
637 % \ifx\pdfstringdefDisableCommands\@undefined\else
638 % \pdfstringdefDisableCommands{\languageshorthands{system}}%
639 % \fi}

```

7.7.5 `fancyhdr`

`\FOREIGNLANGUAGE` The package `fancyhdr` treats the running head and foot lines somewhat differently as the standard classes. A symptom of this is that the command `\foreignlanguage` which `babel` adds to the marks can end up inside the argument of `\MakeUppercase`. To prevent unexpected results we need to define `\FOREIGNLANGUAGE` here.

```

640 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
641 \lowercase{\foreignlanguage{#1}}}

```

`\substitutefontfamily` 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. This command is deprecated. Use the tools provides by \TeX .

```

642 \def\substitutefontfamily#1#2#3{%
643   \lowercase{\immediate\openout15=#1#2.fd\relax}%
644   \immediate\write15{%
645     \string\ProvidesFile{#1#2.fd}%
646     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
647     \space generated font description file]^J
648     \string\DeclareFontFamily{#1}{#2}{}}^J
649     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
650     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
651     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
652     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
653     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
654     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
655     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
656     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
657   }%
658   \closeout15
659 }
660 \@onlypreamble\substitutefontfamily

```

7.8 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. Unfortunately, `fontenc` deletes its package options, so we must guess which encodings has been loaded by traversing `\@filelist` to search for `(enc)enc.def`. 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`

```

661 \bbl@trace{Encoding and fonts}
662 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
663 \newcommand\BabelNonText{TS1,T3,TS3}
664 \let\org@TeX\TeX
665 \let\org@LaTeX\LaTeX
666 \let\ensureascii\@firstofone
667 \AtBeginDocument{%
668   \in@false
669   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
670     \ifin@false
671       \lowercase{\bbl@xin@{,#1enc.def},{,\@filelist,}}%
672       \fi}%
673   \ifin@ % if a text non-ascii has been loaded
674     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
675     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
676     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
677     \def\bbl@tempb#1@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
678     \def\bbl@tempc#1ENC.DEF#2\@{\%
679       \ifx\@empty#2\else
680         \bbl@ifunset{T#1}%
681         {}%
682         {\bbl@xin@{,#1,},{,\BabelNonASCII,\BabelNonText,}}%
683         \ifin@
684           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
685           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%

```



```

686         \else
687         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
688         \fi}%
689     \fi}%
690     \bbl@foreach\@filelist{\bbl@tempb#1@@}% TODO - @@ de mas??
691     \bbl@xin@{\,cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
692     \ifin@ \else
693     \edef\ensureascii#1{{%
694         \noexpand\fontencoding{cf@encoding}\noexpand\selectfont#1}}%
695     \fi
696 \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.

```

697 \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.

```

698 \AtBeginDocument{%
699     \@ifpackageloaded{fontspec}%
700     {\xdef\latinencoding{%
701         \ifx\UTFencname\@undefined
702         EU\ifcase\bbl@engine\or2\or1\fi
703         \else
704         \UTFencname
705         \fi}}%
706     {\gdef\latinencoding{OT1}%
707         \ifx\cf@encoding\bbl@t@one
708         \xdef\latinencoding{\bbl@t@one}%
709         \else
710         \ifx\@fontenc@load@list\@undefined
711         \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}}%
712         \else
713         \def\@elt#1{, #1,}%
714         \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
715         \let\@elt\relax
716         \bbl@xin@{, T1, }\bbl@tempa
717         \ifin@
718         \xdef\latinencoding{\bbl@t@one}%
719         \fi
720         \fi
721     \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.

```

722 \DeclareRobustCommand{\latintext}{%
723     \fontencoding{\latinencoding}\selectfont
724     \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.

```

725 \ifx\@undefined\DeclareTextFontCommand
726     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}

```

```

727 \else
728   \DeclareTextFontCommand{\textlatin}{\latintext}
729 \fi

```

7.9 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.

As a first step, add a handler for bidi and digits (and potentially other processes) just before `luaotfload` is applied, which is loaded by default by \LaTeX . Just in case, consider the possibility it has not been loaded.

```

730 \ifodd\bbl@engine
731   \def\bbl@activate@preotf{%
732     \let\bbl@activate@preotf\relax % only once
733     \directlua{
734       Babel = Babel or {}
735       %
736       function Babel.pre_otfload_v(head)
737         if Babel.numbers and Babel.digits_mapped then
738           head = Babel.numbers(head)
739         end
740         if Babel.bidi_enabled then
741           head = Babel.bidi(head, false, dir)
742         end
743         return head
744       end
745       %
746       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
747         if Babel.numbers and Babel.digits_mapped then
748           head = Babel.numbers(head)
749         end
750         if Babel.bidi_enabled then
751           head = Babel.bidi(head, false, dir)
752         end
753         return head
754       end
755       %
756       luatexbase.add_to_callback('pre_linebreak_filter',
757         Babel.pre_otfload_v,
758         'Babel.pre_otfload_v',

```

```

759     luatexbase.priority_in_callback('pre_linebreak_filter',
760     'luaotfload.node_processor') or nil)
761 %
762     luatexbase.add_to_callback('hpack_filter',
763     Babel.pre_otfload_h,
764     'Babel.pre_otfload_h',
765     luatexbase.priority_in_callback('hpack_filter',
766     'luaotfload.node_processor') or nil)
767 }}
768 \fi

```

The basic setup. In luatex, the output is modified at a very low level to set the \bodydir to the \pagedir.

```

769 \bbl@trace{Loading basic (internal) bidi support}
770 \ifodd\bbl@engine
771   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
772     \let\bbl@beforeforeign\leavevmode
773     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
774     \RequirePackage{luatexbase}
775     \bbl@activate@preotf
776     \directlua{
777       require('babel-data-bidi.lua')
778       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
779       require('babel-bidi-basic.lua')
780       \or
781       require('babel-bidi-basic-r.lua')
782     \fi}
783 % TODO - to locale_props, not as separate attribute
784 \newattribute\bbl@attr@dir
785 % TODO. I don't like it, hackish:
786 \bbl@exp{\output{\bodydir\pagedir\the\output}}
787 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
788 \fi\fi
789 \else
790   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
791     \bbl@error
792     {The bidi method 'basic' is available only in\\%
793     luatex. I'll continue with 'bidi=default', so\\%
794     expect wrong results}%
795     {See the manual for further details.}%
796     \let\bbl@beforeforeign\leavevmode
797     \AtEndOfPackage{%
798       \EnableBabelHook{babel-bidi}%
799       \bbl@xebidipar}
800   \fi\fi
801   \def\bbl@loadxebidi#1{%
802     \ifx\RTLfootnotetext\@undefined
803       \AtEndOfPackage{%
804         \EnableBabelHook{babel-bidi}%
805         \ifx\fontspec\@undefined
806           \bbl@loadfontspec % bidi needs fontspec
807         \fi
808         \usepackage#1{bidi}}%
809     \fi}
810   \ifnum\bbl@bidimode>200
811     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
812     \bbl@tentative{bidi=bidi}
813     \bbl@loadxebidi{}
814   \or

```

```

815     \bbl@loadxebidi{[rldocument]}
816   \or
817     \bbl@loadxebidi{}
818   \fi
819 \fi
820 \fi
821 \ifnum\bbl@bidimode=\@ne
822   \let\bbl@beforeforeign\leavevmode
823   \ifodd\bbl@engine
824     \newattribute\bbl@attr@dir
825     \bbl@exp{\output{\bodydir\pagedir\the\output}}%
826   \fi
827   \AtEndOfPackage{%
828     \EnableBabelHook{babel-bidi}%
829     \ifodd\bbl@engine\else
830       \bbl@xebidipar
831     \fi}
832 \fi

```

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

```

833 \bbl@trace{Macros to switch the text direction}
834 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
835 \def\bbl@rscripts{% TODO. Base on codes ??
836   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
837   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
838   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
839   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
840   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
841   Old South Arabian,}%
842 \def\bbl@provide@dirs#1{%
843   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
844   \ifin@
845     \global\bbl@csarg\chardef{wdir@#1}\@ne
846     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
847   \ifin@
848     \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
849   \fi
850   \else
851     \global\bbl@csarg\chardef{wdir@#1}\z@
852   \fi
853   \ifodd\bbl@engine
854     \bbl@csarg\ifcase{wdir@#1}%
855       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
856     \or
857       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
858     \or
859       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
860     \fi
861   \fi}
862 \def\bbl@switchdir{%
863   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
864   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
865   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
866 \def\bbl@setdirs#1{% TODO - math
867   \ifcase\bbl@select@type % TODO - strictly, not the right test
868     \bbl@bodydir{#1}%
869     \bbl@paddir{#1}%
870   \fi

```

```

871 \bbl@textdir{#1}}
872 % TODO. Only if \bbl@bidimode > 0?:
873 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
874 \DisableBabelHook{babel-bidi}

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

875 \ifodd\bbl@engine % luatex=1
876 \chardef\bbl@thetextdir\z@
877 \chardef\bbl@thepardir\z@
878 \def\bbl@getluadir#1{%
879   \directlua{
880     if tex.#1dir == 'TLT' then
881       tex.sprint('0')
882     elseif tex.#1dir == 'TRT' then
883       tex.sprint('1')
884     end}}
885 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
886   \ifcase#3\relax
887     \ifcase\bbl@getluadir{#1}\relax\else
888       #2 TLT\relax
889     \fi
890   \else
891     \ifcase\bbl@getluadir{#1}\relax
892       #2 TRT\relax
893     \fi
894   \fi}
895 \def\bbl@textdir#1{%
896   \bbl@setluadir{text}\textdir{#1}%
897   \chardef\bbl@thetextdir#1\relax
898   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
899 \def\bbl@pardir#1{%
900   \bbl@setluadir{par}\pardir{#1}%
901   \chardef\bbl@thepardir#1\relax}
902 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
903 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
904 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
905 % Sadly, we have to deal with boxes in math with basic.
906 % Activated every math with the package option bidi=:
907 \ifnum\bbl@bidimode>\z@
908   \def\bbl@mathboxdir{%
909     \ifcase\bbl@thetextdir\relax
910       \everyhbox{\bbl@mathboxdir@aux L}%
911     \else
912       \everyhbox{\bbl@mathboxdir@aux R}%
913     \fi}
914   \def\bbl@mathboxdir@aux#1{%
915     \@ifnextchar\egroup{}{\textdir T#1T\relax}}
916   \frozen@everymath\expandafter{%
917     \expandafter\bbl@mathboxdir\the\frozen@everymath}
918   \frozen@everydisplay\expandafter{%
919     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
920   \fi
921 \else % pdftex=0, xetex=2
922   \newcount\bbl@dirlevel
923   \chardef\bbl@thetextdir\z@
924   \chardef\bbl@thepardir\z@
925   \def\bbl@textdir#1{%
926     \ifcase#1\relax
927       \chardef\bbl@thetextdir\z@

```

```

928     \bbl@textdir@i\beginL\endL
929   \else
930     \chardef\bbl@thetextdir\@ne
931     \bbl@textdir@i\beginR\endR
932   \fi}
933 \def\bbl@textdir@i#1#2{%
934   \ifhmode
935     \ifnum\currentgrouplevel>\z@
936       \ifnum\currentgrouplevel=\bbl@dirlevel
937         \bbl@error{Multiple bidi settings inside a group}%
938         {I'll insert a new group, but expect wrong results.}%
939         \bgroup\aftergroup#2\aftergroup\egroup
940       \else
941         \ifcase\currentgrouptype\or % 0 bottom
942           \aftergroup#2% 1 simple {}
943         \or
944           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
945         \or
946           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
947         \or\or\or % vbox vtop align
948         \or
949           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
950         \or\or\or\or\or\or % output math disc insert vcent mathchoice
951         \or
952           \aftergroup#2% 14 \begingroup
953         \else
954           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
955         \fi
956       \fi
957       \bbl@dirlevel\currentgrouplevel
958     \fi
959   #1%
960   \fi}
961 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
962 \let\bbl@bodydir\@gobble
963 \let\bbl@pagedir\@gobble
964 \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).

```

965 \def\bbl@xebidipar{%
966   \let\bbl@xebidipar\relax
967   \TeXeTstate\@ne
968   \def\bbl@xeeverypar{%
969     \ifcase\bbl@thepardir
970       \ifcase\bbl@thetextdir\else\beginR\fi
971     \else
972       {\setbox\z@\lastbox\beginR\box\z@}%
973     \fi}%
974   \let\bbl@severypar\everypar
975   \newtoks\everypar
976   \everypar=\bbl@severypar
977   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
978 \ifnum\bbl@bidimode>200
979   \let\bbl@textdir@i\@gobbletwo
980   \let\bbl@xebidipar\@empty
981   \AddBabelHook{bidi}{foreign}{%
982     \def\bbl@tempa{\def\BabelText####1}%

```

```

983 \ifcase\bbl@thetextdir
984 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
985 \else
986 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
987 \fi}
988 \def\bbl@paddir#1{\ifcase#1\relax\setLR\else\setRL\fi}
989 \fi
990 \fi

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

991 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
992 \AtBeginDocument{%
993 \ifx\pdfstringdefDisableCommands\@undefined\else
994 \ifx\pdfstringdefDisableCommands\relax\else
995 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
996 \fi
997 \fi}

```

7.10 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`.

```

998 \bbl@trace{Local Language Configuration}
999 \ifx\loadlocalcfg\@undefined
1000 \@ifpackagewith{babel}{noconfigs}%
1001 {\let\loadlocalcfg\@gobble}%
1002 {\def\loadlocalcfg#1{%
1003 \InputIfFileExists{#1.cfg}%
1004 {\typeout{*****^J%
1005 * Local config file #1.cfg used^^J%
1006 *}}}%
1007 \@empty}}
1008 \fi

```

7.11 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).

```

1009 \bbl@trace{Language options}
1010 \let\bbl@afterlang\relax
1011 \let\BabelModifiers\relax
1012 \let\bbl@loaded\@empty
1013 \def\bbl@load@language#1{%
1014 \InputIfFileExists{#1.ldf}%
1015 {\edef\bbl@loaded{\CurrentOption
1016 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1017 \expandafter\let\expandafter\bbl@afterlang
1018 \csname\CurrentOption.ldf-h@@k\endcsname
1019 \expandafter\let\expandafter\BabelModifiers
1020 \csname bbl@mod@\CurrentOption\endcsname}%
1021 {\bbl@error{%
1022 Unknown option '\CurrentOption'. Either you misspelled it\\%
1023 or the language definition file \CurrentOption.ldf was not found}}%
1024 Valid options are, among others: shorthands=, KeepShorthandsActive,\\%

```

```

1025     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1026     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.

```

1027 \def\bbl@try@load@lang#1#2#3{%
1028   \IfFileExists{\CurrentOption.lda}%
1029   {\bbl@load@language{\CurrentOption}}}%
1030   {#1\bbl@load@language{#2}#3}}
1031 \DeclareOption{hebrew}{%
1032   \input{rlbabel.def}%
1033   \bbl@load@language{hebrew}}
1034 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1035 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1036 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1037 \DeclareOption{polutonikogreek}{%
1038   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}%
1039 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1040 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1041 \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 `.lda` 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.

```

1042 \ifx\bbl@opt@config\@nnil
1043   \@ifpackagewith{babel}{noconfigs}{}%
1044   {\InputIfFileExists{bblopts.cfg}%
1045     {\typeout{*****^J%
1046               * Local config file bblopts.cfg used^^J%
1047               *}}}%
1048   {}}%
1049 \else
1050   \InputIfFileExists{\bbl@opt@config.cfg}%
1051   {\typeout{*****^J%
1052             * Local config file \bbl@opt@config.cfg used^^J%
1053             *}}}%
1054   {\bbl@error{%
1055     Local config file '\bbl@opt@config.cfg' not found}{%
1056     Perhaps you misspelled it.}}}%
1057 \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.

```

1058 \let\bbl@tempc\relax
1059 \bbl@foreach\bbl@language@opts{%
1060   \ifcase\bbl@iniflag % Default
1061     \bbl@ifunset{ds@#1}%
1062     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1063     {}%
1064   \or % provide=*
1065     \@gobble % case 2 same as 1
1066   \or % provide+=*
1067     \bbl@ifunset{ds@#1}%
1068     {\IfFileExists{#1.lda}{%
1069       {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}}}%

```



```

1070     {}%
1071     \bbl@ifunset{ds@#1}%
1072     {\def\bbl@tempc{#1}%
1073      \DeclareOption{#1}{%
1074        \ifnum\bbl@iniflag>\@ne
1075          \bbl@ldfinit
1076          \babelprovide[import]{#1}%
1077          \bbl@afterldf{}%
1078        \else
1079          \bbl@load@language{#1}%
1080        \fi}}%
1081     {}%
1082 \or    % provide*=*
1083 \def\bbl@tempc{#1}%
1084 \bbl@ifunset{ds@#1}%
1085     {\DeclareOption{#1}{%
1086       \bbl@ldfinit
1087       \babelprovide[import]{#1}%
1088       \bbl@afterldf{}}}%
1089     {}%
1090 \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.

```

1091 \let\bbl@tempb\@nnil
1092 \bbl@foreach\@classoptionslist{%
1093   \bbl@ifunset{ds@#1}%
1094   {\IfFileExists{#1.ldf}%
1095    {\def\bbl@tempb{#1}%
1096     \DeclareOption{#1}{%
1097       \ifnum\bbl@iniflag>\@ne
1098         \bbl@ldfinit
1099         \babelprovide[import]{#1}%
1100         \bbl@afterldf{}%
1101       \else
1102         \bbl@load@language{#1}%
1103       \fi}}%
1104   {\IfFileExists{babel-#1.tex}% TODO. Copypaste pattern
1105    {\def\bbl@tempb{#1}%
1106     \DeclareOption{#1}{%
1107       \ifnum\bbl@iniflag>\@ne
1108         \bbl@ldfinit
1109         \babelprovide[import]{#1}%
1110         \bbl@afterldf{}%
1111       \else
1112         \bbl@load@language{#1}%
1113       \fi}}%
1114    {}}}%
1115   {}}

```

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

```

1116 \ifnum\bbl@iniflag=\z@\else
1117   \ifx\bbl@opt@main\@nnil
1118     \ifx\bbl@tempc\relax
1119       \let\bbl@opt@main\bbl@tempb
1120     \else
1121       \let\bbl@opt@main\bbl@tempc
1122     \fi

```

```

1123 \fi
1124 \fi
1125 \ifx\babel@opt@main\@nnil\else
1126 \expandafter
1127 \let\expandafter\babel@loadmain\csname ds@babel@opt@main\endcsname
1128 \expandafter\let\csname ds@babel@opt@main\endcsname\empty
1129 \fi

```

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

The options have to be processed in the order in which the user specified them (except, of course, global options, which \TeX processes before):

```

1130 \def\AfterBabelLanguage#1{%
1131 \babel@ifsamestring\CurrentOption{#1}{\global\babel@add\babel@afterlang}{}}
1132 \DeclareOption*{}
1133 \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.

```

1134 \babel@trace{Option 'main'}
1135 \ifx\babel@opt@main\@nnil
1136 \edef\babel@tempa{\@classoptionslist,\babel@language@opts}
1137 \let\babel@tempc\empty
1138 \babel@for\babel@tempb\babel@tempa{%
1139 \babel@xin@{\babel@tempb,}{,\babel@loaded,}%
1140 \ifin\edef\babel@tempc{\babel@tempb}\fi}
1141 \def\babel@tempa#1,#2\@nnil{\def\babel@tempb{#1}}
1142 \expandafter\babel@tempa\babel@loaded,\@nnil
1143 \ifx\babel@tempb\babel@tempc\else
1144 \babel@warning{%
1145 Last declared language option is '\babel@tempc',\%
1146 but the last processed one was '\babel@tempb'.\%
1147 The main language can't be set as both a global\%
1148 and a package option. Use 'main=\babel@tempc' as\%
1149 option. Reported}%
1150 \fi
1151 \else
1152 \ifodd\babel@iniflag % case 1,3
1153 \babel@ldfinit
1154 \let\CurrentOption\babel@opt@main
1155 \ifx\babel@opt@provide\@nnil
1156 \babel@exp{\@babelprovide[import,main]{\babel@opt@main}}
1157 \else
1158 \babel@exp{\@babelprovide[\babel@opt@provide,main]{\babel@opt@main}}%
1159 \fi
1160 \babel@afterldf{}%
1161 \else % case 0,2
1162 \chardef\babel@iniflag\z@ % Force ldf
1163 \expandafter\let\csname ds@babel@opt@main\endcsname\babel@loadmain
1164 \ExecuteOptions{\babel@opt@main}
1165 \DeclareOption*{}%
1166 \ProcessOptions*
1167 \fi
1168 \fi
1169 \def\AfterBabelLanguage{%
1170 \babel@error
1171 {Too late for \string\AfterBabelLanguage}%

```

```

1172   {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
\bbbl@main@language, has become defined. If not, no language has been loaded and an error
message is displayed.

1173 \ifx\bbbl@main@language\@undefined
1174   \bbbl@info{%
1175     You haven't specified a language. I'll use 'nil'\%
1176     as the main language. Reported}
1177   \bbbl@load@language{nil}
1178 \fi
1179 </package>
1180 <*core>

```

8 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.

8.1 Tools

```

1181 \ifx\ldf@quit\@undefined\else
1182 \endinput\fi % Same line!
1183 <<Make sure ProvidesFile is defined>>
1184 \ProvidesFile{babel.def}[(<date>)](<version>)[Babel common definitions]

```

The file babel.def expects some definitions made in the L^AT_EX 2_ε style file. So, In L^AT_EX 2.09 and 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).

```

1185 \ifx\AtBeginDocument\@undefined % TODO. change test.
1186   <<Emulate LaTeX>>
1187   \def\languagename{english}%
1188   \let\bbbl@opt@shorthands\@nnil
1189   \def\bbbl@ifshorthand#1#2#3{#2}%
1190   \let\bbbl@language@opts\@empty
1191   \ifx\babeloptionstrings\@undefined
1192     \let\bbbl@opt@strings\@nnil
1193   \else
1194     \let\bbbl@opt@strings\babeloptionstrings
1195   \fi
1196   \def\BabelStringsDefault{generic}
1197   \def\bbbl@tempa{normal}
1198   \ifx\babeloptionmath\bbbl@tempa
1199     \def\bbbl@mathnormal{\noexpand\textormath}
1200   \fi
1201   \def\AfterBabelLanguage#1#2{}
1202   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1203   \let\bbbl@afterlang\relax
1204   \def\bbbl@opt@safe{BR}

```

```

1205 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
1206 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1207 \expandafter\newif\csname ifbbl@single\endcsname
1208 \chardef\bbl@bidimode\z@
1209 \fi

```

Exit immediately with 2.09. An error is raised by the sty file, but also try to minimize the number of errors.

```

1210 \ifx\bbl@trace\@undefined
1211 \let\LdfInit\endinput
1212 \def\ProvidesLanguage#1\endinput}
1213 \endinput\fi % Same line!

```

And continue.

9 Multiple languages

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

Plain T_EX 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.

```

1214 <<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.

```

1215 \def\bbl@version{<<version>>}
1216 \def\bbl@date{<<date>>}
1217 \def\adddialect#1#2{%
1218   \global\chardef#1#2\relax
1219   \bbl@usehooks{adddialect}{#1}{#2}}%
1220 \begingroup
1221   \count#1\relax
1222   \def\bbl@elt##1##2###3###4{%
1223     \ifnum\count@=##2\relax
1224       \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
1225       \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
1226         set to \expandafter\string\csname l@##1\endcsname\\%
1227         (\string\language\the\count@). Reported}%
1228       \def\bbl@elt####1####2####3####4{}}%
1229   \fi}%
1230   \bbl@cs{languages}%
1231 \endgroup

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises and 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 intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```

1232 \def\bbl@fixname#1{%
1233   \begingroup
1234   \def\bbl@tempe{l@}%
1235   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1236   \bbl@tempd
1237   {\lowercase\expandafter\bbl@tempd}%
1238   {\uppercase\expandafter\bbl@tempd}%
1239   \@empty
1240   {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1241   {\uppercase\expandafter\bbl@tempd}}}%
1242   {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1243   {\lowercase\expandafter\bbl@tempd}}}%

```

```

1244 \empty
1245 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1246 \bbl@tempd
1247 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1248 \def\bbl@iflanguage#1{%
1249 \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.

```

1250 \def\bbl@bcpcase#1#2#3#4\@#5{%
1251 \ifx\empty#3%
1252 \uppercase{\def#5{#1#2}}%
1253 \else
1254 \uppercase{\def#5{#1}}%
1255 \lowercase{\edef#5{#5#2#3#4}}%
1256 \fi}
1257 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1258 \let\bbl@bcp\relax
1259 \lowercase{\def\bbl@tempa{#1}}%
1260 \ifx\empty#2%
1261 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1262 \else\ifx\empty#3%
1263 \bbl@bcpcase#2\empty\empty\@bbl@tempb
1264 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1265 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1266 {}%
1267 \ifx\bbl@bcp\relax
1268 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1269 \fi
1270 \else
1271 \bbl@bcpcase#2\empty\empty\@bbl@tempb
1272 \bbl@bcpcase#3\empty\empty\@bbl@tempc
1273 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1274 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1275 {}%
1276 \ifx\bbl@bcp\relax
1277 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1278 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1279 {}%
1280 \fi
1281 \ifx\bbl@bcp\relax
1282 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1283 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1284 {}%
1285 \fi
1286 \ifx\bbl@bcp\relax
1287 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1288 \fi
1289 \fi\fi}
1290 \let\bbl@initoload\relax
1291 \def\bbl@provide@locale{%
1292 \ifx\babelprovide\undefined
1293 \bbl@error{For a language to be defined on the fly 'base'\%
1294 is not enough, and the whole package must be\%
1295 loaded. Either delete the 'base' option or\%
1296 request the languages explicitly}%

```

```

1297             {See the manual for further details.}%
1298 \fi
1299 % TODO. Option to search if loaded, with \LocaleForEach
1300 \let\bbl@auxname\language % Still necessary. TODO
1301 \bbl@ifunset{\bbl@bcp@map@\language}{}% Move uplevel??
1302 { \edef\language{\@nameuse{\bbl@bcp@map@\language}}}%
1303 \ifbbl@bcp@allowed
1304 \expandafter\ifx\csname date\language\endcsname\relax
1305 \expandafter
1306 \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@
1307 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
1308 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1309 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1310 \expandafter\ifx\csname date\language\endcsname\relax
1311 \let\bbl@initoload\bbl@bcp
1312 \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1313 \let\bbl@initoload\relax
1314 \fi
1315 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1316 \fi
1317 \fi
1318 \fi
1319 \expandafter\ifx\csname date\language\endcsname\relax
1320 \IfFileExists{babel-\language.tex}%
1321 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
1322 {}%
1323 \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.

```

1324 \def\iflanguage#1{%
1325 \bbl@iflanguage{#1}{%
1326 \ifnum\csname l@#1\endcsname=\language
1327 \expandafter\@firstoftwo
1328 \else
1329 \expandafter\@secondoftwo
1330 \fi}}

```

9.1 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.

```

1331 \let\bbl@select@type\z@
1332 \edef\selectlanguage{%
1333 \noexpand\protect
1334 \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`.

```
1335 \ifx\@undefined\protect\let\protect\relax\fi
```

The following definition is preserved for backwards compatibility. It is related to a trick for 2.09.

```
1336 \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.

```
1337 \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`

```
1338 \def\bbl@push@language{%
1339   \ifx\language\@undefined\else
1340     \ifx\currentgrouplevel\@undefined
1341       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1342     \else
1343       \ifnum\currentgrouplevel=\z@
1344         \xdef\bbl@language@stack{\language+}%
1345       \else
1346         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1347       \fi
1348     \fi
1349   \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`.

```
1350 \def\bbl@pop@lang#1+#2\@{%
1351   \edef\language{#1}%
1352   \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).

```
1353 \let\bbl@ifrestoring\@secondoftwo
1354 \def\bbl@pop@language{%
1355   \expandafter\bbl@pop@lang\bbl@language@stack\@
1356   \let\bbl@ifrestoring\@firstoftwo
1357   \expandafter\bbl@set@language\expandafter{\language}%
1358   \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).

```
1359 \chardef\localeid\z@
1360 \def\bbl@id@last{0}    % No real need for a new counter
1361 \def\bbl@id@assign{%
```

```

1362 \bbl@ifunset{\bbl@id@\language}%
1363 {\count@\bbl@id@last\relax
1364 \advance\count@\@ne
1365 \bbl@csarg\chardef{id@\language}\count@
1366 \edef\bbl@id@last{\the\count@}%
1367 \ifcase\bbl@engine\or
1368 \directlua{
1369     Babel = Babel or {}
1370     Babel.locale_props = Babel.locale_props or {}
1371     Babel.locale_props[\bbl@id@last] = {}
1372     Babel.locale_props[\bbl@id@last].name = '\language'
1373 }%
1374 \fi}%
1375 }%
1376 \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

1377 \expandafter\def\csname selectlanguage \endcsname#1{%
1378 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
1379 \bbl@push@language
1380 \aftergroup\bbl@pop@language
1381 \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).

```

1382 \def\BabelContentsFiles{toc,lof,lot}
1383 \def\bbl@set@language#1{% from selectlanguage, pop@
1384 % The old buggy way. Preserved for compatibility.
1385 \edef\language{%
1386 \ifnum\escapechar=\expandafter`\string#1\@empty
1387 \else\string#1\@empty\fi}%
1388 \ifcat\relax\noexpand#1%
1389 \expandafter\ifx\csname date\language\endcsname\relax
1390 \edef\language{#1}%
1391 \let\localename\language
1392 \else
1393 \bbl@info{Using '\string\language' instead of 'language' is%%
1394 deprecated. If what you want is to use a%%
1395 macro containing the actual locale, make%%
1396 sure it does not not match any language.%%
1397 Reported}%
1398 \ifx\scantokens\@undefined
1399 \def\localename{??}%
1400 \else
1401 \scantokens\expandafter{\expandafter
1402 \def\expandafter\localename\expandafter{\language}}%
1403 \fi
1404 \fi
1405 \else

```



```

1406 \def\localename{#1}% This one has the correct catcodes
1407 \fi
1408 \select@language{\language}%
1409 % write to auxs
1410 \expandafter\ifx\csname date\language\endcsname\relax\else
1411 \if@filesw
1412 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1413 \bbl@savelastskip
1414 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1415 \bbl@restorelastskip
1416 \fi
1417 \bbl@usehooks{write}{}}%
1418 \fi
1419 \fi}
1420 %
1421 \let\bbl@restorelastskip\relax
1422 \def\bbl@savelastskip{%
1423 \let\bbl@restorelastskip\relax
1424 \ifvmode
1425 \ifdim\lastskip=\z@
1426 \let\bbl@restorelastskip\nobreak
1427 \else
1428 \bbl@exp{%
1429 \def\\bbl@restorelastskip{%
1430 \skip@=\the\lastskip
1431 \\nobreak \vskip-\skip@ \vskip\skip@}}%
1432 \fi
1433 \fi}
1434 %
1435 \newif\ifbbl@bcpallowed
1436 \bbl@bcpallowedfalse
1437 \def\select@language#1{% from set@, babel@aux
1438 % set hmap
1439 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1440 % set name
1441 \edef\language{#1}%
1442 \bbl@fixname\language
1443 % TODO. name@map must be here?
1444 \bbl@provide@locale
1445 \bbl@iflanguage\language{%
1446 \expandafter\ifx\csname date\language\endcsname\relax
1447 \bbl@error
1448 {Unknown language '\language'. Either you have\\%
1449 misspelled its name, it has not been installed,\\%
1450 or you requested it in a previous run. Fix its name,\\%
1451 install it or just rerun the file, respectively. In\\%
1452 some cases, you may need to remove the aux file}%
1453 {You may proceed, but expect wrong results}%
1454 \else
1455 % set type
1456 \let\bbl@select@type\z@
1457 \expandafter\bbl@switch\expandafter{\language}%
1458 \fi}}
1459 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1460 \select@language{#1}%
1461 \bbl@foreach\BabelContentsFiles{%
1462 \@writefile{##1}{\babel@toc{#1}{#2}}}% %% TODO - ok in plain?
1463 \def\babel@toc#1#2{%
1464 \select@language{#1}}

```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring \TeX in a certain pre-defined state. The name of the language is stored in the control sequence `\language`. Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros. The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\<lang>hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\<lang>hyphenmins` will be used.

```

1465 \newif\ifbbl@usedategroup
1466 \def\bbl@switch#1{% from select@, foreign@
1467 % make sure there is info for the language if so requested
1468 \bbl@ensureinfo{#1}%
1469 % restore
1470 \originalTeX
1471 \expandafter\def\expandafter\originalTeX\expandafter{%
1472 \csname noextras#1\endcsname
1473 \let\originalTeX\empty
1474 \babel@beginsave}%
1475 \bbl@usehooks{afterreset}{}%
1476 \languageshorthands{none}%
1477 % set the locale id
1478 \bbl@id@assign
1479 % switch captions, date
1480 % No text is supposed to be added here, so we remove any
1481 % spurious spaces.
1482 \bbl@bsphack
1483 \ifcase\bbl@select@type
1484 \csname captions#1\endcsname\relax
1485 \csname date#1\endcsname\relax
1486 \else
1487 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1488 \ifin@
1489 \csname captions#1\endcsname\relax
1490 \fi
1491 \bbl@xin@{,date,}{, \bbl@select@opts,}%
1492 \ifin@ % if \foreign... within \<lang>date
1493 \csname date#1\endcsname\relax
1494 \fi
1495 \fi
1496 \bbl@esphack
1497 % switch extras
1498 \bbl@usehooks{beforeextras}{}%
1499 \csname extras#1\endcsname\relax
1500 \bbl@usehooks{afterextras}{}%
1501 % > babel-ensure
1502 % > babel-sh-<short>
1503 % > babel-bidi
1504 % > babel-fontspec
1505 % hyphenation - case mapping
1506 \ifcase\bbl@opt@hyphenmap\or
1507 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1508 \ifnum\bbl@hymapsel>4\else
1509 \csname\language @bbl@hyphenmap\endcsname
1510 \fi

```

```

1511 \chardef\bbbl@opt@hyphenmap\z@
1512 \else
1513 \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
1514 \csname\language @bbbl@hyphenmap\endcsname
1515 \fi
1516 \fi
1517 \let\bbbl@hymapsel\@cclv
1518 % hyphenation - select rules
1519 \ifnum\csname l@\language\endcsname=\l@unhyphenated
1520 \edef\bbbl@tempa{u}%
1521 \else
1522 \edef\bbbl@tempa{\bbbl@cl{lnbrk}}%
1523 \fi
1524 % linebreaking - handle u, e, k (v in the future)
1525 \bbbl@xin@{/u}{/\bbbl@tempa}%
1526 \ifin@else\bbbl@xin@{/e}{/\bbbl@tempa}\fi % elongated forms
1527 \ifin@else\bbbl@xin@{/k}{/\bbbl@tempa}\fi % only kashida
1528 \ifin@else\bbbl@xin@{/v}{/\bbbl@tempa}\fi % variable font
1529 \ifin@
1530 % unhyphenated/kashida/elongated = allow stretching
1531 \language\l@unhyphenated
1532 \babel@savevariable\emergencystretch
1533 \emergencystretch\maxdimen
1534 \babel@savevariable\hbadness
1535 \hbadness\@M
1536 \else
1537 % other = select patterns
1538 \bbbl@patterns{#1}%
1539 \fi
1540 % hyphenation - mins
1541 \babel@savevariable\lefthyphenmin
1542 \babel@savevariable\righthyphenmin
1543 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1544 \set@hyphenmins\tw@\thr@\relax
1545 \else
1546 \expandafter\expandafter\expandafter\set@hyphenmins
1547 \csname #1hyphenmins\endcsname\relax
1548 \fi}

```

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

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

```

1549 \long\def\otherlanguage#1{%
1550 \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@\fi
1551 \csname selectlanguage \endcsname{#1}%
1552 \ignorespaces}

```

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

```

1553 \long\def\endotherlanguage{%
1554 \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`.

```

1555 \expandafter\def\csname otherlanguage*\endcsname{%

```

```

1556 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}]
1557 \def\bbl@otherlanguage@s[#1]#2{%
1558 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1559 \def\bbl@select@opts{#1}%
1560 \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”.

```

1561 \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`.

```

1562 \providecommand\bbl@beforeforeign{}
1563 \edef\foreignlanguage{%
1564 \noexpand\protect
1565 \expandafter\noexpand\csname foreignlanguage \endcsname}
1566 \expandafter\def\csname foreignlanguage \endcsname{%
1567 \@ifstar\bbl@foreign@s\bbl@foreign@x}
1568 \providecommand\bbl@foreign@x[3][]{%
1569 \begingroup
1570 \def\bbl@select@opts{#1}%
1571 \let\BabelText\@firstofone
1572 \bbl@beforeforeign
1573 \foreign@language{#2}%
1574 \bbl@usehooks{foreign}{}}%
1575 \BabelText{#3}% Now in horizontal mode!
1576 \endgroup}
1577 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1578 \begingroup
1579 {\par}%
1580 \let\bbl@select@opts\@empty
1581 \let\BabelText\@firstofone
1582 \foreign@language{#1}%
1583 \bbl@usehooks{foreign*}{}}%
1584 \bbl@dirparastext
1585 \BabelText{#2}% Still in vertical mode!
1586 {\par}%
1587 \endgroup}

```

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

```

1588 \def\foreign@language#1{%
1589   % set name
1590   \edef\language#1}%
1591   \ifbbl@usedategroup
1592     \bbl@add\bbl@select@opts{,date,}%
1593     \bbl@usedategroupfalse
1594   \fi
1595   \bbl@fixname\language
1596   % TODO. name@map here?
1597   \bbl@provide@locale
1598   \bbl@iflanguage\language{%
1599     \expandafter\ifx\csname date\language\endcsname\relax
1600       \bbl@warning % TODO - why a warning, not an error?
1601       {Unknown language '#1'. Either you have\\%
1602        misspelled its name, it has not been installed,\\%
1603        or you requested it in a previous run. Fix its name,\\%
1604        install it or just rerun the file, respectively. In\\%
1605        some cases, you may need to remove the aux file.\\%
1606        I'll proceed, but expect wrong results.\\%
1607        Reported}%
1608     \fi
1609     % set type
1610     \let\bbl@select@type\@ne
1611     \expandafter\bbl@switch\expandafter{\language}}

```

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

It also sets hyphenation exceptions, but only once, because they are global (here language `\lcode'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.

```

1612 \let\bbl@hyphlist\@empty
1613 \let\bbl@hyphenation@\relax
1614 \let\bbl@pttnlist\@empty
1615 \let\bbl@patterns@\relax
1616 \let\bbl@hymapsel=\@cclv
1617 \def\bbl@patterns#1{%
1618   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1619     \csname l@#1\endcsname
1620     \edef\bbl@tempa{#1}%
1621   \else
1622     \csname l@#1:\f@encoding\endcsname
1623     \edef\bbl@tempa{#1:\f@encoding}%
1624   \fi
1625   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1626   % > luatex
1627   \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1628     \begingroup
1629     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1630     \ifin\else
1631       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
1632     \hyphenation{%
1633       \bbl@hyphenation@

```

```

1634 \ifundefined{bbl@hyphenation@#1}%
1635 \empty
1636 {\space\csname bbl@hyphenation@#1\endcsname}}%
1637 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1638 \fi
1639 \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*`.

```

1640 \def\hyphenrules#1{%
1641 \edef\bbl@tempf{#1}%
1642 \bbl@fixname\bbl@tempf
1643 \bbl@iflanguage\bbl@tempf{%
1644 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1645 \ifx\languageshortands\undefined\else
1646 \languageshortands{none}%
1647 \fi
1648 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1649 \set@hyphenmins\tw@\thr@@\relax
1650 \else
1651 \expandafter\expandafter\expandafter\set@hyphenmins
1652 \csname\bbl@tempf hyphenmins\endcsname\relax
1653 \fi}}
1654 \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.

```

1655 \def\providehyphenmins#1#2{%
1656 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1657 \@namedef{#1hyphenmins}{#2}%
1658 \fi}

```

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

```

1659 \def\set@hyphenmins#1#2{%
1660 \lefthyphenmin#1\relax
1661 \righthyphenmin#2\relax}

```

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

```

1662 \ifx\ProvidesFile\undefined
1663 \def\ProvidesLanguage#1[#2 #3 #4]{%
1664 \wlog{Language: #1 #4 #3 <#2>}%
1665 }
1666 \else
1667 \def\ProvidesLanguage#1{%
1668 \begingroup
1669 \catcode`\ 10 %
1670 \@makeother\/%
1671 \@ifnextchar[%]
1672 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1673 \def\@provideslanguage#1[#2]{%
1674 \wlog{Language: #1 #2}%

```

```

1675 \expandafter\edef\csname ver@#1.1df\endcsname{#2}%
1676 \endgroup}
1677 \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`.

```

1678 \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.

```

1679 \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’:

```

1680 \providecommand\setlocale{%
1681 \bbl@error
1682 {Not yet available}%
1683 {Find an armchair, sit down and wait}}
1684 \let\uselocale\setlocale
1685 \let\locale\setlocale
1686 \let\selectlocale\setlocale
1687 \let\localename\setlocale
1688 \let\textlocale\setlocale
1689 \let\textlanguage\setlocale
1690 \let\languagetext\setlocale

```

9.2 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.

```

1691 \edef\bbl@nulllanguage{\string\language=0}
1692 \ifx\PackageError\undefined % TODO. Move to Plain
1693 \def\bbl@error#1#2{%
1694 \begingroup
1695 \newlinechar=^^J
1696 \def^^J(babel) }%
1697 \errhelp{#2}\errmessage{\#1}%
1698 \endgroup}
1699 \def\bbl@warning#1{%
1700 \begingroup
1701 \newlinechar=^^J
1702 \def^^J(babel) }%
1703 \message{\#1}%
1704 \endgroup}
1705 \let\bbl@infowarn\bbl@warning
1706 \def\bbl@info#1{%
1707 \begingroup
1708 \newlinechar=^^J
1709 \def^^J}%
1710 \wlog{#1}%

```

```

1711 \endgroup}
1712 \fi
1713 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1714 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1715 \global\@namedef{#2}{\textbf{?#1?}}%
1716 \@nameuse{#2}%
1717 \edef\bbl@tempa{#1}%
1718 \bbl@sreplace\bbl@tempa{name}{}%
1719 \bbl@warning{% TODO.
1720 \@backslashchar#1 not set for '\language'. Please,\%
1721 define it after the language has been loaded\%
1722 (typically in the preamble) with:\%
1723 \string\setlocalecaption{\language}{\bbl@tempa}{..\%
1724 Reported}}
1725 \def\bbl@tentative{\protect\bbl@tentative@i}
1726 \def\bbl@tentative@i#1{%
1727 \bbl@warning{%
1728 Some functions for '#1' are tentative.\%
1729 They might not work as expected and their behavior\%
1730 could change in the future.\%
1731 Reported}}
1732 \def\@nolanerr#1{%
1733 \bbl@error
1734 {You haven't defined the language '#1' yet.\%
1735 Perhaps you misspelled it or your installation\%
1736 is not complete}%
1737 {Your command will be ignored, type <return> to proceed}}
1738 \def\@nopatterns#1{%
1739 \bbl@warning
1740 {No hyphenation patterns were preloaded for\%
1741 the language '#1' into the format.\%
1742 Please, configure your TeX system to add them and\%
1743 rebuild the format. Now I will use the patterns\%
1744 preloaded for \bbl@nulllanguage\space instead}}
1745 \let\bbl@usehooks\@gobbletwo
1746 \ifx\bbl@onlyswitch\@empty\endinput\fi
1747 % Here ended switch.def

Here ended switch.def.

1748 \ifx\directlua\@undefined\else
1749 \ifx\bbl@luapatterns\@undefined
1750 \input luababel.def
1751 \fi
1752 \fi
1753 <<Basic macros>>
1754 \bbl@trace{Compatibility with language.def}
1755 \ifx\bbl@languages\@undefined
1756 \ifx\directlua\@undefined
1757 \openin1 = language.def % TODO. Remove hardcoded number
1758 \ifeof1
1759 \closein1
1760 \message{I couldn't find the file language.def}
1761 \else
1762 \closein1
1763 \begingroup
1764 \def\addlanguage#1#2#3#4#5{%
1765 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1766 \global\expandafter\let\csname l@#1\endcsname
1767 \csname lang@#1\endcsname

```



```

1768      \fi}%
1769      \def\uselanguage#1{%
1770      \input language.def
1771      \endgroup
1772      \fi
1773      \fi
1774      \chardef\l@english\z@
1775      \fi

```

`\addto` It takes two arguments, a *control sequence* and T_EX-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.

```

1776 \def\addto#1#2{%
1777   \ifx#1\undefined
1778     \def#1{#2}%
1779   \else
1780     \ifx#1\relax
1781       \def#1{#2}%
1782     \else
1783       {\toks@\expandafter{#1#2}%
1784        \xdef#1{\the\toks@}}%
1785     \fi
1786   \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?

```

1787 \def\bbl@withactive#1#2{%
1788   \begingroup
1789   \lccode`~=#2\relax
1790   \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 T_EX 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`.

```

1791 \def\bbl@redefine#1{%
1792   \edef\bbl@tempa{\bbl@stripslash#1}%
1793   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1794   \expandafter\def\csname\bbl@tempa\endcsname{
1795   \@onlypreamble\bbl@redefine

```

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

```

1796 \def\bbl@redefine@long#1{%
1797   \edef\bbl@tempa{\bbl@stripslash#1}%
1798   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1799   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1800   \@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_`.

```

1801 \def\bbl@redefineroobust#1{%
1802   \edef\bbl@tempa{\bbl@stripslash#1}%
1803   \bbl@ifunset{\bbl@tempa\space}%
1804   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%

```

```

1805 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1806 {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1807 \@namedef{\bbl@tempa\space}}
1808 \@onlypreamble\bbl@redefineroobust

```

9.3 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.

```

1809 \bbl@trace{Hooks}
1810 \newcommand\AddBabelHook[3][{}]{%
1811 \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1812 \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1813 \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1814 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1815 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1816 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1817 \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1818 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1819 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1820 \def\bbl@usehooks#1#2{%
1821 \def\bbl@elth##1{%
1822 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}}%
1823 \bbl@cs{ev@#1@}%
1824 \ifx\language\@undefined\else % Test required for Plain (?)
1825 \def\bbl@elth##1{%
1826 \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}}%
1827 \bbl@cl{ev@#1}%
1828 \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).

```

1829 \def\bbl@evargs{,% <- don't delete this comma
1830 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1831 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1832 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1833 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1834 beforestart=0,language=2}

```

`\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.

```

1835 \bbl@trace{Defining babelensure}
1836 \newcommand\babelensure[2][{}]{% TODO - revise test files
1837 \AddBabelHook{babel-ensure}{afterextras}{%
1838 \ifcase\bbl@select@type
1839 \bbl@cl{e}%
1840 \fi}%
1841 \begingroup
1842 \let\bbl@ens@include\@empty

```

```

1843 \let\bbl@ens@exclude\@empty
1844 \def\bbl@ens@fontenc{\relax}%
1845 \def\bbl@tempb##1{%
1846   \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1847 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1848 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1849 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1850 \def\bbl@tempc{\bbl@ensure}%
1851 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1852   \expandafter{\bbl@ens@include}}%
1853 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1854   \expandafter{\bbl@ens@exclude}}%
1855 \toks@\expandafter{\bbl@tempc}%
1856 \bbl@exp{%
1857 \endgroup
1858 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1859 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1860 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1861   \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1862     \edef##1{\noexpand\bbl@nocaption
1863       {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1864   \fi
1865   \ifx##1\@empty\else
1866     \in@{##1}{#2}%
1867     \ifin@\else
1868       \bbl@ifunset{\bbl@ensure@\language\name}%
1869       {\bbl@exp{%
1870         \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1871           \\\foreignlanguage{\language\name}%
1872             {\ifx\relax#3\else
1873               \\\fontencoding{#3}\selectfont
1874               \fi
1875               #####1}}}%
1876       }%
1877       \toks@\expandafter{##1}%
1878       \edef##1{%
1879         \bbl@csarg\noexpand{\ensure@\language\name}%
1880         {\the\toks@}}%
1881       \fi
1882       \expandafter\bbl@tempb
1883     \fi}%
1884 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1885 \def\bbl@tempa##1{% elt for include list
1886   \ifx##1\@empty\else
1887     \bbl@csarg\in@{\ensure@\language\name\expandafter}\expandafter{##1}%
1888     \ifin@\else
1889       \bbl@tempb##1\@empty
1890     \fi
1891     \expandafter\bbl@tempa
1892   \fi}%
1893 \bbl@tempa#1\@empty}
1894 \def\bbl@captionslist{%
1895 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1896 \contentsname\listfigurename\listtablename\indexname\figurename
1897 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1898 \alsoname\proofname\glossaryname}

```

9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1899 \bbl@trace{Macros for setting language files up}
1900 \def\bbl@ldfinit{%
1901   \let\bbl@screset\@empty
1902   \let\BabelStrings\bbl@opt@string
1903   \let\BabelOptions\@empty
1904   \let\BabelLanguages\relax
1905   \ifx\originalTeX\@undefined
1906     \let\originalTeX\@empty
1907   \else
1908     \originalTeX
1909   \fi}
1910 \def\LdfInit#1#2{%
1911   \chardef\atcatcode=\catcode`\@
1912   \catcode`\@=11\relax
1913   \chardef\eqcatcode=\catcode`\=
1914   \catcode`\==12\relax
1915   \expandafter\if\expandafter\@backslashchar
1916     \expandafter\@car\string#2\@nil
1917     \ifx#2\@undefined\else
1918       \ldf@quit{#1}%
1919     \fi
1920   \else
1921     \expandafter\ifx\csname#2\endcsname\relax\else
1922       \ldf@quit{#1}%
1923     \fi
1924   \fi
1925   \bbl@ldfinit}

```

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

```

1926 \def\ldf@quit#1{%
1927   \expandafter\main@language\expandafter{#1}%
1928   \catcode`\@=\atcatcode \let\atcatcode\relax
1929   \catcode`\==\eqcatcode \let\eqcatcode\relax
1930   \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.

```

1931 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1932   \bbl@afterlang
1933   \let\bbl@afterlang\relax
1934   \let\BabelModifiers\relax
1935   \let\bbl@screset\relax}%
1936 \def\ldf@finish#1{%
1937   \ifx\loadlocalcfg@undefined\else % For LaTeX 209
1938     \loadlocalcfg{#1}%
1939   \fi
1940   \bbl@afterldf{#1}%
1941   \expandafter\main@language\expandafter{#1}%
1942   \catcode`\@=\atcatcode \let\atcatcode\relax
1943   \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 .

```

1944 \@onlypreamble\LdfInit
1945 \@onlypreamble\ldf@quit
1946 \@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.

```

1947 \def\main@language#1{%
1948   \def\bbl@main@language{#1}%
1949   \let\language\bbl@main@language % TODO. Set localename
1950   \bbl@id@assign
1951   \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.

```

1952 \def\bbl@beforestart{%
1953   \def\@nolanerr##1{%
1954     \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1955   \bbl@usehooks{beforestart}{}%
1956   \global\let\bbl@beforestart\relax}
1957 \AtBeginDocument{%
1958   {\@nameuse{bbl@beforestart}}% Group!
1959   \if@filesw
1960     \providecommand\babel@aux[2]{}%
1961     \immediate\write\@mainaux{%
1962       \string\providecommand\string\babel@aux[2]{}%
1963       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1964   \fi
1965   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1966   \ifbbl@single % must go after the line above.
1967     \renewcommand\selectlanguage[1]{}%
1968     \renewcommand\foreignlanguage[2]{#2}%
1969     \global\let\babel@aux@gobbletwo % Also as flag
1970   \fi
1971   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1972 \def\select@language@x#1{%
1973   \ifcase\bbl@select@type

```

```

1974 \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1975 \else
1976 \select@language{#1}%
1977 \fi}

```

9.5 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.

```

1978 \bbl@trace{Shorhands}
1979 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1980 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1981 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1982 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1983 \begingroup
1984 \catcode`#1\active
1985 \nfss@catcodes
1986 \ifnum\catcode`#1=\active
1987 \endgroup
1988 \bbl@add\nfss@catcodes{\@makeother#1}%
1989 \else
1990 \endgroup
1991 \fi
1992 \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.

```

1993 \def\bbl@remove@special#1{%
1994 \begingroup
1995 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1996 \else\noexpand##1\noexpand##2\fi}%
1997 \def\do{\x\do}%
1998 \def\@makeother{\x\@makeother}%
1999 \edef\x{\endgroup
2000 \def\noexpand\dospecials{\dospecials}%
2001 \expandafter\ifx\cname @sanitize\endcsname\relax\else
2002 \def\noexpand\@sanitize{\@sanitize}%
2003 \fi}%
2004 \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).

```

2005 \def\bbl@active@def#1#2#3#4{%
2006   \@namedef{#3#1}{%
2007     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
2008       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
2009     \else
2010       \bbl@afterfi\csname#2@sh@#1\endcsname
2011     \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.

```

2012 \long\@namedef{#3@arg#1}##1{%
2013   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
2014     \bbl@afterelse\csname#4#1\endcsname##1%
2015   \else
2016     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
2017   \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.

```

2018 \def\initiate@active@char#1{%
2019   \bbl@ifunset{active@char\string#1}%
2020   {\bbl@withactive
2021     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
2022   {}}

```

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).

```

2023 \def\@initiate@active@char#1#2#3{%
2024   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
2025   \ifx#1\@undefined
2026     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
2027   \else
2028     \bbl@csarg\let{oridef@@#2}#1%
2029     \bbl@csarg\edef{oridef@#2}{%
2030       \let\noexpand#1%
2031       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
2032   \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).

```

2033 \ifx#1#3\relax
2034   \expandafter\let\csname normal@char#2\endcsname#3%
2035 \else
2036   \bbl@info{Making #2 an active character}%
2037   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2038   \@namedef{normal@char#2}{%
2039     \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
2040   \else
2041     \@namedef{normal@char#2}{#3}%
2042   \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).

```

2043 \bbl@restoreactive{#2}%
2044 \AtBeginDocument{%
2045   \catcode`#2\active
2046   \if@filesw
2047     \immediate\write\@mainaux{\catcode`\string#2\active}%
2048   \fi}%
2049 \expandafter\bbl@add@special\csname#2\endcsname
2050 \catcode`#2\active
2051 \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}`).

```

2052 \let\bbl@tempa\@firstoftwo
2053 \if\string^#2%
2054   \def\bbl@tempa{\noexpand\textormath}%
2055 \else
2056   \ifx\bbl@mathnormal\@undefined\else
2057     \let\bbl@tempa\bbl@mathnormal
2058   \fi
2059 \fi
2060 \expandafter\edef\csname active@char#2\endcsname{%
2061   \bbl@tempa
2062     {\noexpand\if@safe@actives
2063       \noexpand\expandafter
2064         \expandafter\noexpand\csname normal@char#2\endcsname
2065       \noexpand\else
2066         \noexpand\expandafter
2067         \expandafter\noexpand\csname bbl@doactive#2\endcsname
2068       \noexpand\fi}%
2069   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2070 \bbl@csarg\edef{doactive#2}{%
2071   \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!).

```

2072 \bbl@csarg\edef{active@#2}{%
2073   \noexpand\active@prefix\noexpand#1%
2074   \expandafter\noexpand\csname active@char#2\endcsname}%
2075 \bbl@csarg\edef{normal@#2}{%
2076   \noexpand\active@prefix\noexpand#1%
2077   \expandafter\noexpand\csname normal@char#2\endcsname}%
2078 \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.

```

2079 \bbl@active@def#2\user@group{user@active}{language@active}%

```



```

2080 \bbl@active@def#2\language@group{language@active}{system@active}%
2081 \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.

```

2082 \expandafter\edef\csname\user@group @sh@#2@\endcsname
2083 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2084 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
2085 {\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.

```

2086 \if\string'#2%
2087 \let\prim@s\bbl@prim@s
2088 \let\active@math@prime#1%
2089 \fi
2090 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

2091 <<{*More package options}>> ≡
2092 \DeclareOption{math=active}{}
2093 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2094 <</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.

```

2095 \@ifpackagewith{babel}{KeepShorthandsActive}%
2096 {\let\bbl@restoreactive\@gobble}%
2097 {\def\bbl@restoreactive#1{%
2098   \bbl@exp{%
2099     \\\AfterBabelLanguage\\\CurrentOption
2100     {\catcode`#1=\the\catcode`#1\relax}%
2101     \\\AtEndOfPackage
2102     {\catcode`#1=\the\catcode`#1\relax}}}%
2103 \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.

```

2104 \def\bbl@sh@select#1#2{%
2105   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2106     \bbl@afterelse\bbl@scndcs
2107   \else
2108     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2109   \fi}

```

`\active@prefix` The command \active@prefix which is used in the expansion of active characters has a function similar to \OT1-cmd in that it \protects the active character whenever \protect is *not* \@typeset@protect. The \@gobble is needed to remove a token such as \activechar: (when the

double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

2110 \begingroup
2111 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2112 {\gdef\active@prefix#1{%
2113   \ifx\protect\@typeset@protect
2114   \else
2115     \ifx\protect\@unexpandable@protect
2116       \noexpand#1%
2117     \else
2118       \protect#1%
2119     \fi
2120     \expandafter\@gobble
2121   \fi}}
2122 {\gdef\active@prefix#1{%
2123   \ifincsname
2124     \string#1%
2125     \expandafter\@gobble
2126   \else
2127     \ifx\protect\@typeset@protect
2128     \else
2129       \ifx\protect\@unexpandable@protect
2130         \noexpand#1%
2131       \else
2132         \protect#1%
2133       \fi
2134       \expandafter\expandafter\expandafter\@gobble
2135     \fi
2136   \fi}}
2137 \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⟩`.

```

2138 \newif\if@safe@actives
2139 \@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.

```

2140 \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`.

```

2141 \chardef\bbl@activated\z@
2142 \def\bbl@activate#1{%
2143   \chardef\bbl@activated\@ne
2144   \bbl@withactive{\expandafter\let\expandafter}#1%
2145   \csname bbl@active@\string#1\endcsname}
2146 \def\bbl@deactivate#1{%
2147   \chardef\bbl@activated\tw@
2148   \bbl@withactive{\expandafter\let\expandafter}#1%
2149   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2150 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2151 \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.

```

2152 \def\babel@texpdf#1#2#3#4{%
2153   \ifx\texorpdfstring\@undefined
2154     \textormath{#1}{#3}%
2155   \else
2156     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2157     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2158   \fi}
2159 %
2160 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2161 \def\@decl@short#1#2#3\@nil#4{%
2162   \def\bbl@tempa{#3}%
2163   \ifx\bbl@tempa\@empty
2164     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2165     \bbl@ifunset{#1@sh@\string#2@}{}%
2166     {\def\bbl@tempa{#4}%
2167      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2168      \else
2169        \bbl@info
2170        {Redefining #1 shorthand \string#2\\%
2171         in language \CurrentOption}%
2172      \fi}%
2173     \@namedef{#1@sh@\string#2@}{#4}%
2174   \else
2175     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2176     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2177     {\def\bbl@tempa{#4}%
2178      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2179      \else
2180        \bbl@info
2181        {Redefining #1 shorthand \string#2\string#3\\%
2182         in language \CurrentOption}%
2183      \fi}%
2184     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2185   \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.

```

2186 \def\textormath{%
2187   \ifmmode
2188     \expandafter\@secondoftwo
2189   \else
2190     \expandafter\@firstoftwo
2191   \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’.

```

2192 \def\user@group{user}
2193 \def\language@group{english} % TODO. I don't like defaults
2194 \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.

2195 \def\useshorthands{%
2196   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2197 \def\bbl@usesh@s#1{%
2198   \bbl@usesh@x
2199   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2200   {#1}}
2201 \def\bbl@usesh@x#1#2{%
2202   \bbl@ifshorthand{#2}%
2203   {\def\user@group{user}%
2204     \initiate@active@char{#2}%
2205     #1%
2206     \bbl@activate{#2}}%
2207   {\bbl@error
2208     {I can't declare a shorthand turned off (\string#2)}
2209     {Sorry, but you can't use shorthands which have been\\
2210      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.

2211 \def\user@language@group{user@\language@group}
2212 \def\bbl@set@user@generic#1#2{%
2213   \bbl@ifunset{user@generic@active#1}%
2214   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2215     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2216     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2217       \expandafter\noexpand\csname normal@char#1\endcsname}%
2218     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2219       \expandafter\noexpand\csname user@active#1\endcsname}}%
2220   \@empty}
2221 \newcommand\defineshorthand[3][user]{%
2222   \edef\bbl@tempa{\zap@space#1 \@empty}%
2223   \bbl@for\bbl@tempb\bbl@tempa{%
2224     \if*\expandafter\car\bbl@tempb\@nil
2225       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2226       \@expandtwoargs
2227       \bbl@set@user@generic{\expandafter\string\car#2\@nil}\bbl@tempb
2228     \fi
2229     \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].

2230 \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".

2231 \def\aliasshorthand#1#2{%

```

```

2232 \bbl@ifshorthand{#2}%
2233 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2234 \ifx\document\@notprerr
2235 \notshorthand{#2}%
2236 \else
2237 \initiate@active@char{#2}%
2238 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2239 \csname active@char\string#1\endcsname
2240 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2241 \csname normal@char\string#1\endcsname
2242 \bbl@activate{#2}%
2243 \fi
2244 \fi}%
2245 {\bbl@error
2246 {Cannot declare a shorthand turned off (\string#2)}
2247 {Sorry, but you cannot use shorthands which have been\\%
2248 turned off in the package options}}}

```

\@notshorthand

```

2249 \def\@notshorthand#1{%
2250 \bbl@error{%
2251 The character '\string #1' should be made a shorthand character;\\%
2252 add the command \string\usesshorthands\string{#1\string} to
2253 the preamble.\\%
2254 I will ignore your instruction}%
2255 {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.

```

2256 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2257 \DeclareRobustCommand*\shorthandoff{%
2258 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2259 \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.

```

2260 \def\bbl@switch@sh#1#2{%
2261 \ifx#2\@nnil\else
2262 \bbl@ifunset{\bbl@active@\string#2}%
2263 {\bbl@error
2264 {I can't switch '\string#2' on or off--not a shorthand}%
2265 {This character is not a shorthand. Maybe you made\\%
2266 a typing mistake? I will ignore your instruction.}}%
2267 {\ifcase#1% off, on, off*
2268 \catcode`#212\relax
2269 \or
2270 \catcode`#2\active
2271 \bbl@ifunset{\bbl@shdef@\string#2}%
2272 {}%
2273 {\bbl@withactive{\expandafter\let\expandafter}#2%
2274 \csname bbl@shdef@\string#2\endcsname
2275 \bbl@csarg\let{shdef@\string#2}\relax}%
2276 \ifcase\bbl@activated\or

```

```

2277         \bbl@activate{#2}%
2278     \else
2279         \bbl@deactivate{#2}%
2280     \fi
2281 \or
2282     \bbl@ifunset{bbl@shdef@\string#2}%
2283     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}{#2}%
2284     }%
2285     \csname bbl@oricat@\string#2\endcsname
2286     \csname bbl@oridef@\string#2\endcsname
2287 \fi}%
2288 \bbl@afterfi\bbl@switch@sh#1%
2289 \fi}

```

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

```

2290 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2291 \def\bbl@putsh#1{%
2292     \bbl@ifunset{bbl@active@\string#1}%
2293     {\bbl@putsh@i#1\@empty\@nnil}%
2294     {\csname bbl@active@\string#1\endcsname}}
2295 \def\bbl@putsh@i#1#2\@nnil{%
2296     \csname\language@group @sh@\string#1@%
2297     \ifx\@empty#2\else\string#2\fi\endcsname}
2298 \ifx\bbl@opt@shorthands\@nnil\else
2299     \let\bbl@s@initiate@active@char\initiate@active@char
2300 \def\initiate@active@char#1{%
2301     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2302 \let\bbl@s@switch@sh\bbl@switch@sh
2303 \def\bbl@switch@sh#1#2{%
2304     \ifx#2\@nnil\else
2305         \bbl@afterfi
2306         \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2307     \fi}
2308 \let\bbl@s@activate\bbl@activate
2309 \def\bbl@activate#1{%
2310     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2311 \let\bbl@s@deactivate\bbl@deactivate
2312 \def\bbl@deactivate#1{%
2313     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2314 \fi

```

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

```

2315 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

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

```

2316 \def\bbl@prim@s{%
2317     \prime\futurelet\@let@token\bbl@pr@m@s}
2318 \def\bbl@if@primes#1#2{%
2319     \ifx#1\@let@token
2320         \expandafter\@firstoftwo
2321     \else\ifx#2\@let@token
2322         \bbl@afterelse\expandafter\@firstoftwo
2323     \else
2324         \bbl@afterfi\expandafter\@secondoftwo
2325     \fi\fi}

```

```

2326 \begingroup
2327 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2328 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\'
2329 \lowercase{%
2330 \gdef\bbl@pr@m@s{%
2331 \bbl@if@primes""%
2332 \pr@@@s
2333 {\bbl@if@primes*^\pr@@@t\egroup}}
2334 \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).

```

2335 \initiate@active@char{~}
2336 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2337 \bbl@activate{~}

```

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

```

2338 \expandafter\def\csname OT1dqpos\endcsname{127}
2339 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

2340 \ifx\f@encoding\undefined
2341 \def\f@encoding{OT1}
2342 \fi

```

9.6 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.

```

2343 \bbl@trace{Language attributes}
2344 \newcommand\languageattribute[2]{%
2345 \def\bbl@tempc{#1}%
2346 \bbl@fixname\bbl@tempc
2347 \bbl@iflanguage\bbl@tempc{%
2348 \bbl@vforeach{#2}{%

```

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

```

2349 \ifx\bbl@known@attribs\undefined
2350 \in@false
2351 \else
2352 \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,%
2353 \fi
2354 \ifin@
2355 \bbl@warning{%
2356 You have more than once selected the attribute '##1'\%
2357 for language #1. Reported}%
2358 \else

```

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

```

2359 \bbl@exp{%
2360   \\\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2361 \edef\bbl@tempa{\bbl@tempc-##1}%
2362 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2363 {\curname\bbl@tempc @attr##1\endcsname}%
2364 {\@attrerr{\bbl@tempc}{##1}}%
2365 \fi}}
2366 \@onlypreamble\languageattribute

```

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

```

2367 \newcommand*{\@attrerr}[2]{%
2368   \bbl@error
2369   {The attribute #2 is unknown for language #1.}%
2370   {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}`.

```

2371 \def\bbl@declare@ttribute#1#2#3{%
2372   \bbl@xin@{, #2, }{, \BabelModifiers,}%
2373   \ifin@
2374     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2375   \fi
2376   \bbl@add@list\bbl@attributes{#1-#2}%
2377   \expandafter\def\curname#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.

```

2378 \def\bbl@ifattributeset#1#2#3#4{%
2379   \ifx\bbl@known@attribs\undefined
2380     \in@false
2381   \else
2382     \bbl@xin@{, #1-#2, }{, \bbl@known@attribs,}%
2383   \fi
2384   \ifin@
2385     \bbl@afterelse#3%
2386   \else
2387     \bbl@afterfi#4%
2388   \fi}

```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```

2389 \def\bbl@ifknown@ttrib#1#2{%
2390   \let\bbl@tempa\@secondoftwo
2391   \bbl@loopx\bbl@tempb{#2}%
2392   \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
2393   \ifin@
2394     \let\bbl@tempa\@firstoftwo
2395   \else

```



```

2396 \fi}%
2397 \bbl@tempa}

```

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

```

2398 \def\bbl@clear@ttribs{%
2399 \ifx\bbl@attributes\undefined\else
2400 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2401 \expandafter\bbl@clear@ttrib\bbl@tempa.
2402 }%
2403 \let\bbl@attributes\undefined
2404 \fi}
2405 \def\bbl@clear@ttrib#1-#2.{%
2406 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2407 \AtBeginDocument{\bbl@clear@ttribs}

```

9.7 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`

```

2408 \bbl@trace{Macros for saving definitions}
2409 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2410 \newcount\babel@savecnt
2411 \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.

```

2412 \def\babel@save#1{%
2413 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2414 \toks@\expandafter{\originalTeX\let#1=}%
2415 \bbl@exp{%
2416 \def\\originalTeX{\the\toks@<babel@number\babel@savecnt>\relax}}%
2417 \advance\babel@savecnt@one}
2418 \def\babel@savevariable#1{%
2419 \toks@\expandafter{\originalTeX #1=}%
2420 \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 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.

```

2421 \def\bbl@frenchspacing{%
2422 \ifnum\the\sfcode`\.=\@m

```

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

```

2423 \let\bbl@nonfrenchspacing\relax
2424 \else
2425 \frenchspacing
2426 \let\bbl@nonfrenchspacing\nonfrenchspacing
2427 \fi}
2428 \let\bbl@nonfrenchspacing\nonfrenchspacing
2429 \let\bbl@elt\relax
2430 \edef\bbl@fs@chars{%
2431 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2432 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2433 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
2434 \def\bbl@pre@fs{%
2435 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
2436 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
2437 \def\bbl@post@fs{%
2438 \bbl@save@sfcodes
2439 \edef\bbl@tempa{\bbl@cl{frspc}}%
2440 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
2441 \if u\bbl@tempa % do nothing
2442 \else\if n\bbl@tempa % non french
2443 \def\bbl@elt##1##2##3{%
2444 \ifnum\sfcode`##1=##2\relax
2445 \babel@savevariable{\sfcode`##1}%
2446 \sfcode`##1=##3\relax
2447 \fi}%
2448 \bbl@fs@chars
2449 \else\if y\bbl@tempa % french
2450 \def\bbl@elt##1##2##3{%
2451 \ifnum\sfcode`##1=##3\relax
2452 \babel@savevariable{\sfcode`##1}%
2453 \sfcode`##1=##2\relax
2454 \fi}%
2455 \bbl@fs@chars
2456 \fi\fi\fi}

```

9.8 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.

```

2457 \bbl@trace{Short tags}
2458 \def\babeltags#1{%
2459 \edef\bbl@tempa{\zap@space#1 \@empty}%
2460 \def\bbl@tempb##1=##2\@{#}%
2461 \edef\bbl@tempc{%
2462 \noexpand\newcommand
2463 \expandafter\noexpand\csname ##1\endcsname{%
2464 \noexpand\protect
2465 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2466 \noexpand\newcommand
2467 \expandafter\noexpand\csname text##1\endcsname{%
2468 \noexpand\foreignlanguage{##2}}}
2469 \bbl@tempc}%
2470 \bbl@for\bbl@tempa\bbl@tempa{%
2471 \expandafter\bbl@tempb\bbl@tempa\@{}}

```

9.9 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.

```

2472 \bbl@trace{Hyphens}
2473 \@onlypreamble\babelhyphenation
2474 \AtEndOfPackage{%
2475   \newcommand\babelhyphenation[2][\@empty]{%
2476     \ifx\bbl@hyphenation@relax
2477       \let\bbl@hyphenation@\@empty
2478     \fi
2479     \ifx\bbl@hyphlist\@empty\else
2480       \bbl@warning{%
2481         You must not intermingle \string\selectlanguage\space and\\%
2482         \string\babelhyphenation\space or some exceptions will not\\%
2483         be taken into account. Reported}%
2484       \fi
2485       \ifx\@empty#1%
2486         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2487       \else
2488         \bbl@vforeach{#1}{%
2489           \def\bbl@tempa{##1}%
2490           \bbl@fixname\bbl@tempa
2491           \bbl@iflanguage\bbl@tempa{%
2492             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2493               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2494               {}%
2495               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2496               #2}}}%
2497         \fi}}

```

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

```

2498 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2499 \def\bbl@t@one{T1}
2500 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

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

```

2501 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2502 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2503 \def\bbl@hyphen{%
2504   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2505 \def\bbl@hyphen@i#1#2{%
2506   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2507   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2508   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

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

```

2509 \def\bbl@usehyphen#1{%
2510   \leavevmode
2511   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2512   \nobreak\hskip\z@skip}
2513 \def\bbl@usehyphen#1{%
2514   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2515 \def\bbl@hyphenchar{%
2516   \ifnum\hyphenchar\font=\m@ne
2517     \babe\nullhyphen
2518   \else
2519     \char\hyphenchar\font
2520   \fi}

```

Finally, we define the hyphen “types”. Their names will not change, so you may use them in ldf’s. After a space, the \mbox in \bbl@hy@nobreak is redundant.

```

2521 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2522 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2523 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2524 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2525 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2526 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2527 \def\bbl@hy@repeat{%
2528   \bbl@usehyphen{%
2529     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2530 \def\bbl@hy@repeat{%
2531   \bbl@usehyphen{%
2532     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2533 \def\bbl@hy@empty{\hskip\z@skip}
2534 \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.

```

2535 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

9.10 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.

```

2536 \bbl@trace{Multiencoding strings}
2537 \def\bbl@tglobal#1{\global\let#1#1}
2538 \def\bbl@recatcode#1{% TODO. Used only once?
2539   \@tempcnta="7F
2540   \def\bbl@tempa{%
2541     \ifnum\@tempcnta>"FF\else
2542       \catcode\@tempcnta=#1\relax
2543       \advance\@tempcnta\@ne
2544       \expandafter\bbl@tempa
2545     \fi}%
2546   \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).

```
2547 \@ifpackagewith{babel}{nocase}%
2548   {\let\bbl@patchuclc\relax}%
2549   {\def\bbl@patchuclc{%
2550     \global\let\bbl@patchuclc\relax
2551     \g@addto@macro\@uclclist{\reserved@b\reserved@b\bbl@uclc}}%
2552     \gdef\bbl@uclc##1{%
2553       \let\bbl@encoded\bbl@encoded@uclc
2554       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2555       {##1}%
2556       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2557         \csname\language @bbl@uclc\endcsname}%
2558       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2559     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2560     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2561 \langle\langle *More package options\rangle\rangle \equiv
2562 \DeclareOption{nocase}{}
2563 \rangle\rangle\langle\langle /More package options\rangle\rangle
```

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

```
2564 \langle\langle *More package options\rangle\rangle \equiv
2565 \let\bbl@opt@strings\@nnil % accept strings=value
2566 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2567 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2568 \def\BabelStringsDefault{generic}
2569 \rangle\rangle\langle\langle /More package options\rangle\rangle
```

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.

```
2570 \@onlypreamble\StartBabelCommands
2571 \def\StartBabelCommands{%
2572   \begingroup
2573   \bbl@recatcode{11}%
2574   \langle\langle Macros local to BabelCommands\rangle\rangle
2575   \def\bbl@provstring##1##2{%
2576     \providecommand##1{##2}%
2577     \bbl@tglobal##1}%
2578   \global\let\bbl@scafter\@empty
2579   \let\StartBabelCommands\bbl@startcmds
2580   \ifx\BabelLanguages\relax
2581     \let\BabelLanguages\CurrentOption
2582   \fi
2583   \begingroup
2584   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2585   \StartBabelCommands}
2586 \def\bbl@startcmds{%
2587   \ifx\bbl@screset\@nnil\else
2588     \bbl@usehooks{stopcommands}{}%
2589   \fi
```

```

2590 \endgroup
2591 \begingroup
2592 \@ifstar
2593 {\ifx\bbbl@opt@strings\@nnil
2594   \let\bbbl@opt@strings\BabelStringsDefault
2595   \fi
2596   \bbbl@startcmds@i}%
2597 \bbbl@startcmds@i}
2598 \def\bbbl@startcmds@i#1#2{%
2599 \edef\bbbl@L{\zap@space#1 \@empty}%
2600 \edef\bbbl@G{\zap@space#2 \@empty}%
2601 \bbbl@startcmds@ii}
2602 \let\bbbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

2603 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
2604 \let\SetString@gobbletwo
2605 \let\bbbl@stringdef@gobbletwo
2606 \let\AfterBabelCommands@gobble
2607 \ifx\@empty#1%
2608   \def\bbbl@sc@label{generic}%
2609   \def\bbbl@encstring##1##2{%
2610     \ProvideTextCommandDefault##1{##2}%
2611     \bbbl@toglobal##1%
2612     \expandafter\bbbl@toglobal\csname\string?\string##1\endcsname}%
2613   \let\bbbl@sctest\in@true
2614 \else
2615   \let\bbbl@sc@charset\space % <- zapped below
2616   \let\bbbl@sc@fontenc\space % <- " "
2617   \def\bbbl@tempa##1=##2\@nil{%
2618     \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2619   \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
2620   \def\bbbl@tempa##1 ##2{% space -> comma
2621     ##1%
2622     \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
2623   \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
2624   \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
2625   \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
2626   \def\bbbl@encstring##1##2{%
2627     \bbbl@foreach\bbbl@sc@fontenc{%
2628       \bbbl@ifunset{T@###1}%
2629       {}%
2630       {\ProvideTextCommand##1{####1}{##2}%
2631         \bbbl@toglobal##1%
2632         \expandafter
2633         \bbbl@toglobal\csname###1\string##1\endcsname}}}%
2634   \def\bbbl@sctest{%
2635     \bbbl@xin@{\bbbl@opt@strings,}{\bbbl@sc@label,\bbbl@sc@fontenc,}}%
2636 \fi
2637 \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
2638 \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded

```

```

2639 \let\AfterBabelCommands\bbbl@aftercmds
2640 \let\SetString\bbbl@setstring
2641 \let\bbbl@stringdef\bbbl@encstring
2642 \else % ie, strings=value
2643 \bbbl@scstest
2644 \ifin@
2645 \let\AfterBabelCommands\bbbl@aftercmds
2646 \let\SetString\bbbl@setstring
2647 \let\bbbl@stringdef\bbbl@provstring
2648 \fi\fi\fi
2649 \bbbl@scswitch
2650 \ifx\bbbl@G\@empty
2651 \def\SetString##1##2{%
2652 \bbbl@error{Missing group for string \string##1}%
2653 {You must assign strings to some category, typically\\%
2654 captions or extras, but you set none}}%
2655 \fi
2656 \ifx\@empty#1%
2657 \bbbl@usehooks{defaultcommands}{}%
2658 \else
2659 \@expandtwoargs
2660 \bbbl@usehooks{encodedcommands}{\bbbl@sc@charset}{\bbbl@sc@fontenc}}%
2661 \fi}

```

There are two versions of `\bbbl@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 (`\bbbl@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 `\bbbl@forlang` loops `\bbbl@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 `\bbbl@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).

```

2662 \def\bbbl@forlang#1#2{%
2663 \bbbl@for#1\bbbl@L{%
2664 \bbbl@xin@{, #1,}{, \BabelLanguages,}%
2665 \ifin@#2\relax\fi}}
2666 \def\bbbl@scswitch{%
2667 \bbbl@forlang\bbbl@tempa{%
2668 \ifx\bbbl@G\@empty\else
2669 \ifx\SetString\@gobbletwo\else
2670 \edef\bbbl@GL{\bbbl@G\bbbl@tempa}%
2671 \bbbl@xin@{, \bbbl@GL,}{, \bbbl@screset,}%
2672 \ifin@\else
2673 \global\expandafter\let\csname\bbbl@GL\endcsname\@undefined
2674 \xdef\bbbl@screset{\bbbl@screset, \bbbl@GL}%
2675 \fi
2676 \fi
2677 \fi}}
2678 \AtEndOfPackage{%
2679 \def\bbbl@forlang#1#2{\bbbl@for#1\bbbl@L{\bbbl@ifunset{date#1}{}{#2}}}%
2680 \let\bbbl@scswitch\relax}
2681 \onlypreamble\EndBabelCommands
2682 \def\EndBabelCommands{%
2683 \bbbl@usehooks{stopcommands}{}%
2684 \endgroup
2685 \endgroup
2686 \bbbl@scafter}
2687 \let\bbbl@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.

```

2688 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2689   \bbl@forlang\bbl@tempa{%
2690     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2691     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2692     {\bbl@exp{%
2693       \global\bbbl@add<\bbl@G\bbl@tempa>{\bbbl@scset\#1<\bbl@LC>}}}%
2694     }%
2695   \def\BabelString{#2}%
2696   \bbl@usehooks{stringprocess}{}%
2697   \expandafter\bbl@stringdef
2698   \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`.

```

2699 \ifx\bbl@opt@strings\relax
2700   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2701   \bbl@patchuclc
2702   \let\bbl@encoded\relax
2703   \def\bbl@encoded@uclc#1{%
2704     \@inmathwarn#1%
2705     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2706       \expandafter\ifx\csname ?\string#1\endcsname\relax
2707         \TextSymbolUnavailable#1%
2708       \else
2709         \csname ?\string#1\endcsname
2710       \fi
2711     \else
2712       \csname\cf@encoding\string#1\endcsname
2713     \fi}
2714 \else
2715   \def\bbl@scset#1#2{\def#1{#2}}
2716 \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.

```

2717 << *Macros local to BabelCommands >> ≡
2718 \def\SetStringLoop##1##2{%
2719   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2720   \count@\z@
2721   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2722     \advance\count@\@ne
2723     \toks@\expandafter{\bbl@tempa}%
2724     \bbl@exp{%
2725       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2726       \count@=\the\count@\relax}}}%
2727 <</Macros local to BabelCommands >>

```

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

```

2728 \def\bbl@aftercmds#1{%
2729   \toks@\expandafter{\bbl@scafter#1}%
2730   \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.

```

2731 <<*Macros local to BabelCommands>> ≡
2732 \newcommand\SetCase[3][]{%
2733   \bbl@patchuclc
2734   \bbl@forlang\bbl@tempa{%
2735     \expandafter\bbl@encstring
2736     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2737     \expandafter\bbl@encstring
2738     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2739     \expandafter\bbl@encstring
2740     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2741 <</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.

```

2742 <<*Macros local to BabelCommands>> ≡
2743 \newcommand\SetHyphenMap[1]{%
2744   \bbl@forlang\bbl@tempa{%
2745     \expandafter\bbl@stringdef
2746     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2747 <</Macros local to BabelCommands>>

```

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

```

2748 \newcommand\BabelLower[2]{% one to one.
2749   \ifnum\lccode#1=#2\else
2750     \babel@savevariable{\lccode#1}%
2751     \lccode#1=#2\relax
2752   \fi}
2753 \newcommand\BabelLowerMM[4]{% many-to-many
2754   \@tempcnta=#1\relax
2755   \@tempcntb=#4\relax
2756   \def\bbl@tempa{%
2757     \ifnum\@tempcnta>#2\else
2758       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2759       \advance\@tempcnta#3\relax
2760       \advance\@tempcntb#3\relax
2761       \expandafter\bbl@tempa
2762     \fi}%
2763   \bbl@tempa}
2764 \newcommand\BabelLowerM0[4]{% many-to-one
2765   \@tempcnta=#1\relax
2766   \def\bbl@tempa{%
2767     \ifnum\@tempcnta>#2\else
2768       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2769       \advance\@tempcnta#3
2770       \expandafter\bbl@tempa
2771     \fi}%
2772   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2773 <<*More package options>> ≡
2774 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2775 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2776 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2777 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2778 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}

```

2779 <</More package options>>

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

```
2780 \AtEndOfPackage{%
2781   \ifx\bbbl@opt@hyphenmap\undefined
2782     \bbbl@xin@{,}{\bbbl@language@opts}%
2783     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
2784   \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.

```
2785 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2786   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
2787 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
2788   \bbbl@trim@def\bbbl@tempa{#2}%
2789   \bbbl@xin@{.template}{\bbbl@tempa}%
2790   \ifin@
2791     \bbbl@ini@captions@template{#3}{#1}%
2792   \else
2793     \edef\bbbl@tempd{%
2794       \expandafter\expandafter\expandafter
2795       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2796     \bbbl@xin@
2797       {\expandafter\string\csname #2name\endcsname}%
2798       {\bbbl@tempd}%
2799     \ifin@ % Renew caption
2800       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
2801     \ifin@
2802       \bbbl@exp{%
2803         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2804         {\bbbl@scset\<#2name>\<#1#2name>}%
2805         {}}%
2806       \else % Old way converts to new way
2807         \bbbl@ifunset{#1#2name}%
2808         {\bbbl@exp{%
2809           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2810           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2811           {\def\<#2name>\<#1#2name>}}%
2812           {}}%
2813         {}%
2814       \fi
2815     \else
2816       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2817       \ifin@ % New way
2818         \bbbl@exp{%
2819           \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
2820           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2821           {\bbbl@scset\<#2name>\<#1#2name>}%
2822           {}}%
2823       \else % Old way, but defined in the new way
2824         \bbbl@exp{%
2825           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2826           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2827           {\def\<#2name>\<#1#2name>}}%
2828           {}}%
2829       \fi%
2830     \fi
2831   \@namedef{#1#2name}{#3}%
```

```

2832 \toks@\expandafter{\bbl@captionslist}%
2833 \bbl@exp{\in@{<#2name>}{\the\toks@}}%
2834 \ifin@else
2835 \bbl@exp{\bbl@add\bbl@captionslist{<#2name>}}%
2836 \bbl@tglobal\bbl@captionslist
2837 \fi
2838 \fi}
2839 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

9.11 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.

```

2840 \bbl@trace{Macros related to glyphs}
2841 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2842 \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2843 \setbox\z@\hbox{\lower\dimen\z@ \box\z@}\ht\z@\ht\tw@ \dp\z@\dp\tw@}

```

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

```

2844 \def\save@sfc@q#1{\leavevmode
2845 \begingroup
2846 \edef\SF{\spacefactor\the\spacefactor}#1\SF
2847 \endgroup}

```

9.12 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`.

9.12.1 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.

```

2848 \ProvideTextCommand{\quotedblbase}{OT1}{%
2849 \save@sfc@q{\set@low@box{\textquotedblright\}}%
2850 \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

2851 \ProvideTextCommandDefault{\quotedblbase}{%
2852 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2853 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2854 \save@sfc@q{\set@low@box{\textquoteright\}}%
2855 \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

2856 \ProvideTextCommandDefault{\quotesinglbase}{%
2857 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2858 \ProvideTextCommand{\guillemetleft}{OT1}{%
2859 \ifmmode
2860 \ll
2861 \else

```

```

2862 \save@sf@q{\nobreak
2863 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2864 \fi}
2865 \ProvideTextCommand{\guillemetright}{OT1}{%
2866 \ifmmode
2867 \gg
2868 \else
2869 \save@sf@q{\nobreak
2870 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2871 \fi}
2872 \ProvideTextCommand{\guillemotleft}{OT1}{%
2873 \ifmmode
2874 \ll
2875 \else
2876 \save@sf@q{\nobreak
2877 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2878 \fi}
2879 \ProvideTextCommand{\guillemotright}{OT1}{%
2880 \ifmmode
2881 \gg
2882 \else
2883 \save@sf@q{\nobreak
2884 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2885 \fi}

```

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

```

2886 \ProvideTextCommandDefault{\guillemetleft}{%
2887 \UseTextSymbol{OT1}{\guillemetleft}}
2888 \ProvideTextCommandDefault{\guillemetright}{%
2889 \UseTextSymbol{OT1}{\guillemetright}}
2890 \ProvideTextCommandDefault{\guillemotleft}{%
2891 \UseTextSymbol{OT1}{\guillemotleft}}
2892 \ProvideTextCommandDefault{\guillemotright}{%
2893 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2894 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2895 \ifmmode
2896 <%
2897 \else
2898 \save@sf@q{\nobreak
2899 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2900 \fi}
2901 \ProvideTextCommand{\guilsinglright}{OT1}{%
2902 \ifmmode
2903 >%
2904 \else
2905 \save@sf@q{\nobreak
2906 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2907 \fi}

```

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

```

2908 \ProvideTextCommandDefault{\guilsinglleft}{%
2909 \UseTextSymbol{OT1}{\guilsinglleft}}
2910 \ProvideTextCommandDefault{\guilsinglright}{%
2911 \UseTextSymbol{OT1}{\guilsinglright}}

```

9.12.2 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.

```
2912 \DeclareTextCommand{\ij}{OT1}{%
2913   i\kern-0.02em\bbl@allowhyphens j}
2914 \DeclareTextCommand{\IJ}{OT1}{%
2915   I\kern-0.02em\bbl@allowhyphens J}
2916 \DeclareTextCommand{\ij}{T1}{\char188}
2917 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2918 \ProvideTextCommandDefault{\ij}{%
2919   \UseTextSymbol{OT1}{\ij}}
2920 \ProvideTextCommandDefault{\IJ}{%
2921   \UseTextSymbol{OT1}{\IJ}}
```

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

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

```
2922 \def\crrtic@{\hrule height0.1ex width0.3em}
2923 \def\crttic@{\hrule height0.1ex width0.33em}
2924 \def\ddj@{%
2925   \setbox0\hbox{d}\dimen@=\ht0
2926   \advance\dimen@1ex
2927   \dimen@.45\dimen@
2928   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2929   \advance\dimen@ii.5ex
2930   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2931 \def\DDJ@{%
2932   \setbox0\hbox{D}\dimen@=.55\ht0
2933   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2934   \advance\dimen@ii.15ex % correction for the dash position
2935   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2936   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2937   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2938 %
2939 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2940 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2941 \ProvideTextCommandDefault{\dj}{%
2942   \UseTextSymbol{OT1}{\dj}}
2943 \ProvideTextCommandDefault{\DJ}{%
2944   \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.

```
2945 \DeclareTextCommand{\SS}{OT1}{SS}
2946 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.12.3 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 2947 \ProvideTextCommandDefault{\glq}{%
2948 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2949 \ProvideTextCommand{\grq}{T1}{%
2950 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2951 \ProvideTextCommand{\grq}{TU}{%
2952 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2953 \ProvideTextCommand{\grq}{OT1}{%
2954 \save@sf@q{\kern-.0125em
2955 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2956 \kern.07em\relax}}
2957 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2958 \ProvideTextCommandDefault{\glqq}{%
2959 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2960 \ProvideTextCommand{\grqq}{T1}{%
2961 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2962 \ProvideTextCommand{\grqq}{TU}{%
2963 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2964 \ProvideTextCommand{\grqq}{OT1}{%
2965 \save@sf@q{\kern-.07em
2966 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2967 \kern.07em\relax}}
2968 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2969 \ProvideTextCommandDefault{\flq}{%
2970 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2971 \ProvideTextCommandDefault{\frq}{%
2972 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2973 \ProvideTextCommandDefault{\flqq}{%
2974 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2975 \ProvideTextCommandDefault{\frqq}{%
2976 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

9.12.4 Umlauts and tremas

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

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

```
2977 \def\umlauthigh{%
2978 \def\bbl@umlauta##1{\leavevmode\bgroup%
2979 \expandafter\accent\csname\fontencoding dpos\endcsname
2980 ##1\bbl@allowhyphens\egroup}%
2981 \let\bbl@umlaute\bbl@umlauta}
2982 \def\umlautlow{%
2983 \def\bbl@umlauta{\protect\lower@umlaut}}
```

```

2984 \def\umlaute\lower@umlaute{\protect\lower@umlaute}}
2985 \def\umlaute\lower@umlaute{\protect\lower@umlaute}}
2986 \umlaute\lower@umlaute

```

`\lower@umlaute` The command `\lower@umlaute` 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.

```

2987 \expandafter\ifx\csname U@D\endcsname\relax
2988 \csname newdimen\endcsname\U@D
2989 \fi

```

The following code fools T_EX's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

2990 \def\lower@umlaute#1{%
2991 \leavevmode\bggroup
2992 \U@D 1ex%
2993 {\setbox\z@\hbox{%
2994 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2995 \dimen@ -.45ex\advance\dimen@\ht\z@
2996 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2997 \expandafter\accent\csname\fontencoding dqpos\endcsname
2998 \fontdimen5\font\U@D #1%
2999 \egroup}

```

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

```

3000 \AtBeginDocument{%
3001 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlaut{a}}%
3002 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaut{e}}%
3003 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaut{i}}%
3004 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaut{\i}}%
3005 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlaut{o}}%
3006 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlaut{u}}%
3007 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlaut{A}}%
3008 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaut{E}}%
3009 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaut{I}}%
3010 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlaut{O}}%
3011 \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlaut{U}}%

```

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

```

3012 \ifx\l@english\undefined
3013 \chardef\l@english\z@
3014 \fi
3015 % The following is used to cancel rules in ini files (see Amharic).
3016 \ifx\l@unhyphenated\undefined
3017 \newlanguage\l@unhyphenated
3018 \fi

```

9.13 Layout

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

```
3019 \bbl@trace{Bidi layout}
3020 \providecommand\IfBabelLayout[3]{#3}%
3021 \newcommand\BabelPatchSection[1]{%
3022   \@ifundefined{#1}{}{%
3023     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3024     \@namedef{#1}{%
3025       \@ifstar{\bbl@presec@s{#1}}%
3026       {\@dblarg{\bbl@presec@x{#1}}}}}%
3027 \def\bbl@presec@x#1[#2]#3{%
3028   \bbl@exp{%
3029     \\\select@language@x{\bbl@main@language}%
3030     \\\bbl@cs{sspre@#1}%
3031     \\\bbl@cs{ss@#1}%
3032     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
3033     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3034     \\\select@language@x{\language}}}%
3035 \def\bbl@presec@s#1#2{%
3036   \bbl@exp{%
3037     \\\select@language@x{\bbl@main@language}%
3038     \\\bbl@cs{sspre@#1}%
3039     \\\bbl@cs{ss@#1}*%
3040     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3041     \\\select@language@x{\language}}}%
3042 \IfBabelLayout{sectioning}%
3043   {\BabelPatchSection{part}%
3044     \BabelPatchSection{chapter}%
3045     \BabelPatchSection{section}%
3046     \BabelPatchSection{subsection}%
3047     \BabelPatchSection{subsubsection}%
3048     \BabelPatchSection{paragraph}%
3049     \BabelPatchSection{subparagraph}}%
3050   \def\babel@toc#1{%
3051     \select@language@x{\bbl@main@language}}}%
3052 \IfBabelLayout{captions}%
3053   {\BabelPatchSection{caption}}}
```

9.14 Load engine specific macros

```
3054 \bbl@trace{Input engine specific macros}
3055 \ifcase\bbl@engine
3056   \input txtbabel.def
3057 \or
3058   \input luababel.def
3059 \or
3060   \input xebabel.def
3061 \fi
```

9.15 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.

```
3062 \bbl@trace{Creating languages and reading ini files}
3063 \let\bbl@extend@ini\@gobble
3064 \newcommand\babelprovide[2][]{%
3065   \let\bbl@savelangname\language
```



```

3066 \edef\bbl@savelocaleid{\the\localeid}%
3067 % Set name and locale id
3068 \edef\languagenname{#2}%
3069 \bbl{id@assign
3070 % Initialize keys
3071 \let\bbl@KVP@captions\@nil
3072 \let\bbl@KVP@date\@nil
3073 \let\bbl@KVP@import\@nil
3074 \let\bbl@KVP@main\@nil
3075 \let\bbl@KVP@script\@nil
3076 \let\bbl@KVP@language\@nil
3077 \let\bbl@KVP@hyphenrules\@nil
3078 \let\bbl@KVP@linebreaking\@nil
3079 \let\bbl@KVP@justification\@nil
3080 \let\bbl@KVP@mapfont\@nil
3081 \let\bbl@KVP@maparabic\@nil
3082 \let\bbl@KVP@mapdigits\@nil
3083 \let\bbl@KVP@intraspace\@nil
3084 \let\bbl@KVP@intrapenalty\@nil
3085 \let\bbl@KVP@onchar\@nil
3086 \let\bbl@KVP@transforms\@nil
3087 \global\let\bbl@release@transforms\@empty
3088 \let\bbl@KVP@alph\@nil
3089 \let\bbl@KVP@Alph\@nil
3090 \let\bbl@KVP@labels\@nil
3091 \bbl@csarg\let{KVP@labels*}\@nil
3092 \global\let\bbl@inidata\@empty
3093 \global\let\bbl@extend@ini\@gobble
3094 \gdef\bbl@key@list{;}%
3095 \bbl@forkv{#1}{% TODO - error handling
3096   \in@{/}{##1}%
3097   \ifin@
3098     \global\let\bbl@extend@ini\bbl@extend@ini@aux
3099     \bbl@renewinikey##1\@{##2}%
3100   \else
3101     \bbl@csarg\def{KVP@##1}{##2}%
3102   \fi}%
3103 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
3104 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel#2}\@n\@tw@}%
3105 % == init ==
3106 \ifx\bbl@screset\@undefined
3107   \bbl@ldfinit
3108 \fi
3109 % ==
3110 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3111 \ifcase\bbl@howloaded
3112   \let\bbl@lbkflag\@empty % new
3113 \else
3114   \ifx\bbl@KVP@hyphenrules\@nil\else
3115     \let\bbl@lbkflag\@empty
3116   \fi
3117   \ifx\bbl@KVP@import\@nil\else
3118     \let\bbl@lbkflag\@empty
3119   \fi
3120 \fi
3121 % == import, captions ==
3122 \ifx\bbl@KVP@import\@nil\else
3123   \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
3124   {\ifx\bbl@initoload\relax

```

```

3125         \begingroup
3126         \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3127         \bb1@input@texini{#2}%
3128         \endgroup
3129         \else
3130         \xdef\bb1@KVP@import{\bb1@initoload}%
3131         \fi}%
3132     {}%
3133 \fi
3134 \ifx\bb1@KVP@captions\@nil
3135     \let\bb1@KVP@captions\bb1@KVP@import
3136 \fi
3137 % ==
3138 \ifx\bb1@KVP@transforms\@nil\else
3139     \bb1@replace\bb1@KVP@transforms{ }{,}%
3140 \fi
3141 % == Load ini ==
3142 \ifcase\bb1@howloaded
3143     \bb1@provide@new{#2}%
3144 \else
3145     \bb1@ifblank{#1}%
3146     {}% With \bb1@load@basic below
3147     {\bb1@provide@renew{#2}}%
3148 \fi
3149 % Post tasks
3150 % -----
3151 % == subsequent calls after the first provide for a locale ==
3152 \ifx\bb1@inidata\@empty\else
3153     \bb1@extend@ini{#2}%
3154 \fi
3155 % == ensure captions ==
3156 \ifx\bb1@KVP@captions\@nil\else
3157     \bb1@ifunset{\bb1@extracaps@#2}%
3158     {\bb1@exp{\labelensure[exclude=\\today]{#2}}}%
3159     {\toks@\expandafter\expandafter\expandafter
3160      {\csname \bb1@extracaps@#2\endcsname}%
3161      \bb1@exp{\labelensure[exclude=\\today,include=\the\toks@]}{#2}}%
3162     \bb1@ifunset{\bb1@ensure@\language}%
3163     {\bb1@exp%
3164      \\\DeclareRobustCommand\<\bb1@ensure@\language>[1]{%
3165       \\\foreignlanguage{\language}%
3166       {####1}}}%
3167     {}%
3168     \bb1@exp{%
3169      \\\bb1@tglobal\<\bb1@ensure@\language>%
3170      \\\bb1@tglobal\<\bb1@ensure@\language\space>}%
3171 \fi
3172 % ==
3173 % At this point all parameters are defined if 'import'. Now we
3174 % execute some code depending on them. But what about if nothing was
3175 % imported? We just set the basic parameters, but still loading the
3176 % whole ini file.
3177 \bb1@load@basic{#2}%
3178 % == script, language ==
3179 % Override the values from ini or defines them
3180 \ifx\bb1@KVP@script\@nil\else
3181     \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
3182 \fi
3183 \ifx\bb1@KVP@language\@nil\else

```

```

3184 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
3185 \fi
3186 % == onchar ==
3187 \ifx\bbl@KVP@onchar\@nil\else
3188 \bbl@luahyphenate
3189 \directlua{
3190   if Babel.locale_mapped == nil then
3191     Babel.locale_mapped = true
3192     Babel.linebreaking.add_before(Babel.locale_map)
3193     Babel.loc_to_scr = {}
3194     Babel.chr_to_loc = Babel.chr_to_loc or {}
3195   end}%
3196 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3197 \ifin@
3198 \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
3199 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}}%
3200 \fi
3201 \bbl@exp{\bbl@add\bbl@starthyphens
3202   {\bbl@patterns@lua{\languagename}}}%
3203 % TODO - error/warning if no script
3204 \directlua{
3205   if Babel.script_blocks['\bbl@cl{sbc}'] then
3206     Babel.loc_to_scr[\the\localeid] =
3207       Babel.script_blocks['\bbl@cl{sbc}']
3208     Babel.locale_props[\the\localeid].lc = \the\localeid\space
3209     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
3210   end
3211 }%
3212 \fi
3213 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3214 \ifin@
3215 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
3216 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
3217 \directlua{
3218   if Babel.script_blocks['\bbl@cl{sbc}'] then
3219     Babel.loc_to_scr[\the\localeid] =
3220       Babel.script_blocks['\bbl@cl{sbc}']
3221   end}%
3222 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
3223 \AtBeginDocument{%
3224   \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3225   {\selectfont}}%
3226 \def\bbl@mapselect{%
3227   \let\bbl@mapselect\relax
3228   \edef\bbl@prefontid{\fontid\font}}%
3229 \def\bbl@mapdir##1{%
3230   {\def\languagename{##1}%
3231     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3232     \bbl@switchfont
3233     \directlua{
3234       Babel.locale_props[\the\csname bbl@id@##1\endcsname]
3235         [\bbl@prefontid] = \fontid\font\space}}}%
3236   \fi
3237   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
3238   \fi
3239 % TODO - catch non-valid values
3240 \fi
3241 % == mapfont ==
3242 % For bidi texts, to switch the font based on direction

```

```

3243 \ifx\bb1@KVP@mapfont\@nil\else
3244   \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}}}%
3245   {\bb1@error{Option '\bb1@KVP@mapfont' unknown for\%
3246     mapfont. Use 'direction'.%
3247     {See the manual for details.}}}%
3248   \bb1@ifunset{\bb1@lsys@\language}\bb1@provide@lsys{\language}}}%
3249   \bb1@ifunset{\bb1@wdir@\language}\bb1@provide@dirs{\language}}}%
3250   \ifx\bb1@mapselect\@undefined % TODO. See onchar
3251     \AtBeginDocument{%
3252       \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
3253       {\selectfont}}}%
3254     \def\bb1@mapselect{%
3255       \let\bb1@mapselect\relax
3256       \edef\bb1@prefontid{\fontid\font}}%
3257     \def\bb1@mapdir##1{%
3258       {\def\language{##1}%
3259       \let\bb1@ifrestoring\@firstoftwo % avoid font warning
3260       \bb1@switchfont
3261       \directlua{Babel.fontmap
3262         [\the\csname \bb1@wdir@##1\endcsname]%
3263         [\bb1@prefontid]=\fontid\font}}}%
3264     \fi
3265     \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%
3266   \fi
3267   % == Line breaking: intraspace, intrapenalty ==
3268   % For CJK, East Asian, Southeast Asian, if interspace in ini
3269   \ifx\bb1@KVP@intraspace\@nil\else % We can override the ini or set
3270     \bb1@csarg\edef{intsp@#2}{\bb1@KVP@intraspace}%
3271   \fi
3272   \bb1@provide@intraspace
3273   % == Line breaking: justification ==
3274   \ifx\bb1@KVP@justification\@nil\else
3275     \let\bb1@KVP@linebreaking\bb1@KVP@justification
3276   \fi
3277   \ifx\bb1@KVP@linebreaking\@nil\else
3278     \bb1@xin@{\bb1@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3279     \ifin@
3280       \bb1@csarg\xdef
3281         {\lnbrk@\language}{\expandafter\@car\bb1@KVP@linebreaking\@nil}%
3282     \fi
3283   \fi
3284   \bb1@xin@{/e}{/\bb1@cl{\lnbrk}}%
3285   \ifin@else\bb1@xin@{/k}{/\bb1@cl{\lnbrk}}\fi
3286   \ifin@\bb1@arabicjust\fi
3287   % == Line breaking: hyphenate.other.(locale|script) ==
3288   \ifx\bb1@lbkflag\@empty
3289     \bb1@ifunset{\bb1@hyotl@\language}}}%
3290     {\bb1@csarg\bb1@replace{hyotl@\language}{ }{,}}%
3291     \bb1@startcommands*\language}}}%
3292     \bb1@csarg\bb1@foreach{hyotl@\language}}{%
3293       \ifcase\bb1@engine
3294         \ifnum##1<257
3295           \SetHyphenMap{\BabelLower{##1}{##1}}%
3296         \fi
3297       \else
3298         \SetHyphenMap{\BabelLower{##1}{##1}}%
3299       \fi}%
3300     \bb1@endcommands}%
3301   \bb1@ifunset{\bb1@hyots@\language}}}%

```

```

3302     {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
3303     \bbl@csarg\bbl@foreach{hyots@language}{%
3304         \ifcase\bbl@engine
3305             \ifnum##1<257
3306                 \global\lccode##1=##1\relax
3307             \fi
3308         \else
3309             \global\lccode##1=##1\relax
3310         \fi}}%
3311 \fi
3312 % == Counters: maparabic ==
3313 % Native digits, if provided in ini (TeX level, xe and lua)
3314 \ifcase\bbl@engine\else
3315     \bbl@ifunset{bbl@dgnat@language}{}%
3316     {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
3317         \expandafter\expandafter\expandafter
3318         \bbl@setdigits\csname bbl@dgnat@language\endcsname
3319         \ifx\bbl@KVP@maparabic\@nil\else
3320             \ifx\bbl@latinarabic\@undefined
3321                 \expandafter\let\expandafter\@arabic
3322                 \csname bbl@counter@language\endcsname
3323             \else % ie, if layout=counters, which redefines \@arabic
3324                 \expandafter\let\expandafter\bbl@latinarabic
3325                 \csname bbl@counter@language\endcsname
3326             \fi
3327         \fi
3328     \fi}%
3329 \fi
3330 % == Counters: mapdigits ==
3331 % Native digits (lua level).
3332 \ifodd\bbl@engine
3333     \ifx\bbl@KVP@mapdigits\@nil\else
3334         \bbl@ifunset{bbl@dgnat@language}{}%
3335         {\RequirePackage{luatexbase}%
3336         \bbl@activate@preotf
3337         \directlua{
3338             Babel = Babel or {} %%% -> presets in luababel
3339             Babel.digits_mapped = true
3340             Babel.digits = Babel.digits or {}
3341             Babel.digits[\the\localeid] =
3342                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3343             if not Babel.numbers then
3344                 function Babel.numbers(head)
3345                     local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3346                     local GLYPH = node.id'glyph'
3347                     local inmath = false
3348                     for item in node.traverse(head) do
3349                         if not inmath and item.id == GLYPH then
3350                             local temp = node.get_attribute(item, LOCALE)
3351                             if Babel.digits[temp] then
3352                                 local chr = item.char
3353                                 if chr > 47 and chr < 58 then
3354                                     item.char = Babel.digits[temp][chr-47]
3355                                 end
3356                             end
3357                             elseif item.id == node.id'math' then
3358                                 inmath = (item.subtype == 0)
3359                             end
3360                         end

```

```

3361             return head
3362         end
3363     end
3364 }}%
3365 \fi
3366 \fi
3367 % == Counters: alph, Alph ==
3368 % What if extras<lang> contains a \babel@save\@alph? It won't be
3369 % restored correctly when exiting the language, so we ignore
3370 % this change with the \bbl@alph@saved trick.
3371 \ifx\bbl@KVP@alph\@nil\else
3372     \bbl@extras@wrap{\bbl@alph@saved}%
3373     {\let\bbl@alph@saved\@alph}%
3374     {\let\@alph\bbl@alph@saved
3375     \babel@save\@alph}%
3376     \bbl@exp{%
3377         \bbl@add\<extras\language>{%
3378             \let\@alph\<bbl@cntr\bbl@KVP@alph @\language>}}%
3379 \fi
3380 \ifx\bbl@KVP@Alph\@nil\else
3381     \bbl@extras@wrap{\bbl@Alph@saved}%
3382     {\let\bbl@Alph@saved\@Alph}%
3383     {\let\@Alph\bbl@Alph@saved
3384     \babel@save\@Alph}%
3385     \bbl@exp{%
3386         \bbl@add\<extras\language>{%
3387             \let\@Alph\<bbl@cntr\bbl@KVP@Alph @\language>}}%
3388 \fi
3389 % == require.babel in ini ==
3390 % To load or reload the babel-*.tex, if require.babel in ini
3391 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3392     \bbl@ifunset\bbl@rqtex\language\}%
3393     {\expandafter\ifx\csname bbl@rqtex\language\endcsname\@empty\else
3394         \let\BabelBeforeIni@gobbletwo
3395         \chardef\atcatcode=\catcode\@
3396         \catcode\@=11\relax
3397         \bbl@input@texini{\bbl@cs{rqtex\language}}%
3398         \catcode\@=\atcatcode
3399         \let\atcatcode\relax
3400         \global\bbl@csarg\let{rqtex\language}\relax
3401     \fi}%
3402 \fi
3403 % == frenchspacing ==
3404 \ifcase\bbl@howloaded\in@true\else\in@false\fi
3405 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
3406 \ifin@
3407     \bbl@extras@wrap{\bbl@pre@fs}%
3408     {\bbl@pre@fs}%
3409     {\bbl@post@fs}%
3410 \fi
3411 % == Release saved transforms ==
3412 \bbl@release@transforms\relax % \relax closes the last item.
3413 % == main ==
3414 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3415     \let\language\bbl@savelangname
3416     \chardef\localeid\bbl@savelocaleid\relax
3417 \fi}

```

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

macros. Remember \bbl@startcommands opens a group.

```
3418 \def\bbl@provide@new#1{%
3419   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3420   \@namedef{extras#1}{}%
3421   \@namedef{noextras#1}{}%
3422   \bbl@startcommands*{#1}{captions}%
3423   \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
3424   \def\bbl@tempb##1{%              elt for \bbl@captionslist
3425     \ifx##1\@empty\else
3426       \bbl@exp{%
3427         \\SetString\\##1{%
3428           \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3429       \expandafter\bbl@tempb
3430     \fi}%
3431   \expandafter\bbl@tempb\bbl@captionslist\@empty
3432   \else
3433     \ifx\bbl@initoload\relax
3434       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
3435     \else
3436       \bbl@read@ini{\bbl@initoload}2% % Same
3437     \fi
3438   \fi
3439   \StartBabelCommands*{#1}{date}%
3440   \ifx\bbl@KVP@import\@nil
3441     \bbl@exp{%
3442       \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
3443   \else
3444     \bbl@savetoday
3445     \bbl@savestate
3446   \fi
3447   \bbl@endcommands
3448   \bbl@load@basic{#1}%
3449   % == hyphenmins == (only if new)
3450   \bbl@exp{%
3451     \gdef\<#1hyphenmins>{%
3452       {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3453       {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
3454     % == hyphenrules (also in renew) ==
3455     \bbl@provide@hyphens{#1}%
3456     \ifx\bbl@KVP@main\@nil\else
3457       \expandafter\main@language\expandafter{#1}%
3458     \fi}
3459 %
3460 \def\bbl@provide@renew#1{%
3461   \ifx\bbl@KVP@captions\@nil\else
3462     \StartBabelCommands*{#1}{captions}%
3463     \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
3464     \EndBabelCommands
3465   \fi
3466   \ifx\bbl@KVP@import\@nil\else
3467     \StartBabelCommands*{#1}{date}%
3468     \bbl@savetoday
3469     \bbl@savestate
3470     \EndBabelCommands
3471   \fi
3472   % == hyphenrules (also in new) ==
3473   \ifx\bbl@lbkflag\@empty
3474     \bbl@provide@hyphens{#1}%
```

```
3475 \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.)

```
3476 \def\bbl@load@basic#1{%
3477   \ifcase\bbl@howloaded\or\or
3478     \ifcase\csname bbl@llevel@\language\endcsname
3479       \bbl@csarg\let{lname@\language}\relax
3480     \fi
3481   \fi
3482   \bbl@ifunset{bbl@lname@#1}%
3483   {\def\BabelBeforeIni##1##2{%
3484     \begingroup
3485       \let\bbl@ini@captions@aux\@gobbletwo
3486       \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3487       \bbl@read@ini{##1}1%
3488       \ifx\bbl@initoload\relax\endinput\fi
3489     \endgroup}%
3490   \begingroup      % boxed, to avoid extra spaces:
3491   \ifx\bbl@initoload\relax
3492     \bbl@input@texini{##1}%
3493   \else
3494     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3495   \fi
3496   \endgroup}%
3497   {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
3498 \def\bbl@provide@hyphens#1{%
3499   \let\bbl@tempa\relax
3500   \ifx\bbl@KVP@hyphenrules\@nil\else
3501     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3502     \bbl@foreach\bbl@KVP@hyphenrules{%
3503       \ifx\bbl@tempa\relax      % if not yet found
3504         \bbl@ifsamestring{##1}{+}%
3505         {{\bbl@exp{\addlanguage\<l@##1>}}}%
3506         {}}%
3507         \bbl@ifunset{l@##1}%
3508         {}%
3509         {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
3510       \fi}%
3511   \fi
3512   \ifx\bbl@tempa\relax %      if no opt or no language in opt found
3513     \ifx\bbl@KVP@import\@nil
3514       \ifx\bbl@initoload\relax\else
3515         \bbl@exp{%
3516           \bbl@ifblank{\bbl@cs{hyphr@#1}}%
3517           {}%
3518           {\let\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3519       \fi
3520     \else % if importing
3521       \bbl@exp{%
3522         \bbl@ifblank{\bbl@cs{hyphr@#1}}%
3523         {}%
3524         {\let\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3525     \fi
3526   \fi
3527   \bbl@ifunset{bbl@tempa}%      ie, relax or undefined
```



```

3528 {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
3529 {\bbl@exp{\addialelect\<l@#1>\language}}%
3530 {}}%          so, l@<lang> is ok - nothing to do
3531 {\bbl@exp{\addialelect\<l@#1>\bbl@tempa}}}% found in opt list or ini

```

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

```

3532 \def\bbl@input@texini#1{%
3533 \bbl@bsphack
3534 \bbl@exp{%
3535 \catcode\%%=14 \catcode\==0
3536 \catcode\{=1 \catcode\}=2
3537 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
3538 \catcode\%%=\the\catcode\%\relax
3539 \catcode\==\the\catcode\=\relax
3540 \catcode\{=\the\catcode\{\relax
3541 \catcode\}=\the\catcode\}\relax}%
3542 \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.

```

3543 \def\bbl@inline#1\bbl@inline{%
3544 \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore#1\@@}% ]
3545 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
3546 \def\bbl@iniskip#1\@@{%          if starts with ;
3547 \def\bbl@inistore#1=#2\@@{%      full (default)
3548 \bbl@trim@def\bbl@tempa{#1}%
3549 \bbl@trim\toks@{#2}%
3550 \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
3551 \ifin@else
3552 \bbl@exp{%
3553 \g@addto@macro\bbl@inidata{%
3554 \bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3555 \fi}
3556 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
3557 \bbl@trim@def\bbl@tempa{#1}%
3558 \bbl@trim\toks@{#2}%
3559 \bbl@xin@{.identification.}{.\bbl@section.}%
3560 \ifin@
3561 \bbl@exp{\g@addto@macro\bbl@inidata{%
3562 \bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3563 \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.

```

3564 \ifx\bbl@readstream\undefined
3565 \csname newread\endcsname\bbl@readstream
3566 \fi
3567 \def\bbl@read@ini#1#2{%
3568 \global\let\bbl@extend@ini\gobble
3569 \openin\bbl@readstream=babel-#1.ini
3570 \ifeof\bbl@readstream
3571 \bbl@error
3572 {There is no ini file for the requested language\%
3573 (#1). Perhaps you misspelled it or your installation\%

```

```

3574         is not complete.}%
3575     {Fix the name or reinstall babel.}%
3576 \else
3577     % == Store ini data in \bbl@inidata ==
3578     \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
3579     \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
3580     \bbl@info{Importing
3581         \ifcase#2font and identification \or basic \fi
3582         data for \language\name\%
3583         from babel-#1.ini. Reported}%
3584     \ifnum#2=\z@
3585         \global\let\bbl@inidata\@empty
3586         \let\bbl@inistore\bbl@inistore@min    % Remember it's local
3587     \fi
3588     \def\bbl@section{identification}%
3589     \bbl@exp{\ \bbl@inistore tag.ini=#1\ \ \ \ \}%
3590     \bbl@inistore load.level=#2\ \ \
3591     \loop
3592     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3593         \endlinechar\m@ne
3594         \read\bbl@readstream to \bbl@line
3595         \endlinechar\^^M
3596         \ifx\bbl@line\@empty\else
3597             \expandafter\bbl@iniline\bbl@line\bbl@iniline
3598         \fi
3599     \repeat
3600     % == Process stored data ==
3601     \bbl@csarg\def{lini@\language}{#1}%
3602     \bbl@read@ini@aux
3603     % == 'Export' data ==
3604     \bbl@ini@exports{#2}%
3605     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
3606     \global\let\bbl@inidata\@empty
3607     \bbl@exp{\ \bbl@add@list\ \bbl@ini@loaded{\language}}%
3608     \bbl@tglobal\bbl@ini@loaded
3609     \fi}
3610 \def\bbl@read@ini@aux{%
3611     \let\bbl@savestrings\@empty
3612     \let\bbl@savetoday\@empty
3613     \let\bbl@savestate\@empty
3614     \def\bbl@elt##1##2##3{%
3615         \def\bbl@section{##1}%
3616         \in@{=date.}{=##1}% Find a better place
3617         \ifin@
3618             \bbl@ini@calendar{##1}%
3619         \fi
3620         \bbl@ifunset{bbl@inikv@##1}{}%
3621         {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
3622     \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.

```

3623 \def\bbl@extend@ini@aux#1{%
3624     \bbl@startcommands*{#1}{captions}%
3625     % Activate captions/... and modify exports
3626     \bbl@csarg\def{inikv@captions.licr}##1##2{%
3627         \setlocalecaption{#1}{##1}{##2}}%
3628     \def\bbl@inikv@captions##1##2{%
3629         \bbl@ini@captions@aux{##1}{##2}}%

```

```

3630 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3631 \def\bbl@exportkey##1##2##3{%
3632   \bbl@ifunset{\bbl@kv@##2}{}%
3633   {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
3634     \bbl@exp{\global\let<\bbl@##1@<\language>\<\bbl@kv@##2>}%
3635     \fi}}%
3636 % As with \bbl@read@ini, but with some changes
3637 \bbl@read@ini@aux
3638 \bbl@ini@exports\tw@
3639 % Update inidata@lang by pretending the ini is read.
3640 \def\bbl@elt##1##2##3{%
3641   \def\bbl@section{##1}%
3642   \bbl@iniline##2=##3\bbl@iniline}%
3643   \csname bbl@inidata@#1\endcsname
3644   \global\bbl@csarg\let{inidata@#1}\bbl@inidata
3645   \StartBabelCommands*{#1}{date}% And from the import stuff
3646   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3647   \bbl@savetoday
3648   \bbl@savestate
3649   \bbl@endcommands}

```

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

```

3650 \def\bbl@ini@calendar#1{%
3651   \lowercase{\def\bbl@tempa{=#1=}}%
3652   \bbl@replace\bbl@tempa{=date.gregorian}{}%
3653   \bbl@replace\bbl@tempa{=date.}{}%
3654   \in@{.licr}{#1}%
3655   \ifin@
3656     \ifcase\bbl@engine
3657       \bbl@replace\bbl@tempa{.licr}{}%
3658     \else
3659       \let\bbl@tempa\relax
3660     \fi
3661   \fi
3662   \ifx\bbl@tempa\relax\else
3663     \bbl@replace\bbl@tempa{=}{}%
3664     \bbl@exp{%
3665       \def<\bbl@inikv@#1>####1####2{%
3666         \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
3667   \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).

```

3668 \def\bbl@renewinikey#1/#2\@#3{%
3669   \edef\bbl@tempa{\zap@space #1 \@empty}% section
3670   \edef\bbl@tempb{\zap@space #2 \@empty}% key
3671   \bbl@trim\toks@{#3}% value
3672   \bbl@exp{%
3673     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb}%
3674     \\g@addto@macro\\bbl@inidata{%
3675       \\\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.

```

3676 \def\bbl@exportkey#1#2#3{%
3677   \bbl@ifunset{\bbl@kv@##2}%
3678   {\bbl@csarg\gdef{#1@<\language>}{#3}}%

```

```

3679 {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3680 \bbl@csarg\gdef{#1@\language}\{#3}%
3681 \else
3682 \bbl@exp{\global\let<\bbl@#1@\language>\<\bbl@@kv@#2>}%
3683 \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.

```

3684 \def\bbl@iniwarning#1{%
3685 \bbl@ifunset{\bbl@@kv@identification.warning#1}{}%
3686 {\bbl@warning{%
3687 From babel-\bbl@cs{lini@\language}.ini:\\
3688 \bbl@cs{@kv@identification.warning#1}\\
3689 Reported }}}
3690 %
3691 \let\bbl@release@transforms\@empty
3692 %
3693 \def\bbl@ini@exports#1{%
3694 % Identification always exported
3695 \bbl@iniwarning}%
3696 \ifcase\bbl@engine
3697 \bbl@iniwarning{.pdflatex}%
3698 \or
3699 \bbl@iniwarning{.lualatex}%
3700 \or
3701 \bbl@iniwarning{.xelatex}%
3702 \fi%
3703 \bbl@exportkey{llevel}{identification.load.level}{}%
3704 \bbl@exportkey{elname}{identification.name.english}{}%
3705 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3706 {\csname bbl@elname@\language\endcsname}}%
3707 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
3708 \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
3709 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3710 \bbl@exportkey{esname}{identification.script.name}{}%
3711 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3712 {\csname bbl@esname@\language\endcsname}}%
3713 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
3714 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3715 % Also maps bcp47 -> language
3716 \ifbbl@bcptoname
3717 \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
3718 \fi
3719 % Conditional
3720 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
3721 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3722 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3723 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3724 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3725 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3726 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3727 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3728 \bbl@exportkey{intsp}{typography.intraspaces}{u}%
3729 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3730 \bbl@exportkey{chrng}{characters.ranges}{}%
3731 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3732 \ifnum#1=\tw@ % only (re)new
3733 \bbl@exportkey{rqtex}{identification.require.babel}{}%

```

```

3734 \bbl@tglobal\bbl@savetoday
3735 \bbl@tglobal\bbl@savestate
3736 \bbl@savestrings
3737 \fi
3738 \fi}

```

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

```

3739 \def\bbl@inikv#1#2{%      key=value
3740 \toks@{#2}%              This hides #'s from ini values
3741 \bbl@csarg\edef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3742 \let\bbl@inikv@identification\bbl@inikv
3743 \let\bbl@inikv@typography\bbl@inikv
3744 \let\bbl@inikv@characters\bbl@inikv
3745 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3746 \def\bbl@inikv@counters#1#2{%
3747 \bbl@ifsamestring{#1}{digits}%
3748 {\bbl@error{The counter name 'digits' is reserved for mapping\\
3749 decimal digits}%
3750 {Use another name.}}%
3751 }%
3752 \def\bbl@tempc{#1}%
3753 \bbl@trim@def{\bbl@tempb*}{#2}%
3754 \in@{.1$}{#1$}%
3755 \ifin@
3756 \bbl@replace\bbl@tempc{.1}{}%
3757 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3758 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3759 \fi
3760 \in@{.F.}{#1}%
3761 \ifin\else\in@{.S.}{#1}\fi
3762 \ifin@
3763 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3764 \else
3765 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3766 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3767 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3768 \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.

```

3769 \ifcase\bbl@engine
3770 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3771 \bbl@ini@captions@aux{#1}{#2}}
3772 \else
3773 \def\bbl@inikv@captions#1#2{%
3774 \bbl@ini@captions@aux{#1}{#2}}
3775 \fi

```

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

```

3776 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3777 \bbl@replace\bbl@tempa{.template}{}%
3778 \def\bbl@toreplace{#1}{}%
3779 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%

```

```

3780 \bbl@replace\bbl@toreplace{[]}{\csname}%
3781 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3782 \bbl@replace\bbl@toreplace{[]}{name\endcsname}}}%
3783 \bbl@replace\bbl@toreplace{[]}{\endcsname}}}%
3784 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3785 \ifin@
3786 \nameuse{\bbl@patch\bbl@tempa}%
3787 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3788 \fi
3789 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3790 \ifin@
3791 \toks@\expandafter{\bbl@toreplace}%
3792 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3793 \fi}
3794 \def\bbl@ini@captions@aux#1#2{%
3795 \bbl@trim@def\bbl@tempa{#1}%
3796 \bbl@xin@{.template}{\bbl@tempa}%
3797 \ifin@
3798 \bbl@ini@captions@template{#2}\languagename
3799 \else
3800 \bbl@ifblank{#2}%
3801 {\bbl@exp%
3802 \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3803 {\bbl@trim\toks@{#2}}%
3804 \bbl@exp%
3805 \bbl@add\bbl@savestrings%
3806 \SetString\<\bbl@tempa name>{\the\toks@}}%
3807 \toks@\expandafter{\bbl@captionslist}%
3808 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3809 \ifin@else
3810 \bbl@exp%
3811 \bbl@add\<\bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3812 \bbl@tglobal\<\bbl@extracaps@\languagename>%
3813 \fi
3814 \fi}

```

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

```

3815 \def\bbl@list@the{%
3816 part,chapter,section,subsection,subsubsection,paragraph,%
3817 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3818 table,page,footnote,mpfootnote,mpfn}
3819 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3820 \bbl@ifunset{\bbl@map@#1@\languagename}%
3821 {\nameuse{#1}}%
3822 {\nameuse{\bbl@map@#1@\languagename}}}
3823 \def\bbl@inikv@labels#1#2{%
3824 \in@{.map}{#1}%
3825 \ifin@
3826 \ifx\bbl@KVP@labels\@nil\else
3827 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3828 \ifin@
3829 \def\bbl@tempc{#1}%
3830 \bbl@replace\bbl@tempc{.map}{}%
3831 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3832 \bbl@exp%
3833 \gdef\<\bbl@map@\bbl@tempc @\languagename>%
3834 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3835 \bbl@foreach\bbl@list@the{%
3836 \bbl@ifunset{the##1}{}%

```

```

3837      {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3838      \bbl@exp{%
3839      \\bbl@sreplace\<the##1>%
3840      {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3841      \\bbl@sreplace\<the##1>%
3842      {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3843      \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3844      \toks@\expandafter\expandafter\expandafter{%
3845      \csname the##1\endcsname}%
3846      \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3847      \fi}}%
3848      \fi
3849      \fi
3850      %
3851      \else
3852      %
3853      % The following code is still under study. You can test it and make
3854      % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3855      % language dependent.
3856      \in@{enumerate.}{#1}%
3857      \ifin@
3858      \def\bbl@tempa{#1}%
3859      \bbl@replace\bbl@tempa{enumerate.}{}%
3860      \def\bbl@toreplace{#2}%
3861      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3862      \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3863      \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}}%
3864      \toks@\expandafter{\bbl@toreplace}%
3865      % TODO. Execute only once:
3866      \bbl@exp{%
3867      \\bbl@add\<extras\language>{%
3868      \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3869      \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3870      \\bbl@tglobal\<extras\language>}%
3871      \fi
3872      \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.

```

3873 \def\bbl@chapttype{chapter}
3874 \ifx\@makechapterhead\@undefined
3875   \let\bbl@patchchapter\relax
3876 \else\ifx\thechapter\@undefined
3877   \let\bbl@patchchapter\relax
3878 \else\ifx\ps@headings\@undefined
3879   \let\bbl@patchchapter\relax
3880 \else
3881   \def\bbl@patchchapter{%
3882     \global\let\bbl@patchchapter\relax
3883     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3884     \bbl@tglobal\appendix
3885     \bbl@sreplace\ps@headings
3886     {\@chapapp\ \thechapter}%
3887     {\bbl@chapterformat}%
3888     \bbl@tglobal\ps@headings
3889     \bbl@sreplace\chaptermark
3890     {\@chapapp\ \thechapter}%

```

```

3891     {\bbl@chapterformat}%
3892     \bbl@toglobal\chaptermark
3893     \bbl@sreplace\@makechapterhead
3894     {\@chapapp\space\thechapter}%
3895     {\bbl@chapterformat}%
3896     \bbl@toglobal\@makechapterhead
3897     \gdef\bbl@chapterformat{%
3898         \bbl@ifunset{\bbl@\bbl@chapttype fmt@\languagename}%
3899         {\@chapapp\space\thechapter}
3900         {\@nameuse{\bbl@\bbl@chapttype fmt@\languagename}}}}
3901     \let\bbl@patchappendix\bbl@patchchapter
3902 \fi\fi\fi
3903 \ifx\@part\@undefined
3904     \let\bbl@patchpart\relax
3905 \else
3906     \def\bbl@patchpart{%
3907         \global\let\bbl@patchpart\relax
3908         \bbl@sreplace\@part
3909         {\partname\nobreakspace\thepart}%
3910         {\bbl@partformat}%
3911         \bbl@toglobal\@part
3912         \gdef\bbl@partformat{%
3913             \bbl@ifunset{\bbl@partfmt@\languagename}%
3914             {\partname\nobreakspace\thepart}
3915             {\@nameuse{\bbl@partfmt@\languagename}}}}
3916 \fi

Date. TODO. Document

3917 % Arguments are _not_ protected.
3918 \let\bbl@calendar\@empty
3919 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3920 \def\bbl@localedate#1#2#3#4{%
3921     \begingroup
3922     \ifx\@empty#1\@empty\else
3923         \let\bbl@ld@calendar\@empty
3924         \let\bbl@ld@variant\@empty
3925         \edef\bbl@tempa{\zap@space#1 \@empty}%
3926         \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld##1}{##2}}%
3927         \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3928         \edef\bbl@calendar{%
3929             \bbl@ld@calendar
3930             \ifx\bbl@ld@variant\@empty\else
3931                 .\bbl@ld@variant
3932             \fi}%
3933         \bbl@replace\bbl@calendar{gregorian}{}%
3934     \fi
3935     \bbl@cased
3936     {\@nameuse{\bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3937 \endgroup}
3938 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3939 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3940     \bbl@trim@def\bbl@tempa{#1.#2}%
3941     \bbl@ifsamestring{\bbl@tempa}{months.wide}%           to savedate
3942     {\bbl@trim@def\bbl@tempa{#3}%
3943         \bbl@trim\toks@{#5}%
3944         \@temptokena\expandafter{\bbl@savestate}%
3945         \bbl@exp{% Reverse order - in ini last wins
3946             \def\\bbl@savestate{%
3947                 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%

```



```

3948     \the\@temptokena}}}%
3949     {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3950     {\lowercase{\def\bbl@tempb{#6}}}%
3951     \bbl@trim@def\bbl@toreplace{#5}}}%
3952     \bbl@TG@date
3953     \bbl@ifunset{\bbl@date@\language name @}%
3954     {\bbl@exp{\% TODO. Move to a better place.
3955     \gdef\<\language name date>{\protect\<\language name date >}%
3956     \gdef\<\language name date >####1####2####3{%
3957     \bbl@usedategroupttrue
3958     \<bbl@ensure@\language name>{%
3959     \bbl@localedate{####1}{####2}{####3}}}%
3960     \bbl@add\bbl@savetoday{%
3961     \SetString\today{%
3962     \<\language name date>%
3963     {\the\year}{\the\month}{\the\day}}}%
3964     }%
3965     \global\bbl@csarg\let{date@\language name @}\bbl@toreplace
3966     \ifx\bbl@tempb\empty\else
3967     \global\bbl@csarg\let{date@\language name @}\bbl@tempb\bbl@toreplace
3968     \fi}%
3969     {}%

```

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.

```

3970 \let\bbl@calendar\empty
3971 \newcommand\BabelDateSpace{\nobreakspace}
3972 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3973 \newcommand\BabelDated[1]{\number#1}
3974 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3975 \newcommand\BabelDateM[1]{\number#1}
3976 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3977 \newcommand\BabelDateMMMM[1]{%
3978   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3979 \newcommand\BabelDatey[1]{\number#1}%
3980 \newcommand\BabelDateyy[1]{%
3981   \ifnum#1<10 0\number#1 %
3982   \else\ifnum#1<100 \number#1 %
3983   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3984   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3985   \else
3986     \bbl@error
3987     {Currently two-digit years are restricted to the\
3988     range 0-9999.}%
3989     {There is little you can do. Sorry.}%
3990   \fi\fi\fi\fi}%
3991 \newcommand\BabelDateyyy[1]{\number#1} % FIXME - add leading 0
3992 \def\bbl@replace@finish@iii#1{%
3993   \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
3994 \def\bbl@TG@date{%
3995   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3996   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot}}%
3997   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3998   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3999   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
4000   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
4001   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
4002   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%

```

```

4003 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{####1}}%
4004 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}}%
4005 \bbl@replace\bbl@toreplace{[y]}\bbl@datecctr{####1|}%
4006 \bbl@replace\bbl@toreplace{[m]}\bbl@datecctr{####2|}%
4007 \bbl@replace\bbl@toreplace{[d]}\bbl@datecctr{####3|}%
4008 % Note after \bbl@replace \toks@ contains the resulting string.
4009 % TODO - Using this implicit behavior doesn't seem a good idea.
4010 \bbl@replace@finish@iii\bbl@toreplace}
4011 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
4012 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

4013 \let\bbl@release@transforms\@empty
4014 \@namedef{bbl@inikv@transforms.prehyphenation}{%
4015   \bbl@transforms\babelprehyphenation}
4016 \@namedef{bbl@inikv@transforms.posthyphenation}{%
4017   \bbl@transforms\babelposthyphenation}
4018 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
4019 \begingroup
4020   \catcode`\%=12
4021   \catcode`\&=14
4022   \gdef\bbl@transforms#1#2#3{&%
4023     \ifx\bbl@KVP@transforms\@nil\else
4024       \directlua{
4025         str = [=[#2]=]
4026         str = str:gsub('%.%d+%.%d+$', '')
4027         tex.print([[ \def\string\babeltempa{]] .. str .. [{}]])
4028       }&%
4029       \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
4030       \ifin@
4031         \in@{.0$}{#2$}&%
4032         \ifin@
4033           \g@addto@macro\bbl@release@transforms{&%
4034             \relax\bbl@transforms@aux#1{\languagename}{#3}}&%
4035           \else
4036             \g@addto@macro\bbl@release@transforms{, {#3}}&%
4037             \fi
4038           \fi
4039         \fi}
4040 \endgroup

```

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

```

4041 \def\bbl@provide@lsys#1{%
4042   \bbl@ifunset{bbl@lname@#1}%
4043     {\bbl@load@info{#1}}%
4044   }%
4045   \bbl@csarg\let{lsys@#1}\@empty
4046   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
4047   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
4048   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
4049   \bbl@ifunset{bbl@lname@#1}{%
4050     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
4051   \ifcase\bbl@engine\or\or
4052     \bbl@ifunset{bbl@prehc@#1}{}%
4053     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
4054     }%
4055     {\ifx\bbl@xenoxyph\@undefined
4056       \let\bbl@xenoxyph\bbl@xenoxyph@d

```

```

4057 \ifx\AtBeginDocument\@notprerr
4058 \expandafter\@secondoftwo % to execute right now
4059 \fi
4060 \AtBeginDocument{%
4061 \expandafter\bbbl@add
4062 \csname selectfont \endcsname{\bbbl@xeno-hyph}%
4063 \expandafter\selectlanguage\expandafter{\language\language}%
4064 \expandafter\bbbl@tglobal\csname selectfont \endcsname}%
4065 \fi}}%
4066 \fi
4067 \bbbl@csarg\bbbl@tglobal{\sys#1}}
4068 \def\bbbl@xeno-hyph@d{%
4069 \bbbl@ifset{\bbbl@prehc@\language\language}%
4070 {\ifnum\hyphenchar\font=\default-hyphenchar
4071 \iffontchar\font\bbbl@cl{prehc}\relax
4072 \hyphenchar\font\bbbl@cl{prehc}\relax
4073 \else\iffontchar\font"200B
4074 \hyphenchar\font"200B
4075 \else
4076 \bbbl@warning
4077 {Neither 0 nor ZERO WIDTH SPACE are available\\%
4078 in the current font, and therefore the hyphen\\%
4079 will be printed. Try changing the fontspec's\\%
4080 'HyphenChar' to another value, but be aware\\%
4081 this setting is not safe (see the manual)}%
4082 \hyphenchar\font\default-hyphenchar
4083 \fi\fi
4084 \fi}%
4085 {\hyphenchar\font\default-hyphenchar}}
4086 % \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).

```

4087 \def\bbbl@load@info#1{%
4088 \def\BabelBeforeIni##1##2{%
4089 \begingroup
4090 \bbbl@read@ini{##1}0%
4091 \endinput % babel- .tex may contain only preamble's
4092 \endgroup}% boxed, to avoid extra spaces:
4093 {\bbbl@input@texini{#1}}}

```

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

```

4094 \def\bbbl@setdigits#1#2#3#4#5{%
4095 \bbbl@exp{%
4096 \def\<\language\language digits>###1{% ie, \langdigits
4097 \<\bbbl@digits@\language\language>###1\\\@nil}%
4098 \let\<\bbbl@cntr@digits@\language\language>\<\language\language digits>%
4099 \def\<\language\language counter>###1{% ie, \langcounter
4100 \\\expandafter\<\bbbl@counter@\language\language>%
4101 \\\csname c@###1\endcsname}%
4102 \def\<\bbbl@counter@\language\language>###1{% ie, \bbbl@counter@lang
4103 \\\expandafter\<\bbbl@digits@\language\language>%
4104 \\\number###1\\\@nil}}}%
4105 \def\bbbl@tempa##1##2##3##4##5{%
4106 \bbbl@exp{% Wow, quite a lot of hashes! :-(

```



```

4158 \fi}%
4159 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
4160 \def\bbl@alphnum@invalid#1{%
4161 \bbl@error{Alphabetic numeral too large (#1)}%
4162 {Currently this is the limit.}}

```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

4163 \newcommand\localeinfo[1]{%
4164 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
4165 {\bbl@error{I've found no info for the current locale.\%
4166 The corresponding ini file has not been loaded\%
4167 Perhaps it doesn't exist}%
4168 {See the manual for details.}}%
4169 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}%
4170 \@namedef{bbl@info@name.locale}{lcname}
4171 \@namedef{bbl@info@tag.ini}{lini}
4172 \@namedef{bbl@info@name.english}{elname}
4173 \@namedef{bbl@info@name.opentype}{lname}
4174 \@namedef{bbl@info@tag.bcp47}{tbc}
4175 \@namedef{bbl@info@language.tag.bcp47}{lbc}
4176 \@namedef{bbl@info@tag.opentype}{lotf}
4177 \@namedef{bbl@info@script.name}{esname}
4178 \@namedef{bbl@info@script.name.opentype}{sname}
4179 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4180 \@namedef{bbl@info@script.tag.opentype}{sotf}
4181 \let\bbl@ensureinfo\@gobble
4182 \newcommand\BabelEnsureInfo{%
4183 \ifx\InputIfFileExists\undefined\else
4184 \def\bbl@ensureinfo##1{%
4185 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
4186 \fi
4187 \bbl@foreach\bbl@loaded{%
4188 \def\language{##1}%
4189 \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`.

```

4190 \newcommand\getlocaleproperty{%
4191 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4192 \def\bbl@getproperty@s#1#2#3{%
4193 \let#1\relax
4194 \def\bbl@elt##1##2##3{%
4195 \bbl@ifsamestring{##1/##2}{##3}%
4196 {\providecommand#1{##3}%
4197 \def\bbl@elt####1####2####3{}}}%
4198 {}}%
4199 \bbl@cs{inidata@#2}%
4200 \def\bbl@getproperty@x#1#2#3{%
4201 \bbl@getproperty@s{#1}{#2}{#3}%
4202 \ifx#1\relax
4203 \bbl@error
4204 {Unknown key for locale '#2':\%
4205 #3\%
4206 \string#1 will be set to \relax}%
4207 {Perhaps you misspelled it.}%
4208 \fi}
4209 \let\bbl@ini@loaded\@empty
4210 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel bahavior

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

```
4211 \newcommand\babeladjust[1]{% TODO. Error handling.
4212   \bbl@forkv{#1}{%
4213     \bbl@ifunset{bbl@ADJ@##1@##2}%
4214     {\bbl@cs{ADJ@##1}{##2}}%
4215     {\bbl@cs{ADJ@##1@##2}}}
4216 %
4217 \def\bbl@adjust@lua#1#2{%
4218   \ifvmode
4219     \ifnum\currentgrouplevel=\z@
4220       \directlua{ Babel.#2 }%
4221       \expandafter\expandafter\expandafter\@gobble
4222     \fi
4223   \fi
4224   {\bbl@error   % The error is gobbled if everything went ok.
4225     {Currently, #1 related features can be adjusted only\\%
4226       in the main vertical list.}%
4227     {Maybe things change in the future, but this is what it is.}}}
4228 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4229   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4230 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4231   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4232 \@namedef{bbl@ADJ@bidi.text@on}{%
4233   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4234 \@namedef{bbl@ADJ@bidi.text@off}{%
4235   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4236 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4237   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4238 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4239   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4240 %
4241 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4242   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4243 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4244   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4245 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4246   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4247 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4248   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4249 \@namedef{bbl@ADJ@justify.arabic@on}{%
4250   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4251 \@namedef{bbl@ADJ@justify.arabic@off}{%
4252   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4253 %
4254 \def\bbl@adjust@layout#1{%
4255   \ifvmode
4256     #1%
4257     \expandafter\@gobble
4258   \fi
4259   {\bbl@error   % The error is gobbled if everything went ok.
4260     {Currently, layout related features can be adjusted only\\%
4261       in vertical mode.}%
4262     {Maybe things change in the future, but this is what it is.}}}
4263 \@namedef{bbl@ADJ@layout.tabular@on}{%
4264   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4265 \@namedef{bbl@ADJ@layout.tabular@off}{%
```

```

4266 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4267 \@namedef{bbl@ADJ@layout.lists@on}{%
4268 \bbl@adjust@layout{\let\list\bbl@NL@list}}
4269 \@namedef{bbl@ADJ@layout.lists@off}{%
4270 \bbl@adjust@layout{\let\list\bbl@OL@list}}
4271 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4272 \bbl@activateposthyphen}
4273 %
4274 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4275 \bbl@bcpallowedtrue}
4276 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4277 \bbl@bcpallowedfalse}
4278 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4279 \def\bbl@bcp@prefix{#1}}
4280 \def\bbl@bcp@prefix{bcp47-}
4281 \@namedef{bbl@ADJ@autoload.options}#1{%
4282 \def\bbl@autoload@options{#1}}
4283 \let\bbl@autoload@bcptoptions\empty
4284 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4285 \def\bbl@autoload@bcptoptions{#1}}
4286 \newif\ifbbl@bcptoname
4287 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4288 \bbl@bcptonametrue}
4289 \BabelEnsureInfo}
4290 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4291 \bbl@bcptonamefalse}
4292 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4293 \directlua{ Babel.ignore_pre_char = function(node)
4294     return (node.lang == \the\csname l@nohyphenation\endcsname)
4295     end }}
4296 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4297 \directlua{ Babel.ignore_pre_char = function(node)
4298     return false
4299     end }}
4300 % TODO: use babel name, override
4301 %
4302 % As the final task, load the code for lua.
4303 %
4304 \ifx\directlua\@undefined\else
4305 \ifx\bbl@luapatterns\@undefined
4306 \input luabel.def
4307 \fi
4308 \fi
4309 \</core>

A proxy file for switch.def
4310 \<kernel>
4311 \let\bbl@onlyswitch\@empty
4312 \input babel.def
4313 \let\bbl@onlyswitch\@undefined
4314 \</kernel>
4315 \<patterns>

```

11 Loading hyphenation patterns

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

To make sure that \TeX 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```

4316 <<Make sure ProvidesFile is defined>>
4317 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
4318 \xdef\bb1@format{\jobname}
4319 \def\bb1@version{\<version>}
4320 \def\bb1@date{\<date>}
4321 \ifx\AtBeginDocument\@undefined
4322   \def\@empty{}
4323   \let\orig@dump\dump
4324   \def\dump{%
4325     \ifx\@ztryfc\@undefined
4326     \else
4327       \toks0=\expandafter{\@preamblecmds}%
4328       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4329       \def\@begindocumenthook{}%
4330     \fi
4331     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4332 \fi
4333 <<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.

```

4334 \def\process@line#1#2 #3 #4 {%
4335   \ifx=#1%
4336     \process@synonym{#2}%
4337   \else
4338     \process@language{#1#2}{#3}{#4}%
4339   \fi
4340   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bb1@languages` is also set to empty.

```

4341 \toks@{}
4342 \def\bb1@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.

```

4343 \def\process@synonym#1{%
4344   \ifnum\last@language=\m@ne
4345     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4346   \else
4347     \expandafter\chardef\csname l@#1\endcsname\last@language
4348     \wlog{\string\l@#1=\string\language\the\last@language}%
4349     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4350       \csname\language\hyphenmins\endcsname
4351     \let\bb1@elt\relax
4352     \edef\bb1@languages{\bb1@languages\bb1@elt{#1}{\the\last@language}}}%
4353   \fi}

```


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

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

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

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

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

```

4354 \def\process@language#1#2#3{%
4355   \expandafter\addlanguage\csname l@#1\endcsname
4356   \expandafter\language\csname l@#1\endcsname
4357   \edef\language#1%
4358   \bbl@hook@everylanguage{#1}%
4359   % > luatex
4360   \bbl@get@enc#1::@@@
4361   \begingroup
4362     \lefthyphenmin\m@ne
4363     \bbl@hook@loadpatterns{#2}%
4364     % > luatex
4365     \ifnum\lefthyphenmin=\m@ne
4366     \else
4367       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4368         \the\lefthyphenmin\the\righthyphenmin}%
4369     \fi
4370   \endgroup
4371   \def\bbl@tempa{#3}%
4372   \ifx\bbl@tempa\@empty\else
4373     \bbl@hook@loadexceptions{#3}%
4374     % > luatex
4375   \fi
4376   \let\bbl@elt\relax
4377   \edef\bbl@languages{%
4378     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4379   \ifnum\the\language=\z@
4380     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4381       \set@hyphenmins\tw@\thr@@\relax
4382     \else
4383       \expandafter\expandafter\expandafter\set@hyphenmins
4384       \csname #1hyphenmins\endcsname

```

```

4385 \fi
4386 \the\toks@
4387 \toks@{}%
4388 \fi}

```

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

```

4389 \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.

```

4390 \def\bbl@hook@everylanguage#1{}
4391 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4392 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4393 \def\bbl@hook@loadkernel#1{%
4394 \def\addlanguage{\csname newlanguage\endcsname}%
4395 \def\adddialect##1##2{%
4396 \global\chardef##1##2\relax
4397 \wlog{\string##1 = a dialect from \string\language##2}}%
4398 \def\iflanguage#1{%
4399 \expandafter\ifx\csname l@##1\endcsname\relax
4400 \@nolanerr{##1}%
4401 \else
4402 \ifnum\csname l@##1\endcsname=\language
4403 \expandafter\expandafter\expandafter\@firstoftwo
4404 \else
4405 \expandafter\expandafter\expandafter\@secondoftwo
4406 \fi
4407 \fi}%
4408 \def\providehyphenmins##1##2{%
4409 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4410 \@namedef{##1hyphenmins}{##2}%
4411 \fi}%
4412 \def\set@hyphenmins##1##2{%
4413 \lefthyphenmin##1\relax
4414 \righthyphenmin##2\relax}%
4415 \def\selectlanguage{%
4416 \errhelp{Selecting a language requires a package supporting it}%
4417 \errmessage{Not loaded}}%
4418 \let\foreignlanguage\selectlanguage
4419 \let\otherlanguage\selectlanguage
4420 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4421 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4422 \def\setlocale{%
4423 \errhelp{Find an armchair, sit down and wait}%
4424 \errmessage{Not yet available}}%
4425 \let\uselocale\setlocale
4426 \let\locale\setlocale
4427 \let\selectlocale\setlocale
4428 \let\localename\setlocale
4429 \let\textlocale\setlocale
4430 \let\textlanguage\setlocale
4431 \let\languagetext\setlocale}
4432 \begingroup
4433 \def\AddBabelHook#1#2{%
4434 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4435 \def\next{\toks1}%

```

```

4436 \else
4437 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4438 \fi
4439 \next}
4440 \ifx\directlua\@undefined
4441 \ifx\XeTeXinputencoding\@undefined\else
4442 \input xebabel.def
4443 \fi
4444 \else
4445 \input luababel.def
4446 \fi
4447 \openin1 = babel-\bbl@format.cfg
4448 \ifeof1
4449 \else
4450 \input babel-\bbl@format.cfg\relax
4451 \fi
4452 \closein1
4453 \endgroup
4454 \bbl@hook@loadkernel{switch.def}

```

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

```

4455 \openin1 = language.dat

```

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

```

4456 \def\language{english}%
4457 \ifeof1
4458 \message{I couldn't find the file language.dat,\space
4459 I will try the file hyphen.tex}
4460 \input hyphen.tex\relax
4461 \chardef\l@english\z@
4462 \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 .

```

4463 \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.

```

4464 \loop
4465 \endlinechar\m@ne
4466 \read1 to \bbl@line
4467 \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.

```

4468 \if T\ifeof1F\fi T\relax
4469 \ifx\bbl@line\@empty\else
4470 \edef\bbl@line{\bbl@line\space\space\space}%
4471 \expandafter\process@line\bbl@line\relax
4472 \fi
4473 \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.

```

4474 \begingroup
4475   \def\bbl@elt#1#2#3#4{%
4476     \global\language=#2\relax
4477     \gdef\language{#1}%
4478     \def\bbl@elt##1##2##3##4{}}%
4479   \bbl@languages
4480 \endgroup
4481 \fi
4482 \closein1

```

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

```

4483 \if/\the\toks@/\else
4484   \errhelp{language.dat loads no language, only synonyms}
4485   \errmessage{Orphan language synonym}
4486 \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.

```

4487 \let\bbl@line\@undefined
4488 \let\process@line\@undefined
4489 \let\process@synonym\@undefined
4490 \let\process@language\@undefined
4491 \let\bbl@get@enc\@undefined
4492 \let\bbl@hyph@enc\@undefined
4493 \let\bbl@tempa\@undefined
4494 \let\bbl@hook@loadkernel\@undefined
4495 \let\bbl@hook@everylanguage\@undefined
4496 \let\bbl@hook@loadpatterns\@undefined
4497 \let\bbl@hook@loadexceptions\@undefined
4498 \let\patterns\@undefined

```

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].

```

4499 <<(*More package options)>> ≡
4500 \chardef\bbl@bidimode\z@
4501 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4502 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4503 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4504 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4505 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4506 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4507 <</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.

```

4508 <<(*Font selection)>> ≡
4509 \bbl@trace{Font handling with fontspec}
4510 \ifx\ExplSyntaxOn\@undefined\else
4511   \ExplSyntaxOn
4512   \catcode\ =10

```

```

4513 \def\bbl@loadfontspec{%
4514   \usepackage{fontspec}%
4515   \expandafter
4516   \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4517     Font '\l_fontspec_fontname_tl' is using the\\%
4518     default features for language '##1'.\\%
4519     That's usually fine, because many languages\\%
4520     require no specific features, but if the output is\\%
4521     not as expected, consider selecting another font.}
4522   \expandafter
4523   \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4524     Font '\l_fontspec_fontname_tl' is using the\\%
4525     default features for script '##2'.\\%
4526     That's not always wrong, but if the output is\\%
4527     not as expected, consider selecting another font.}}
4528 \ExplSyntaxOff
4529 \fi
4530 \@onlypreamble\babelfont
4531 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4532   \bbl@foreach{#1}{%
4533     \expandafter\ifx\csname date##1\endcsname\relax
4534       \IfFileExists{babel-##1.tex}%
4535         {\babelprovide{##1}}%
4536       {}%
4537     \fi}%
4538   \edef\bbl@tempa{#1}%
4539   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4540   \ifx\fontspec@undefined
4541     \bbl@loadfontspec
4542   \fi
4543   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4544   \bbl@bblfont}
4545 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4546   \bbl@ifunset{\bbl@tempb family}%
4547     {\bbl@providedefam{\bbl@tempb}}%
4548     {\bbl@exp{%
4549       \\bbl@sreplace\<\bbl@tempb family >%
4550       {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4551   % For the default font, just in case:
4552   \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}}%
4553   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4554     {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4555     \bbl@exp{%
4556       \let\<\bbl@tempb dflt@\language>\<\bbl@tempb dflt@>%
4557       \\bbl@font@set\<\bbl@tempb dflt@\language>%
4558       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4559     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4560       \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:

```

4561 \def\bbl@providedefam#1{%
4562   \bbl@exp{%
4563     \\newcommand\<#1default>{}% Just define it
4564     \\bbl@add@list\\bbl@font@fams{#1}%
4565     \\DeclareRobustCommand\<#1family>{%
4566       \\not@math@alphabet\<#1family>\relax
4567       \\fontfamily\<#1default>\selectfont}%
4568     \\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.

```

4569 \def\bbl@nostdfont#1{%
4570   \bbl@ifunset{bbl@WFF@\f@family}%
4571   {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
4572   \bbl@ifwarn{The current font is not a babel standard family:\%
4573     #1%
4574     \fontname\font\\%
4575     There is nothing intrinsically wrong with this warning, and\\%
4576     you can ignore it altogether if you do not need these\\%
4577     families. But if they are used in the document, you should be\\%
4578     aware 'babel' will no set Script and Language for them, so\\%
4579     you may consider defining a new family with \string\babelfont.\\%
4580     See the manual for further details about \string\babelfont.\\%
4581     Reported}}
4582   {}}}%
4583 \gdef\bbl@switchfont{%
4584   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
4585   \bbl@exp{% eg Arabic -> arabic
4586   \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4587   \bbl@foreach\bbl@font@fams{%
4588     \bbl@ifunset{bbl@##1dflt@\language}% (1) language?
4589     {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4590       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4591         {}% 123=F - nothing!
4592         {\bbl@exp{% 3=T - from generic
4593           \global\let<bbl@##1dflt@\language>%
4594           \<bbl@##1dflt@>}}}%
4595         {\bbl@exp{% 2=T - from script
4596           \global\let<bbl@##1dflt@\language>%
4597           \<bbl@##1dflt@*\bbl@tempa>}}}%
4598         {}}}% 1=T - language, already defined
4599   \def\bbl@tempa{\bbl@nostdfont}}}%
4600   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4601     \bbl@ifunset{bbl@##1dflt@\language}%
4602     {\bbl@cs{famrst@##1}%
4603     \global\bbl@csarg\let{famrst@##1}\relax}%
4604     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4605       \\bbl@add\\originalTeX{%
4606       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4607       \<##1default>\<##1family>{##1}}}%
4608       \\bbl@font@set<bbl@##1dflt@\language>% the main part!
4609       \<##1default>\<##1family>}}}%
4610   \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.

```

4611 \ifx\f@family\undefined\else % if latex
4612 \ifcase\bbl@engine % if pdftex
4613   \let\bbl@cckckstdfonts\relax
4614 \else
4615   \def\bbl@cckckstdfonts{%
4616     \begingroup
4617     \global\let\bbl@cckckstdfonts\relax
4618     \let\bbl@tempa\@empty
4619     \bbl@foreach\bbl@font@fams{%
4620       \bbl@ifunset{bbl@##1dflt@}%
4621       {\nameuse{##1family}%
4622       \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4623       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%

```

```

4624         \space\space\fontname\font\\\}%
4625         \bbl@csarg\xdef{##1dflt@}{\f@family}%
4626         \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4627     {}}%
4628     \ifx\bbl@tempa\@empty\else
4629         \bbl@infowarn{The following font families will use the default\\%
4630             settings for all or some languages:\\%
4631             \bbl@tempa
4632             There is nothing intrinsically wrong with it, but\\%
4633             'babel' will no set Script and Language, which could\\%
4634             be relevant in some languages. If your document uses\\%
4635             these families, consider redefining them with \string\babelfont.\\%
4636             Reported}%
4637     \fi
4638 \endgroup}
4639 \fi
4640 \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.

```

4641 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4642     \bbl@xin@{<>}{#1}%
4643     \ifin@
4644         \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4645     \fi
4646     \bbl@exp{%
4647         \def\\#2#1%          eg, \rmdefault{\bbl@rmdflt@lang}
4648         \\bbl@ifsamestring{#2}{\f@family}%
4649         {\\#3%
4650             \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4651         \let\\bbl@tempa\relax}%
4652     {}}}
4653 %      TODO - next should be global?, but even local does its job. I'm
4654 %      still not sure -- must investigate:
4655 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4656     \let\bbl@tempa\bbl@mapselect
4657     \let\bbl@mapselect\relax
4658     \let\bbl@temp@fam#4%          eg, '\rmfamily', to be restored below
4659     \let#4\@empty %          Make sure \renewfontfamily is valid
4660     \bbl@exp{%
4661         \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4662         \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4663         {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4664         \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4665         {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4666         \\renewfontfamily\\#4%
4667         [\bbl@cs{lsys@\language\name},#2]{#3}% ie \bbl@exp{.}{#3}
4668     \begingroup
4669         #4%
4670         \xdef#1{\f@family}%          eg, \bbl@rmdflt@lang{FreeSerif(0)}
4671     \endgroup
4672     \let#4\bbl@temp@fam
4673     \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4674     \let\bbl@mapselect\bbl@tempa}%

```

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

```

4675 \def\bbl@font@rst#1#2#3#4{%
4676   \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`.

```

4677 \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 is explained in the user guide, but essentially – that was not the way to go :-).

```

4678 \newcommand\babelFSstore[2][{%
4679   \bbl@ifblank{#1}%
4680   {\bbl@csarg\def{sname@#2}{Latin}}%
4681   {\bbl@csarg\def{sname@#2}{#1}}%
4682   \bbl@provide@dirs{#2}%
4683   \bbl@csarg\ifnum{wdir@#2}>\z@
4684     \let\bbl@beforeforeign\leavevmode
4685     \EnableBabelHook{babel-bidi}%
4686   \fi
4687   \bbl@foreach{#2}{%
4688     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4689     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4690     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4691 \def\bbl@FSstore#1#2#3#4{%
4692   \bbl@csarg\edef{#2default#1}{#3}%
4693   \expandafter\addto\csname extras#1\endcsname{%
4694     \let#4#3%
4695     \ifx#3\f@family
4696       \edef#3{\csname bbl@#2default#1\endcsname}%
4697       \fontfamily{#3}\selectfont
4698     \else
4699       \edef#3{\csname bbl@#2default#1\endcsname}%
4700       \fi}%
4701   \expandafter\addto\csname noextras#1\endcsname{%
4702     \ifx#3\f@family
4703       \fontfamily{#4}\selectfont
4704       \fi
4705     \let#3#4}}
4706 \let\bbl@langfeatures\@empty
4707 \def\babelFSfeatures{% make sure \fontspec is redefined once
4708   \let\bbl@ori@fontspec\fontspec
4709   \renewcommand\fontspec[1][{%
4710     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4711   \let\babelFSfeatures\bbl@FSfeatures
4712   \babelFSfeatures}
4713 \def\bbl@FSfeatures#1#2{%
4714   \expandafter\addto\csname extras#1\endcsname{%
4715     \babel@save\bbl@langfeatures
4716     \edef\bbl@langfeatures{#2,}}
4717 <</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.

```

4718 <<(*Footnote changes)>> ≡
4719 \bbl@trace{Bidi footnotes}

```



```

4720 \ifnum\bb1@bidimode>\z@
4721 \def\bb1@footnote#1#2#3{%
4722   \@ifnextchar[%
4723     {\bb1@footnote@o{#1}{#2}{#3}}%
4724     {\bb1@footnote@x{#1}{#2}{#3}}}
4725 \long\def\bb1@footnote@x#1#2#3#4{%
4726   \bgroup
4727     \select@language@x{\bb1@main@language}%
4728     \bb1@fn@footnote{#2#1{\ignorespaces#4}#3}%
4729   \egroup}
4730 \long\def\bb1@footnote@o#1#2#3[#4]#5{%
4731   \bgroup
4732     \select@language@x{\bb1@main@language}%
4733     \bb1@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4734   \egroup}
4735 \def\bb1@footnotetext#1#2#3{%
4736   \@ifnextchar[%
4737     {\bb1@footnotetext@o{#1}{#2}{#3}}%
4738     {\bb1@footnotetext@x{#1}{#2}{#3}}}
4739 \long\def\bb1@footnotetext@x#1#2#3#4{%
4740   \bgroup
4741     \select@language@x{\bb1@main@language}%
4742     \bb1@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4743   \egroup}
4744 \long\def\bb1@footnotetext@o#1#2#3[#4]#5{%
4745   \bgroup
4746     \select@language@x{\bb1@main@language}%
4747     \bb1@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4748   \egroup}
4749 \def\BabelFootnote#1#2#3#4{%
4750   \ifx\bb1@fn@footnote\undefined
4751     \let\bb1@fn@footnote\footnote
4752   \fi
4753   \ifx\bb1@fn@footnotetext\undefined
4754     \let\bb1@fn@footnotetext\footnotetext
4755   \fi
4756   \bb1@ifblank{#2}%
4757     {\def#1{\bb1@footnote{\@firstofone}{#3}{#4}}
4758     \@namedef{\bb1@stripslash#1text}%
4759     {\bb1@footnotetext{\@firstofone}{#3}{#4}}}%
4760     {\def#1{\bb1@exp{\bb1@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4761     \@namedef{\bb1@stripslash#1text}%
4762     {\bb1@exp{\bb1@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4763 \fi
4764 <</Footnote changes>>

```

Now, the code.

```

4765 (*xetex)
4766 \def\BabelStringsDefault{unicode}
4767 \let\xebbl@stop\relax
4768 \AddBabelHook{xetex}{encodedcommands}{%
4769   \def\bb1@tempa{#1}%
4770   \ifx\bb1@tempa\empty
4771     \XeTeXinputencoding"bytes"%
4772   \else
4773     \XeTeXinputencoding"#1"%
4774   \fi
4775   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4776 \AddBabelHook{xetex}{stopcommands}{%

```

```

4777 \xebbl@stop
4778 \let\xebbl@stop\relax}
4779 \def\bbl@intraspace#1 #2 #3\@@{%
4780 \bbl@csarg\gdef{\xeisp@{\language\language}}%
4781 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4782 \def\bbl@intrapenalty#1\@@{%
4783 \bbl@csarg\gdef{\xeipn@{\language\language}}%
4784 {\XeTeXlinebreakpenalty #1\relax}}
4785 \def\bbl@provide@intraspace{%
4786 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4787 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4788 \ifin@
4789 \bbl@ifunset{\bbl@intsp@{\language\language}}{%
4790 {\expandafter\ifx\csname\bbl@intsp@{\language\language}\endcsname\@empty\else
4791 \ifx\bbl@KVP@intraspace\@nil
4792 \bbl@exp{%
4793 \\\bbl@intraspace\bbl@cl{intsp}\\\@}%
4794 \fi
4795 \ifx\bbl@KVP@intrapenalty\@nil
4796 \bbl@intrapenalty0\@@
4797 \fi
4798 \fi
4799 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4800 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4801 \fi
4802 \ifx\bbl@KVP@intrapenalty\@nil\else
4803 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4804 \fi
4805 \bbl@exp{%
4806 % TODO. Execute only once (but redundant):
4807 \\\bbl@add\<extras\language>{%
4808 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4809 \<bbl@xeisp@{\language\language}>%
4810 \<bbl@xeipn@{\language\language}>%
4811 \\\bbl@toglobal\<extras\language>%
4812 \\\bbl@add\<noextras\language>{%
4813 \XeTeXlinebreaklocale "en"%
4814 \\\bbl@toglobal\<noextras\language>}}%
4815 \ifx\bbl@ispacesize\@undefined
4816 \gdef\bbl@ispacesize{\bbl@cl{\xeisp}}%
4817 \ifx\AtBeginDocument\@notprerr
4818 \expandafter\@secondoftwo % to execute right now
4819 \fi
4820 \AtBeginDocument{%
4821 \expandafter\bbl@add
4822 \csname selectfont \endcsname{\bbl@ispacesize}%
4823 \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4824 \fi}%
4825 \fi}
4826 \ifx\DisableBabelHook\@undefined\endinput\fi
4827 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4828 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4829 \DisableBabelHook{babel-fontspec}
4830 <<Font selection>>
4831 \input txtbabel.def
4832 </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 pdf_{te}x and xet_{ex}.

```
4833 (*texxet)
4834 \providecommand\bbl@provide@intraspace{}
4835 \bbl@trace{Redefinitions for bidi layout}
4836 \def\bbl@sspre@caption{%
4837   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4838 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4839 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4840 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4841 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4842   \def\@hangfrom#1{%
4843     \setbox\@tempboxa\hbox{#1}%
4844     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4845     \noindent\box\@tempboxa}
4846 \def\raggedright{%
4847   \let\@centercr
4848   \bbl@startskip\z@skip
4849   \@rightskip\@flushglue
4850   \bbl@endskip\@rightskip
4851   \parindent\z@
4852   \parfillskip\bbl@startskip}
4853 \def\raggedleft{%
4854   \let\@centercr
4855   \bbl@startskip\@flushglue
4856   \bbl@endskip\z@skip
4857   \parindent\z@
4858   \parfillskip\bbl@endskip}
4859 \fi
4860 \IfBabelLayout{lists}
4861   {\bbl@sreplace\list
4862     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4863     \def\bbl@listleftmargin{%
4864       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4865     \ifcase\bbl@engine
4866       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4867       \def\p@enumiii{\p@enumii}\theenumii{}
4868     \fi
4869     \bbl@sreplace\@verbatim
4870       {\leftskip\@totalleftmargin}%
4871       {\bbl@startskip\textwidth
4872         \advance\bbl@startskip-\linewidth}%
4873     \bbl@sreplace\@verbatim
4874       {\rightskip\z@skip}%
4875       {\bbl@endskip\z@skip}}%
4876   {}
4877 \IfBabelLayout{contents}
4878   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4879     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4880   {}
4881 \IfBabelLayout{columns}
4882   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%

```

```

4883 \def\bbl@outputbox#1{%
4884 \hb@xt@\textwidth{%
4885 \hskip\columnwidth
4886 \hfil
4887 {\normalcolor\vrule \@width\columnseprule}%
4888 \hfil
4889 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4890 \hskip-\textwidth
4891 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4892 \hskip\columnsep
4893 \hskip\columnwidth}}}%
4894 {}
4895 <<Footnote changes>>
4896 \IfBabelLayout{footnotes}%
4897 {\BabelFootnote\footnote\language\{}}%
4898 \BabelFootnote\localfootnote\language\{}}%
4899 \BabelFootnote\mainfootnote\{}}{}%
4900 {}

```

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.

```

4901 \IfBabelLayout{counters}%
4902 {\let\bbl@latinarabic=\@arabic
4903 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4904 \let\bbl@asciroman=\@roman
4905 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4906 \let\bbl@asciiRoman=\@Roman
4907 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4908 </texxet>

```

13.3 LuaTeX

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

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

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

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

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

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

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a

dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated. This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg. \babelpatterns).

```

4909 <(*luatex)
4910 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4911 \bbl@trace{Read language.dat}
4912 \ifx\bbl@readstream\undefined
4913 \csname newread\endcsname\bbl@readstream
4914 \fi
4915 \begingroup
4916 \toks@{}
4917 \count@ \z@ % 0=start, 1=0th, 2=normal
4918 \def\bbl@process@line#1#2 #3 #4 {%
4919 \ifx=#1%
4920 \bbl@process@synonym{#2}%
4921 \else
4922 \bbl@process@language{#1#2}{#3}{#4}%
4923 \fi
4924 \ignorespaces}
4925 \def\bbl@manylang{%
4926 \ifnum\bbl@last>\@ne
4927 \bbl@info{Non-standard hyphenation setup}%
4928 \fi
4929 \let\bbl@manylang\relax}
4930 \def\bbl@process@language#1#2#3{%
4931 \ifcase\count@
4932 \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4933 \or
4934 \count@\tw@
4935 \fi
4936 \ifnum\count@=\tw@
4937 \expandafter\addlanguage\csname l@#1\endcsname
4938 \language\allocationnumber
4939 \chardef\bbl@last\allocationnumber
4940 \bbl@manylang
4941 \let\bbl@elt\relax
4942 \xdef\bbl@languages{%
4943 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4944 \fi
4945 \the\toks@
4946 \toks@{}}
4947 \def\bbl@process@synonym@aux#1#2{%
4948 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4949 \let\bbl@elt\relax
4950 \xdef\bbl@languages{%
4951 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4952 \def\bbl@process@synonym#1{%
4953 \ifcase\count@
4954 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4955 \or
4956 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4957 \else
4958 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4959 \fi}
4960 \ifx\bbl@languages\undefined % Just a (sensible?) guess

```

```

4961 \chardef\l@english\z@
4962 \chardef\l@USenglish\z@
4963 \chardef\bbl@last\z@
4964 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
4965 \gdef\bbl@languages{%
4966   \bbl@elt{english}{0}{\hyphen.tex}{}%
4967   \bbl@elt{USenglish}{0}{}{}}
4968 \else
4969   \global\let\bbl@languages@format\bbl@languages
4970   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4971     \ifnum#2>\z@\else
4972       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4973     \fi}%
4974   \xdef\bbl@languages{\bbl@languages}%
4975 \fi
4976 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{} } % Define flags
4977 \bbl@languages
4978 \openin\bbl@readstream=language.dat
4979 \ifeof\bbl@readstream
4980   \bbl@warning{I couldn't find language.dat. No additional\%
4981     patterns loaded. Reported}%
4982 \else
4983   \loop
4984     \endlinechar\m@ne
4985     \read\bbl@readstream to \bbl@line
4986     \endlinechar\^^M
4987     \if T\ifeof\bbl@readstream F\fi T\relax
4988     \ifx\bbl@line\@empty\else
4989       \edef\bbl@line{\bbl@line\space\space\space}%
4990       \expandafter\bbl@process@line\bbl@line\relax
4991     \fi
4992   \repeat
4993 \fi
4994 \endgroup
4995 \bbl@trace{Macros for reading patterns files}
4996 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4997 \ifx\babelcatcodetablenum\undefined
4998   \ifx\newcatcodetable\undefined
4999     \def\babelcatcodetablenum{5211}
5000   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5001 \else
5002   \newcatcodetable\babelcatcodetablenum
5003   \newcatcodetable\bbl@pattcodes
5004 \fi
5005 \else
5006   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5007 \fi
5008 \def\bbl@luapatterns#1#2{%
5009   \bbl@get@enc#1::\@@
5010   \setbox\z@\hbox\bgroup
5011     \begingroup
5012       \savecatcodetable\babelcatcodetablenum\relax
5013       \initcatcodetable\bbl@pattcodes\relax
5014       \catcodetable\bbl@pattcodes\relax
5015       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5016       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\-=13
5017       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5018       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5019       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12

```

```

5020 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5021 \input #1\relax
5022 \catcodetable\babelcatcodetablenum\relax
5023 \endgroup
5024 \def\bbl@tempa{#2}%
5025 \ifx\bbl@tempa\@empty\else
5026 \input #2\relax
5027 \fi
5028 \egroup}%
5029 \def\bbl@patterns@lua#1{%
5030 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5031 \csname l@#1\endcsname
5032 \edef\bbl@tempa{#1}%
5033 \else
5034 \csname l@#1:\f@encoding\endcsname
5035 \edef\bbl@tempa{#1:\f@encoding}%
5036 \fi\relax
5037 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5038 \@ifundefined{bbl@hyphendata@the\language}%
5039 {\def\bbl@elt##1##2##3##4{%
5040 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5041 \def\bbl@tempb{##3}%
5042 \ifx\bbl@tempb\@empty\else % if not a synonymous
5043 \def\bbl@tempc{##3}{##4}%
5044 \fi
5045 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5046 \fi}%
5047 \bbl@languages
5048 \@ifundefined{bbl@hyphendata@the\language}%
5049 {\bbl@info{No hyphenation patterns were set for\%
5050 language '\bbl@tempa'. Reported}}%
5051 {\expandafter\expandafter\expandafter\bbl@luapatterns
5052 \csname bbl@hyphendata@the\language\endcsname}}}%
5053 \endinput\fi
5054 % Here ends \ifx\AddBabelHook\@undefined
5055 % A few lines are only read by hyphen.cfg
5056 \ifx\DisableBabelHook\@undefined
5057 \AddBabelHook{luatex}{everylanguage}{%
5058 \def\process@language##1##2##3{%
5059 \def\process@line####1####2 ####3 ####4 {}}}
5060 \AddBabelHook{luatex}{loadpatterns}{%
5061 \input #1\relax
5062 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5063 {{#1}}}%
5064 \AddBabelHook{luatex}{loadexceptions}{%
5065 \input #1\relax
5066 \def\bbl@tempb##1##2{{##1}{##1}}%
5067 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5068 {\expandafter\expandafter\expandafter\bbl@tempb
5069 \csname bbl@hyphendata@the\language\endcsname}}
5070 \endinput\fi
5071 % Here stops reading code for hyphen.cfg
5072 % The following is read the 2nd time it's loaded
5073 \begingroup % TODO - to a lua file
5074 \catcode`\%=12
5075 \catcode`\'=12
5076 \catcode`\`=12
5077 \catcode`\:=12
5078 \directlua{

```

```

5079 Babel = Babel or {}
5080 function Babel.bytes(line)
5081     return line:gsub("(.)",
5082         function (chr) return unicode.utf8.char(string.byte(chr)) end)
5083 end
5084 function Babel.begin_process_input()
5085     if luatexbase and luatexbase.add_to_callback then
5086         luatexbase.add_to_callback('process_input_buffer',
5087             Babel.bytes, 'Babel.bytes')
5088     else
5089         Babel.callback = callback.find('process_input_buffer')
5090         callback.register('process_input_buffer', Babel.bytes)
5091     end
5092 end
5093 function Babel.end_process_input ()
5094     if luatexbase and luatexbase.remove_from_callback then
5095         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5096     else
5097         callback.register('process_input_buffer', Babel.callback)
5098     end
5099 end
5100 function Babel.addpatterns(pp, lg)
5101     local lg = lang.new(lg)
5102     local pats = lang.patterns(lg) or ''
5103     lang.clear_patterns(lg)
5104     for p in pp:gmatch('[^%s]+') do
5105         ss = ''
5106         for i in string.utfcharacters(p:gsub('%d', '')) do
5107             ss = ss .. '%d?' .. i
5108         end
5109         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5110         ss = ss:gsub('%.%%d%?$', '%%.')
5111         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5112         if n == 0 then
5113             tex.sprint(
5114                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5115                 .. p .. [[{ }]])
5116             pats = pats .. ' ' .. p
5117         else
5118             tex.sprint(
5119                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5120                 .. p .. [[{ }]])
5121         end
5122     end
5123     lang.patterns(lg, pats)
5124 end
5125 }
5126 \endgroup
5127 \ifx\newattribute\undefined\else
5128     \newattribute\bbl@attr@locale
5129     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5130     \AddBabelHook{luatex}{beforeextras}{%
5131         \setattribute\bbl@attr@locale\localeid}
5132 \fi
5133 \def\BabelStringsDefault{unicode}
5134 \let\luabbl@stop\relax
5135 \AddBabelHook{luatex}{encodedcommands}{%
5136     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5137     \ifx\bbl@tempa\bbl@tempb\else

```



```

5138 \directlua{Babel.begin_process_input()}%
5139 \def\luabbl@stop{%
5140 \directlua{Babel.end_process_input()}}%
5141 \fi}%
5142 \AddBabelHook{luatex}{stopcommands}{%
5143 \luabbl@stop
5144 \let\luabbl@stop\relax}
5145 \AddBabelHook{luatex}{patterns}{%
5146 \ifundefined{bbl@hyphendata@the\language}%
5147 {\def\bbl@elt##1##2##3##4{%
5148 \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5149 \def\bbl@tempb{##3}%
5150 \ifx\bbl@tempb\empty\else % if not a synonymous
5151 \def\bbl@tempc{##3}{##4}}%
5152 \fi
5153 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5154 \fi}%
5155 \bbl@languages
5156 \ifundefined{bbl@hyphendata@the\language}%
5157 {\bbl@info{No hyphenation patterns were set for\%
5158 language '#2'. Reported}}%
5159 {\expandafter\expandafter\expandafter\bbl@luapatterns
5160 \csname bbl@hyphendata@the\language\endcsname}}}%
5161 \ifundefined{bbl@patterns@}{}%
5162 \begingroup
5163 \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
5164 \ifin@else
5165 \ifx\bbl@patterns@\empty\else
5166 \directlua{ Babel.addpatterns(
5167 [[\bbl@patterns@]], \number\language) }%
5168 \fi
5169 \ifundefined{bbl@patterns@#1}%
5170 \empty
5171 {\directlua{ Babel.addpatterns(
5172 [[\space\csname bbl@patterns@#1\endcsname]],
5173 \number\language) }}%
5174 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5175 \fi
5176 \endgroup}%
5177 \bbl@exp{%
5178 \bbl@ifunset{bbl@prehc@\languagename}{}%
5179 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5180 {\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.

```

5181 \@onlypreamble\babelpatterns
5182 \AtEndOfPackage{%
5183 \newcommand\babelpatterns[2][\empty]{%
5184 \ifx\bbl@patterns@\relax
5185 \let\bbl@patterns@\empty
5186 \fi
5187 \ifx\bbl@pttnlist\empty\else
5188 \bbl@warning{%
5189 You must not intermingle \string\selectlanguage\space and\%
5190 \string\babelpatterns\space or some patterns will not\%
5191 be taken into account. Reported}%
5192 \fi

```

```

5193 \ifx\@empty#1%
5194 \protected@edef\bbl@patterns@\bbl@patterns@space#2}%
5195 \else
5196 \edef\bbl@tempb{\zap@space#1 \@empty}%
5197 \bbl@for\bbl@tempa\bbl@tempb{%
5198 \bbl@fixname\bbl@tempa
5199 \bbl@iflanguage\bbl@tempa{%
5200 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5201 \@ifundefined{bbl@patterns@\bbl@tempa}%
5202 \@empty
5203 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5204 #2}}}%
5205 \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. Replace regular (ie, implicit) discretionary by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary are not touched. See Unicode UAX 14.

```

5206 % TODO - to a lua file
5207 \directlua{
5208   Babel = Babel or {}
5209   Babel.linebreaking = Babel.linebreaking or {}
5210   Babel.linebreaking.before = {}
5211   Babel.linebreaking.after = {}
5212   Babel.locale = {} % Free to use, indexed by \localeid
5213   function Babel.linebreaking.add_before(func)
5214     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5215     table.insert(Babel.linebreaking.before, func)
5216   end
5217   function Babel.linebreaking.add_after(func)
5218     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5219     table.insert(Babel.linebreaking.after, func)
5220   end
5221 }
5222 \def\bbl@intraspace#1 #2 #3\@@{%
5223 \directlua{
5224   Babel = Babel or {}
5225   Babel.intraspaces = Babel.intraspaces or {}
5226   Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5227     {b = #1, p = #2, m = #3}
5228   Babel.locale_props[\the\localeid].intraspace = %
5229     {b = #1, p = #2, m = #3}
5230 }}
5231 \def\bbl@intrapenalty#1\@@{%
5232 \directlua{
5233   Babel = Babel or {}
5234   Babel.intrapenalties = Babel.intrapenalties or {}
5235   Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5236   Babel.locale_props[\the\localeid].intrapenalty = #1
5237 }}
5238 \begingroup
5239 \catcode`\%=12
5240 \catcode`\^=14
5241 \catcode`\'=12
5242 \catcode`\~=12
5243 \gdef\bbl@seaintraspace^

```

```

5244 \let\bbl@seaintraspace\relax
5245 \directlua{
5246   Babel = Babel or {}
5247   Babel.sea_enabled = true
5248   Babel.sea_ranges = Babel.sea_ranges or {}
5249   function Babel.set_chranges (script, chrng)
5250     local c = 0
5251     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5252       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5253       c = c + 1
5254     end
5255   end
5256   function Babel.sea_disc_to_space (head)
5257     local sea_ranges = Babel.sea_ranges
5258     local last_char = nil
5259     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5260     for item in node.traverse(head) do
5261       local i = item.id
5262       if i == node.id'glyph' then
5263         last_char = item
5264       elseif i == 7 and item.subtype == 3 and last_char
5265         and last_char.char > 0x0C99 then
5266         quad = font.getfont(last_char.font).size
5267         for lg, rg in pairs(sea_ranges) do
5268           if last_char.char > rg[1] and last_char.char < rg[2] then
5269             lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5270             local intraspace = Babel.intraspaces[lg]
5271             local intrapenalty = Babel.intrapenalties[lg]
5272             local n
5273             if intrapenalty ~= 0 then
5274               n = node.new(14, 0)      ^% penalty
5275               n.penalty = intrapenalty
5276               node.insert_before(head, item, n)
5277             end
5278             n = node.new(12, 13)      ^% (glue, spaceskip)
5279             node.setglue(n, intraspace.b * quad,
5280               intraspace.p * quad,
5281               intraspace.m * quad)
5282             node.insert_before(head, item, n)
5283             node.remove(head, item)
5284           end
5285         end
5286       end
5287     end
5288   end
5289 }^^
5290 \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.

```

5291 \catcode`\%=14
5292 \gdef\bbl@cjkintraspaces{%

```

```

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

```

```

5352 }%
5353 \bbl@luahyphenate}
5354 \gdef\bbl@luahyphenate{%
5355   \let\bbl@luahyphenate\relax
5356   \directlua{
5357     luatexbase.add_to_callback('hyphenate',
5358       function (head, tail)
5359         if Babel.linebreaking.before then
5360           for k, func in ipairs(Babel.linebreaking.before) do
5361             func(head)
5362           end
5363         end
5364         if Babel.cjk_enabled then
5365           Babel.cjk_linebreak(head)
5366         end
5367         lang.hyphenate(head)
5368         if Babel.linebreaking.after then
5369           for k, func in ipairs(Babel.linebreaking.after) do
5370             func(head)
5371           end
5372         end
5373         if Babel.sea_enabled then
5374           Babel.sea_disc_to_space(head)
5375         end
5376       end,
5377       'Babel.hyphenate')
5378   }
5379 }
5380 \endgroup
5381 \def\bbl@provide@intraspace{%
5382   \bbl@ifunset{bbl@intsp@language}{}%
5383   {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5384     \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5385     \ifin@           % cjk
5386     \bbl@cjk_intraspace
5387     \directlua{
5388       Babel = Babel or {}
5389       Babel.locale_props = Babel.locale_props or {}
5390       Babel.locale_props[\the\localeid].linebreak = 'c'
5391     }%
5392     \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\\@}%
5393     \ifx\bbl@KVP@intrapenalty\@nil
5394       \bbl@intrapenalty0\@@
5395     \fi
5396   \else           % sea
5397     \bbl@sea_intraspace
5398     \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\\@}%
5399     \directlua{
5400       Babel = Babel or {}
5401       Babel.sea_ranges = Babel.sea_ranges or {}
5402       Babel.set_chranges('\bbl@cl{sbcpr}',
5403         '\bbl@cl{chrng}')
5404     }%
5405     \ifx\bbl@KVP@intrapenalty\@nil
5406       \bbl@intrapenalty0\@@
5407     \fi
5408   \fi
5409 \fi
5410 \ifx\bbl@KVP@intrapenalty\@nil\else

```

```

5411 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
5412 \fi}}

```

13.6 Arabic justification

```

5413 \ifnum\bb1@bidimode>100 \ifnum\bb1@bidimode<200
5414 \def\bb1ar@chars{%
5415   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5416   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5417   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5418 \def\bb1ar@elongated{%
5419   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5420   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5421   0649,064A}
5422 \begingroup
5423 \catcode\_ =11 \catcode\`:=11
5424 \gdef\bb1ar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5425 \endgroup
5426 \gdef\bb1@arabicjust{%
5427   \let\bb1@arabicjust\relax
5428   \newattribute\bb1ar@kashida
5429   \bb1ar@kashida=\z@
5430   \expandafter\bb1@add\csname selectfont \endcsname{{\bb1@parsejalt}}%
5431   \directlua{
5432     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5433     Babel.arabic.elong_map[\the\localeid] = {}
5434     luatexbase.add_to_callback('post_linebreak_filter',
5435       Babel.arabic.justify, 'Babel.arabic.justify')
5436     luatexbase.add_to_callback('hpack_filter',
5437       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5438   }}%
5439 % Save both node lists to make replacement. TODO. Save also widths to
5440 % make computations
5441 \def\bb1ar@fetchjalt#1#2#3#4{%
5442   \bb1@exp{\bb1@foreach{#1}}{%
5443     \bb1@ifunset{bb1ar@JE@##1}%
5444     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5445     {\setbox\z@\hbox{^^^200d\char"\@nameuse{bb1ar@JE@##1}#2}}%
5446     \directlua{%
5447       local last = nil
5448       for item in node.traverse(tex.box[0].head) do
5449         if item.id == node.id'glyph' and item.char > 0x600 and
5450           not (item.char == 0x200D) then
5451           last = item
5452         end
5453       end
5454       Babel.arabic.#3['##1#4'] = last.char
5455     }}%
5456 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5457 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5458 % positioning?
5459 \gdef\bb1@parsejalt{%
5460   \ifx\addfontfeature\undefined\else
5461     \bb1@xin@{/e}{/\bb1@cl{lnbrk}}%
5462   \fin@
5463   \directlua{%
5464     if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5465       Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5466       tex.print([[string\csname\space bb1@parsejalt\endcsname]])

```

```

5467         end
5468     }%
5469     \fi
5470 \fi}
5471 \gdef\bbl@parsejalti{%
5472     \begingroup
5473     \let\bbl@parsejalt\relax    % To avoid infinite loop
5474     \edef\bbl@tempb{\fontid\font}%
5475     \bblar@nofswarn
5476     \bblar@fetchjalt\bblar@elongated{}{}{}%
5477     \bblar@fetchjalt\bblar@chars{^^^^064a}{}{}% Alef maksura
5478     \bblar@fetchjalt\bblar@chars{^^^^0649}{}{}% Yeh
5479     \addfontfeature{RawFeature+=jalt}%
5480     % \namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5481     \bblar@fetchjalt\bblar@elongated{}{}{}%
5482     \bblar@fetchjalt\bblar@chars{^^^^064a}{}{}%
5483     \bblar@fetchjalt\bblar@chars{^^^^0649}{}{}%
5484     \directlua{%
5485         for k, v in pairs(Babel.arabic.from) do
5486             if Babel.arabic.dest[k] and
5487                 not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5488                 Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5489                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5490             end
5491         end
5492     }%
5493     \endgroup}
5494 %
5495 \begingroup
5496 \catcode`#=11
5497 \catcode`~ =11
5498 \directlua{
5499
5500 Babel.arabic = Babel.arabic or {}
5501 Babel.arabic.from = {}
5502 Babel.arabic.dest = {}
5503 Babel.arabic.justify_factor = 0.95
5504 Babel.arabic.justify_enabled = true
5505
5506 function Babel.arabic.justify(head)
5507     if not Babel.arabic.justify_enabled then return head end
5508     for line in node.traverse_id(node.id'hlist', head) do
5509         Babel.arabic.justify_hlist(head, line)
5510     end
5511     return head
5512 end
5513
5514 function Babel.arabic.justify_hbox(head, gc, size, pack)
5515     local has_inf = false
5516     if Babel.arabic.justify_enabled and pack == 'exactly' then
5517         for n in node.traverse_id(12, head) do
5518             if n.stretch_order > 0 then has_inf = true end
5519         end
5520         if not has_inf then
5521             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5522         end
5523     end
5524     return head
5525 end

```

```

5526
5527 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5528   local d, new
5529   local k_list, k_item, pos_inline
5530   local width, width_new, full, k_curr, wt_pos, goal, shift
5531   local subst_done = false
5532   local elong_map = Babel.arabic.elong_map
5533   local last_line
5534   local GLYPH = node.id'glyph'
5535   local KASHIDA = luatexbase.registernumber'bblar@kashida'
5536   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5537
5538   if line == nil then
5539     line = {}
5540     line.glue_sign = 1
5541     line.glue_order = 0
5542     line.head = head
5543     line.shift = 0
5544     line.width = size
5545   end
5546
5547   % Exclude last line. todo. But-- it discards one-word lines, too!
5548   % ? Look for glue = 12:15
5549   if (line.glue_sign == 1 and line.glue_order == 0) then
5550     elongs = {} % Stores elongated candidates of each line
5551     k_list = {} % And all letters with kashida
5552     pos_inline = 0 % Not yet used
5553
5554     for n in node.traverse_id(GLYPH, line.head) do
5555       pos_inline = pos_inline + 1 % To find where it is. Not used.
5556
5557       % Elongated glyphs
5558       if elong_map then
5559         local locale = node.get_attribute(n, LOCALE)
5560         if elong_map[locale] and elong_map[locale][n.font] and
5561           elong_map[locale][n.font][n.char] then
5562           table.insert(elongs, {node = n, locale = locale} )
5563           node.set_attribute(n.prev, KASHIDA, 0)
5564         end
5565       end
5566
5567       % Tatwil
5568       if Babel.kashida_wts then
5569         local k_wt = node.get_attribute(n, KASHIDA)
5570         if k_wt > 0 then % todo. parameter for multi inserts
5571           table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5572         end
5573       end
5574
5575     end % of node.traverse_id
5576
5577     if #elongs == 0 and #k_list == 0 then goto next_line end
5578     full = line.width
5579     shift = line.shift
5580     goal = full * Babel.arabic.justify_factor % A bit crude
5581     width = node.dimensions(line.head) % The 'natural' width
5582
5583     % == Elongated ==
5584     % Original idea taken from 'chickenize'

```



```

5585 while (#elongs > 0 and width < goal) do
5586     subst_done = true
5587     local x = #elongs
5588     local curr = elongs[x].node
5589     local oldchar = curr.char
5590     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5591     width = node.dimensions(line.head) % Check if the line is too wide
5592     % Substitute back if the line would be too wide and break:
5593     if width > goal then
5594         curr.char = oldchar
5595         break
5596     end
5597     % If continue, pop the just substituted node from the list:
5598     table.remove(elongs, x)
5599 end
5600
5601 % == Tatwil ==
5602 if #k_list == 0 then goto next_line end
5603
5604 width = node.dimensions(line.head) % The 'natural' width
5605 k_curr = #k_list
5606 wt_pos = 1
5607
5608 while width < goal do
5609     subst_done = true
5610     k_item = k_list[k_curr].node
5611     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5612         d = node.copy(k_item)
5613         d.char = 0x0640
5614         line.head, new = node.insert_after(line.head, k_item, d)
5615         width_new = node.dimensions(line.head)
5616         if width > goal or width == width_new then
5617             node.remove(line.head, new) % Better compute before
5618             break
5619         end
5620         width = width_new
5621     end
5622     if k_curr == 1 then
5623         k_curr = #k_list
5624         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5625     else
5626         k_curr = k_curr - 1
5627     end
5628 end
5629
5630 ::next_line::
5631
5632 % Must take into account marks and ins, see luatex manual.
5633 % Have to be executed only if there are changes. Investigate
5634 % what's going on exactly.
5635 if subst_done and not gc then
5636     d = node.hpack(line.head, full, 'exactly')
5637     d.shift = shift
5638     node.insert_before(head, line, d)
5639     node.remove(head, line)
5640 end
5641 end % if process line
5642 end
5643 }

```

```

5644 \endgroup
5645 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5646 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5647 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5648 \DisableBabelHook{babel-fontspec}
5649 <<Font selection>>

```

13.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5650 % TODO - to a lua file
5651 \directlua{
5652 Babel.script_blocks = {
5653   ['dflt'] = {},
5654   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5655               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5656   ['Armn'] = {{0x0530, 0x058F}},
5657   ['Beng'] = {{0x0980, 0x09FF}},
5658   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5659   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5660   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5661               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5662   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5663   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5664               {0xAB00, 0xAB2F}},
5665   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5666   % Don't follow strictly Unicode, which places some Coptic letters in
5667   % the 'Greek and Coptic' block
5668   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5669   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5670               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5671               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5672               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5673               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5674               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5675   ['Hebr'] = {{0x0590, 0x05FF}},
5676   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5677               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5678   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5679   ['Knda'] = {{0x0C80, 0x0CFF}},
5680   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5681               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5682               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5683   ['Lao0'] = {{0x0E80, 0x0EFF}},
5684   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5685               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5686               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5687   ['Mahj'] = {{0x11150, 0x1117F}},
5688   ['Mlym'] = {{0x0D00, 0x0D7F}},
5689   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5690   ['Orya'] = {{0x0B00, 0x0B7F}},

```

```

5691 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5692 ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5693 ['Taml'] = {{0x0B80, 0x0BFF}},
5694 ['Telu'] = {{0x0C00, 0x0C7F}},
5695 ['Tfng'] = {{0x2D30, 0x2D7F}},
5696 ['Thai'] = {{0x0E00, 0x0E7F}},
5697 ['Tibt'] = {{0x0F00, 0x0FFF}},
5698 ['Vaii'] = {{0xA500, 0xA63F}},
5699 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5700 }
5701
5702 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5703 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5704 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5705
5706 function Babel.locale_map(head)
5707   if not Babel.locale_mapped then return head end
5708
5709   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5710   local GLYPH = node.id('glyph')
5711   local inmath = false
5712   local toloc_save
5713   for item in node.traverse(head) do
5714     local toloc
5715     if not inmath and item.id == GLYPH then
5716       % Optimization: build a table with the chars found
5717       if Babel.chr_to_loc[item.char] then
5718         toloc = Babel.chr_to_loc[item.char]
5719       else
5720         for lc, maps in pairs(Babel.loc_to_scr) do
5721           for _, rg in pairs(maps) do
5722             if item.char >= rg[1] and item.char <= rg[2] then
5723               Babel.chr_to_loc[item.char] = lc
5724               toloc = lc
5725               break
5726             end
5727           end
5728         end
5729       end
5730       % Now, take action, but treat composite chars in a different
5731       % fashion, because they 'inherit' the previous locale. Not yet
5732       % optimized.
5733       if not toloc and
5734         (item.char >= 0x0300 and item.char <= 0x036F) or
5735         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5736         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5737         toloc = toloc_save
5738       end
5739       if toloc and toloc > -1 then
5740         if Babel.locale_props[toloc].lg then
5741           item.lang = Babel.locale_props[toloc].lg
5742           node.set_attribute(item, LOCALE, toloc)
5743         end
5744         if Babel.locale_props[toloc]['/'..item.font] then
5745           item.font = Babel.locale_props[toloc]['/'..item.font]
5746         end
5747         toloc_save = toloc
5748       end
5749     elseif not inmath and item.id == 7 then

```

```

5750     item.replace = item.replace and Babel.locale_map(item.replace)
5751     item.pre      = item.pre and Babel.locale_map(item.pre)
5752     item.post     = item.post and Babel.locale_map(item.post)
5753     elseif item.id == node.id'math' then
5754         inmath = (item.subtype == 0)
5755     end
5756 end
5757 return head
5758 end
5759 }

```

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

```

5760 \newcommand\babelcharproperty[1]{%
5761   \count@=#1\relax
5762   \ifvmode
5763     \expandafter\bbl@chprop
5764   \else
5765     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5766               vertical mode (preamble or between paragraphs)}%
5767     {See the manual for futher info}%
5768   \fi}
5769 \newcommand\bbl@chprop[3][\the\count@]{%
5770   \@tempcnta=#1\relax
5771   \bbl@ifunset{\bbl@chprop@#2}%
5772   {\bbl@error{No property named '#2'. Allowed values are\\%
5773             direction (bc), mirror (bmg), and linebreak (lb)}%
5774    {See the manual for futher info}}%
5775   }%
5776   \loop
5777     \bbl@cs{chprop@#2}{#3}%
5778   \ifnum\count@<\@tempcnta
5779     \advance\count@\@ne
5780   \repeat}
5781 \def\bbl@chprop@direction#1{%
5782   \directlua{
5783     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5784     Babel.characters[\the\count@]['d'] = '#1'
5785   }}
5786 \let\bbl@chprop@bc\bbl@chprop@direction
5787 \def\bbl@chprop@mirror#1{%
5788   \directlua{
5789     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5790     Babel.characters[\the\count@]['m'] = '\number#1'
5791   }}
5792 \let\bbl@chprop@bmg\bbl@chprop@mirror
5793 \def\bbl@chprop@linebreak#1{%
5794   \directlua{
5795     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5796     Babel.cjk_characters[\the\count@]['c'] = '#1'
5797   }}
5798 \let\bbl@chprop@lb\bbl@chprop@linebreak
5799 \def\bbl@chprop@locale#1{%
5800   \directlua{
5801     Babel.chr_to_loc = Babel.chr_to_loc or {}
5802     Babel.chr_to_loc[\the\count@] =
5803       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5804   }}

```

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).

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 discretionary, 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.

```
5805 \begingroup % TODO - to a lua file
5806 \catcode`\~=12
5807 \catcode`\#=12
5808 \catcode`\%=12
5809 \catcode`\&=14
5810 \directlua{
5811   Babel.linebreaking.replacements = {}
5812   Babel.linebreaking.replacements[0] = {} &% pre
5813   Babel.linebreaking.replacements[1] = {} &% post
5814
5815   &% Discretionaries contain strings as nodes
5816   function Babel.str_to_nodes(fn, matches, base)
5817     local n, head, last
5818     if fn == nil then return nil end
5819     for s in string.utfvalues(fn(matches)) do
5820       if base.id == 7 then
5821         base = base.replace
5822       end
5823       n = node.copy(base)
5824       n.char = s
5825       if not head then
5826         head = n
5827       else
5828         last.next = n
5829       end
5830       last = n
5831     end
5832     return head
5833   end
5834
5835   Babel.fetch_subtext = {}
5836
5837   Babel.ignore_pre_char = function(node)
5838     return (node.lang == \the\l@nohyphenation)
5839   end
5840
5841   &% Merging both functions doesn't seem feasible, because there are too
5842   &% many differences.
5843   Babel.fetch_subtext[0] = function(head)
5844     local word_string = ''
5845     local word_nodes = {}
5846     local lang
5847     local item = head
5848     local inmath = false
5849
5850     while item do
```

```

5851
5852     if item.id == 11 then
5853         inmath = (item.subtype == 0)
5854     end
5855
5856     if inmath then
5857         &% pass
5858
5859     elseif item.id == 29 then
5860         local locale = node.get_attribute(item, Babel.attr_locale)
5861
5862         if lang == locale or lang == nil then
5863             lang = lang or locale
5864             if Babel.ignore_pre_char(item) then
5865                 word_string = word_string .. Babel.us_char
5866             else
5867                 word_string = word_string .. unicode.utf8.char(item.char)
5868             end
5869             word_nodes[#word_nodes+1] = item
5870         else
5871             break
5872         end
5873
5874     elseif item.id == 12 and item.subtype == 13 then
5875         word_string = word_string .. ' '
5876         word_nodes[#word_nodes+1] = item
5877
5878         &% Ignore leading unrecognized nodes, too.
5879         elseif word_string ~= '' then
5880             word_string = word_string .. Babel.us_char
5881             word_nodes[#word_nodes+1] = item &% Will be ignored
5882         end
5883
5884         item = item.next
5885     end
5886
5887     &% Here and above we remove some trailing chars but not the
5888     &% corresponding nodes. But they aren't accessed.
5889     if word_string:sub(-1) == ' ' then
5890         word_string = word_string:sub(1,-2)
5891     end
5892     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5893     return word_string, word_nodes, item, lang
5894 end
5895
5896 Babel.fetch_subtext[1] = function(head)
5897     local word_string = ''
5898     local word_nodes = {}
5899     local lang
5900     local item = head
5901     local inmath = false
5902
5903     while item do
5904
5905         if item.id == 11 then
5906             inmath = (item.subtype == 0)
5907         end
5908
5909         if inmath then

```

```

5910         %% pass
5911
5912     elseif item.id == 29 then
5913         if item.lang == lang or lang == nil then
5914             if (item.char ~= 124) and (item.char ~= 61) then %% not =, not |
5915                 lang = lang or item.lang
5916                 word_string = word_string .. unicode.utf8.char(item.char)
5917                 word_nodes[#word_nodes+1] = item
5918             end
5919         else
5920             break
5921         end
5922
5923     elseif item.id == 7 and item.subtype == 2 then
5924         word_string = word_string .. '='
5925         word_nodes[#word_nodes+1] = item
5926
5927     elseif item.id == 7 and item.subtype == 3 then
5928         word_string = word_string .. '|'
5929         word_nodes[#word_nodes+1] = item
5930
5931         %% (1) Go to next word if nothing was found, and (2) implicitly
5932         %% remove leading USs.
5933         elseif word_string == '' then
5934             %% pass
5935
5936         %% This is the responsible for splitting by words.
5937         elseif (item.id == 12 and item.subtype == 13) then
5938             break
5939
5940         else
5941             word_string = word_string .. Babel.us_char
5942             word_nodes[#word_nodes+1] = item %% Will be ignored
5943         end
5944
5945         item = item.next
5946     end
5947
5948     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5949     return word_string, word_nodes, item, lang
5950 end
5951
5952 function Babel.pre_hyphenate_replace(head)
5953     Babel.hyphenate_replace(head, 0)
5954 end
5955
5956 function Babel.post_hyphenate_replace(head)
5957     Babel.hyphenate_replace(head, 1)
5958 end
5959
5960 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5961     local ss = ''
5962     for pp = 1, 40 do
5963         if wn[pp] then
5964             if wn[pp].id == 29 then
5965                 ss = ss .. unicode.utf8.char(wn[pp].char)
5966             else
5967                 ss = ss .. '{' .. wn[pp].id .. '}'
5968             end

```

```

5969     end
5970 end
5971 print('nod', ss)
5972 print('lst_m',
5973       string.rep(' ', unicode.utf8.len(
5974         string.sub(w, 1, last_match))-1) .. '>')
5975 print('str', w)
5976 print('sc', string.rep(' ', sc-1) .. '^')
5977 if first == last then
5978   print('f=l', string.rep(' ', first-1) .. '!')
5979 else
5980   print('f/l', string.rep(' ', first-1) .. '[' ..
5981     string.rep(' ', last-first-1) .. ']')
5982 end
5983 end
5984
5985 Babel.us_char = string.char(31)
5986
5987 function Babel.hyphenate_replace(head, mode)
5988   local u = unicode.utf8
5989   local lbkr = Babel.linebreaking.replacements[mode]
5990
5991   local word_head = head
5992
5993   while true do  &% for each subtext block
5994
5995     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5996
5997     if Babel.debug then
5998       print()
5999       print((mode == 0) and '@@@<' or '@@@>', w)
6000     end
6001
6002     if nw == nil and w == '' then break end
6003
6004     if not lang then goto next end
6005     if not lbkr[lang] then goto next end
6006
6007     &% For each saved (pre|post)hyphenation. TODO. Reconsider how
6008     &% loops are nested.
6009     for k=1, #lbkr[lang] do
6010       local p = lbkr[lang][k].pattern
6011       local r = lbkr[lang][k].replace
6012
6013       if Babel.debug then
6014         print('*****', p, mode)
6015       end
6016
6017       &% This variable is set in some cases below to the first *byte*
6018       &% after the match, either as found by u.match (faster) or the
6019       &% computed position based on sc if w has changed.
6020       local last_match = 0
6021       local step = 0
6022
6023       &% For every match.
6024       while true do
6025         if Babel.debug then
6026           print('====')
6027         end

```



```

6028         local new  %% used when inserting and removing nodes
6029
6030         local matches = { u.match(w, p, last_match) }
6031
6032         if #matches < 2 then break end
6033
6034         %% Get and remove empty captures (with ())'s, which return a
6035         %% number with the position), and keep actual captures
6036         %% (from (...)), if any, in matches.
6037         local first = table.remove(matches, 1)
6038         local last  = table.remove(matches, #matches)
6039         %% Non re-fetched substrings may contain \31, which separates
6040         %% subsubstrings.
6041         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6042
6043         local save_last = last %% with A()BC()D, points to D
6044
6045         %% Fix offsets, from bytes to unicode. Explained above.
6046         first = u.len(w:sub(1, first-1)) + 1
6047         last  = u.len(w:sub(1, last-1)) %% now last points to C
6048
6049         %% This loop stores in n small table the nodes
6050         %% corresponding to the pattern. Used by 'data' to provide a
6051         %% predictable behavior with 'insert' (now w_nodes is modified on
6052         %% the fly), and also access to 'remove'd nodes.
6053         local sc = first-1          %% Used below, too
6054         local data_nodes = {}
6055
6056         for q = 1, last-first+1 do
6057             data_nodes[q] = w_nodes[sc+q]
6058         end
6059
6060         %% This loop traverses the matched substring and takes the
6061         %% corresponding action stored in the replacement list.
6062         %% sc = the position in substr nodes / string
6063         %% rc = the replacement table index
6064         local rc = 0
6065
6066         while rc < last-first+1 do %% for each replacement
6067             if Babel.debug then
6068                 print('.....', rc + 1)
6069             end
6070             sc = sc + 1
6071             rc = rc + 1
6072
6073             if Babel.debug then
6074                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6075                 local ss = ''
6076                 for itt in node.traverse(head) do
6077                     if itt.id == 29 then
6078                         ss = ss .. unicode.utf8.char(itt.char)
6079                     else
6080                         ss = ss .. '{' .. itt.id .. '}'
6081                     end
6082                 end
6083                 print('*****', ss)
6084             end
6085         end
6086

```

```

6087     local crep = r[rc]
6088     local item = w_nodes[sc]
6089     local item_base = item
6090     local placeholder = Babel.us_char
6091     local d
6092
6093     if crep and crep.data then
6094         item_base = data_nodes[crep.data]
6095     end
6096
6097     if crep then
6098         step = crep.step or 0
6099     end
6100
6101     if crep and next(crep) == nil then &% = {}
6102         last_match = save_last    &% Optimization
6103         goto next
6104
6105     elseif crep == nil or crep.remove then
6106         node.remove(head, item)
6107         table.remove(w_nodes, sc)
6108         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6109         sc = sc - 1    &% Nothing has been inserted.
6110         last_match = utf8.offset(w, sc+1+step)
6111         goto next
6112
6113     elseif crep and crep.kashida then &% Experimental
6114         node.set_attribute(item,
6115             luatexbase.registernumber'bblar@kashida',
6116             crep.kashida)
6117         last_match = utf8.offset(w, sc+1+step)
6118         goto next
6119
6120     elseif crep and crep.string then
6121         local str = crep.string(matches)
6122         if str == '' then &% Gather with nil
6123             node.remove(head, item)
6124             table.remove(w_nodes, sc)
6125             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6126             sc = sc - 1    &% Nothing has been inserted.
6127         else
6128             local loop_first = true
6129             for s in string.utfvalues(str) do
6130                 d = node.copy(item_base)
6131                 d.char = s
6132                 if loop_first then
6133                     loop_first = false
6134                     head, new = node.insert_before(head, item, d)
6135                     if sc == 1 then
6136                         word_head = head
6137                     end
6138                     w_nodes[sc] = d
6139                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6140                 else
6141                     sc = sc + 1
6142                     head, new = node.insert_before(head, item, d)
6143                     table.insert(w_nodes, sc, new)
6144                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6145                 end
6146             end
6147         end
6148     end

```

```

6146         if Babel.debug then
6147             print('....', 'str')
6148             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6149         end
6150     end && for
6151         node.remove(head, item)
6152     end && if ''
6153     last_match = utf8.offset(w, sc+1+step)
6154     goto next
6155
6156 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6157     d = node.new(7, 0) && (disc, discretionary)
6158     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6159     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6160     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6161     d.attr = item_base.attr
6162     if crep.pre == nil then && TeXbook p96
6163         d.penalty = crep.penalty or tex.hyphenpenalty
6164     else
6165         d.penalty = crep.penalty or tex.exhyphenpenalty
6166     end
6167     placeholder = '|'
6168     head, new = node.insert_before(head, item, d)
6169
6170 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6171     && ERROR
6172
6173 elseif crep and crep.penalty then
6174     d = node.new(14, 0) && (penalty, userpenalty)
6175     d.attr = item_base.attr
6176     d.penalty = crep.penalty
6177     head, new = node.insert_before(head, item, d)
6178
6179 elseif crep and crep.space then
6180     && 655360 = 10 pt = 10 * 65536 sp
6181     d = node.new(12, 13) && (glue, spaceskip)
6182     local quad = font.getfont(item_base.font).size or 655360
6183     node.setglue(d, crep.space[1] * quad,
6184                  crep.space[2] * quad,
6185                  crep.space[3] * quad)
6186     if mode == 0 then
6187         placeholder = ' '
6188     end
6189     head, new = node.insert_before(head, item, d)
6190
6191 elseif crep and crep.spacefactor then
6192     d = node.new(12, 13) && (glue, spaceskip)
6193     local base_font = font.getfont(item_base.font)
6194     node.setglue(d,
6195                  crep.spacefactor[1] * base_font.parameters['space'],
6196                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6197                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6198     if mode == 0 then
6199         placeholder = ' '
6200     end
6201     head, new = node.insert_before(head, item, d)
6202
6203 elseif mode == 0 and crep and crep.space then
6204     && ERROR

```

```

6205
6206         end    %% ie replacement cases
6207
6208         %% Shared by disc, space and penalty.
6209         if sc == 1 then
6210             word_head = head
6211         end
6212         if crep.insert then
6213             w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6214             table.insert(w_nodes, sc, new)
6215             last = last + 1
6216         else
6217             w_nodes[sc] = d
6218             node.remove(head, item)
6219             w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6220         end
6221
6222         last_match = utf8.offset(w, sc+1+step)
6223
6224         ::next::
6225
6226     end    %% for each replacement
6227
6228     if Babel.debug then
6229         print('.....', '/')
6230         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6231     end
6232
6233     end    %% for match
6234
6235     end    %% for patterns
6236
6237     ::next::
6238     word_head = nw
6239     end    %% for substring
6240     return head
6241 end
6242
6243 %% This table stores capture maps, numbered consecutively
6244 Babel.capture_maps = {}
6245
6246 %% The following functions belong to the next macro
6247 function Babel.capture_func(key, cap)
6248     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
6249     local cnt
6250     local u = unicode.utf8
6251     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(-)}}', Babel.capture_func_map)
6252     if cnt == 0 then
6253         ret = u.gsub(ret, '{{(x%x%x%x)}}',
6254             function (n)
6255                 return u.char(tonumber(n, 16))
6256             end)
6257     end
6258     ret = ret:gsub("%[%[%]]%.%", '')
6259     ret = ret:gsub("%.%.%[%[%]]", '')
6260     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6261 end
6262
6263 function Babel.capt_map(from, mapno)

```

```

6264     return Babel.capture_maps[mapno][from] or from
6265 end
6266
6267 &% Handle the {n|abc|ABC} syntax in captures
6268 function Babel.capture_func_map(capno, from, to)
6269     local u = unicode.utf8
6270     from = u.gsub(from, '{(%x%x%x%x+)}',
6271         function (n)
6272             return u.char(tonumber(n, 16))
6273         end)
6274     to = u.gsub(to, '{(%x%x%x%x+)}',
6275         function (n)
6276             return u.char(tonumber(n, 16))
6277         end)
6278     local froms = {}
6279     for s in string.utfcharacters(from) do
6280         table.insert(froms, s)
6281     end
6282     local cnt = 1
6283     table.insert(Babel.capture_maps, {})
6284     local mlen = table.getn(Babel.capture_maps)
6285     for s in string.utfcharacters(to) do
6286         Babel.capture_maps[mlen][froms[cnt]] = s
6287         cnt = cnt + 1
6288     end
6289     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6290         (mlen) .. ").. " .. "["
6291 end
6292
6293 &% Create/Extend reversed sorted list of kashida weights:
6294 function Babel.capture_kashida(key, wt)
6295     wt = tonumber(wt)
6296     if Babel.kashida_wts then
6297         for p, q in ipairs(Babel.kashida_wts) do
6298             if wt == q then
6299                 break
6300             elseif wt > q then
6301                 table.insert(Babel.kashida_wts, p, wt)
6302                 break
6303             elseif table.getn(Babel.kashida_wts) == p then
6304                 table.insert(Babel.kashida_wts, wt)
6305             end
6306         end
6307     else
6308         Babel.kashida_wts = { wt }
6309     end
6310     return 'kashida = ' .. wt
6311 end
6312 }

```

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

```

6313 \catcode`\#=6
6314 \gdef\babelposthyphenation#1#2#3{&%
6315   \bbl@activateposthyphen
6316   \beginingroup
6317     \def\babeltempa{\bbl@add@list\babeltempb}&%
6318     \let\babeltempb\@empty
6319     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6320     \bbl@replace\bbl@tempa{,}{ ,}&%
6321     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6322       \bbl@ifsamestring{##1}{remove}&%
6323       {\bbl@add@list\babeltempb{nil}}&%
6324       {\directlua{
6325         local rep = {[##1]=]
6326         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6327         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6328         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6329         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6330         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6331         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
6332         tex.print([[\\string\babeltempa{}}] .. rep .. [[]]})
6333       }}&%
6334     \directlua{
6335       local lbkr = Babel.linebreaking.replacements[1]
6336       local u = unicode.utf8
6337       local id = \the\csname l@#1\endcsname
6338       &% Convert pattern:
6339       local patt = string.gsub(==[#2]==, '%s', '')
6340       if not u.find(patt, '('), nil, true) then
6341         patt = '(' .. patt .. ')'
6342       end
6343       patt = string.gsub(patt, '%(%)%^\', '^()')
6344       patt = string.gsub(patt, '%$(%)', '()$')
6345       patt = u.gsub(patt, '{(.)}',
6346         function (n)
6347           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6348         end)
6349       patt = u.gsub(patt, '{(%x%x%x%x%+)}',
6350         function (n)
6351           return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6352         end)
6353       lbkr[id] = lbkr[id] or {}
6354       table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6355     }&%
6356   \endgroup}
6357 % TODO. Copypaste pattern.
6358 \gdef\babelprehyphenation#1#2#3{&%
6359   \bbl@activateprehyphen
6360   \beginingroup
6361     \def\babeltempa{\bbl@add@list\babeltempb}&%
6362     \let\babeltempb\@empty
6363     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6364     \bbl@replace\bbl@tempa{,}{ ,}&%
6365     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6366       \bbl@ifsamestring{##1}{remove}&%
6367       {\bbl@add@list\babeltempb{nil}}&%
6368       {\directlua{
6369         local rep = {[##1]=]
6370         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6371         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')

```

```

6372         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6373         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6374             'space = {' .. '%2, %3, %4' .. '}')
6375         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6376             'spacefactor = {' .. '%2, %3, %4' .. '}')
6377         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6378         tex.print([[\\string\\babeltempa{[]] .. rep .. [{}]])
6379     }}&%
6380 \\directlua{
6381     local lbkr = Babel.linebreaking.replacements[0]
6382     local u = unicode.utf8
6383     local id = \\the\\csname bbl@id@#1\\endcsname
6384     &% Convert pattern:
6385     local patt = string.gsub(==[#2]==, '%s', '')
6386     local patt = string.gsub(patt, '|', ' ')
6387     if not u.find(patt, '()', nil, true) then
6388         patt = '()' .. patt .. '()'
6389     end
6390     &% patt = string.gsub(patt, '%(%)%', '^()')
6391     &% patt = string.gsub(patt, '([^\s])%$%', '%1()$')
6392     patt = u.gsub(patt, '{(.)}',
6393         function (n)
6394             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6395         end)
6396     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6397         function (n)
6398             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6399         end)
6400     lbkr[id] = lbkr[id] or {}
6401     table.insert(lbkr[id], { pattern = patt, replace = { \\babeltempb } })
6402 }&%
6403 \\endgroup}
6404 \\endgroup
6405 \\def\\bbl@activateposthyphen{%
6406     \\let\\bbl@activateposthyphen\\relax
6407     \\directlua{
6408         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6409     }}
6410 \\def\\bbl@activateprehyphen{%
6411     \\let\\bbl@activateprehyphen\\relax
6412     \\directlua{
6413         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6414     }}

```

13.9 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.

```

6415 \bbl@trace{Redefinitions for bidi layout}
6416 \ifx\@eqnnum\@undefined\else
6417   \ifx\bbl@attr@dir\@undefined\else
6418     \edef\@eqnnum{%
6419       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6420       \unexpanded\expandafter{\@eqnnum}}
6421   \fi
6422 \fi
6423 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
6424 \ifnum\bbl@bidimode>\z@
6425   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6426     \bbl@exp{%
6427       \mathdir\the\bodydir
6428       #1%           Once entered in math, set boxes to restore values
6429       \<ifmmode>%
6430         \everyvbox{%
6431           \the\everyvbox
6432           \bodydir\the\bodydir
6433           \mathdir\the\mathdir
6434           \everyhbox{\the\everyhbox}%
6435           \everyvbox{\the\everyvbox}}%
6436         \everyhbox{%
6437           \the\everyhbox
6438           \bodydir\the\bodydir
6439           \mathdir\the\mathdir
6440           \everyhbox{\the\everyhbox}%
6441           \everyvbox{\the\everyvbox}}%
6442       \<fi>}}%
6443   \def\@hangfrom#1{%
6444     \setbox\@tempboxa\hbox{#1}%
6445     \hangindent\wd\@tempboxa
6446     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6447       \shapemode\@ne
6448     \fi
6449     \noindent\box\@tempboxa}
6450 \fi
6451 \IfBabelLayout{tabular}
6452   {\let\bbl@OL@tabular\@tabular
6453     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6454     \let\bbl@NL@tabular\@tabular
6455     \AtBeginDocument{%
6456       \ifx\bbl@NL@tabular\@tabular\else
6457         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6458         \let\bbl@NL@tabular\@tabular
6459       \fi}}
6460   {}
6461 \IfBabelLayout{lists}
6462   {\let\bbl@OL@list\list
6463     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6464     \let\bbl@NL@list\list
6465     \def\bbl@listparshape#1#2#3{%
6466       \parshape #1 #2 #3 %
6467       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6468         \shapemode\tw@
6469       \fi}}
6470   {}
6471 \IfBabelLayout{graphics}
6472   {\let\bbl@pictresetdir\relax
6473     \def\bbl@pictsetdir#1{%

```



```

6474 \ifcase\bbl@thetextdir
6475 \let\bbl@pictresetdir\relax
6476 \else
6477 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6478 \or\textdir TLT
6479 \else\bodydir TLT \textdir TLT
6480 \fi
6481 % \(\text|par)dir required in pgf:
6482 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6483 \fi}%
6484 \ifx\AddToHook\@undefined\else
6485 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6486 \directlua{
6487 Babel.get_picture_dir = true
6488 Babel.picture_has_bidi = 0
6489 function Babel.picture_dir (head)
6490 if not Babel.get_picture_dir then return head end
6491 for item in node.traverse(head) do
6492 if item.id == node.id'glyph' then
6493 local itemchar = item.char
6494 % TODO. Copypaste pattern from Babel.bidi (-r)
6495 local chardata = Babel.characters[itemchar]
6496 local dir = chardata and chardata.d or nil
6497 if not dir then
6498 for nn, et in ipairs(Babel.ranges) do
6499 if itemchar < et[1] then
6500 break
6501 elseif itemchar <= et[2] then
6502 dir = et[3]
6503 break
6504 end
6505 end
6506 end
6507 if dir and (dir == 'al' or dir == 'r') then
6508 Babel.picture_has_bidi = 1
6509 end
6510 end
6511 end
6512 return head
6513 end
6514 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6515 "Babel.picture_dir")
6516 }%
6517 \AtBeginDocument{%
6518 \long\def\put(#1,#2)#3{%
6519 \@killglue
6520 % Try:
6521 \ifx\bbl@pictresetdir\relax
6522 \def\bbl@tempc{0}%
6523 \else
6524 \directlua{
6525 Babel.get_picture_dir = true
6526 Babel.picture_has_bidi = 0
6527 }%
6528 \setbox\z@\hb@xt@\z@{%
6529 \@defaultunitsset\@tempdimc{#1}\unitlength
6530 \kern\@tempdimc
6531 #3\hss}%
6532 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%

```

```

6533 \fi
6534 % Do:
6535 \@defaultunitsset\@tempdimc{#2}\unitlength
6536 \raise\@tempdimc\hb@xt@\z@{%
6537 \@defaultunitsset\@tempdimc{#1}\unitlength
6538 \kern\@tempdimc
6539 {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6540 \ignorespaces}%
6541 \MakeRobust\put}%
6542 \fi
6543 \AtBeginDocument
6544 {\ifx\tikz@atbegin@node\@undefined\else
6545 \ifx\AddToHook\@undefined\else % TODO. Still tentative.
6546 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6547 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6548 \fi
6549 \let\bbbl@OL@pgfpicture\pgfpicture
6550 \bbbl@sreplace\pgfpicture{\pgfpicturetrue}%
6551 {\bbbl@pictsetdir\z@\pgfpicturetrue}%
6552 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6553 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6554 \bbbl@sreplace\tikz{\beginpgroup}%
6555 {\beginpgroup\bbbl@pictsetdir\tw@}%
6556 \fi
6557 \ifx\AddToHook\@undefined\else
6558 \AddToHook{env/tcolorbox/begin}{\bbbl@pictsetdir\@ne}%
6559 \fi
6560 }}
6561 {}

```

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.

```

6562 \IfBabelLayout{counters}%
6563 {\let\bbbl@OL@@textsuperscript\textsuperscript
6564 \bbbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6565 \let\bbbl@latinarabic=\@arabic
6566 \let\bbbl@OL@@arabic\@arabic
6567 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
6568 \@ifpackagewith{babel}{bidi=default}%
6569 {\let\bbbl@asciroman=\@roman
6570 \let\bbbl@OL@@roman\@roman
6571 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
6572 \let\bbbl@asciiRoman=\@Roman
6573 \let\bbbl@OL@@roman\@Roman
6574 \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
6575 \let\bbbl@OL@labelenumii\labelenumii
6576 \def\labelenumii{}\theenumii}%
6577 \let\bbbl@OL@p@enumiii\p@enumiii
6578 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
6579 <<Footnote changes>>
6580 \IfBabelLayout{footnotes}%
6581 {\let\bbbl@OL@footnote\footnote
6582 \BabelFootnote\footnote\language\@language}%
6583 \BabelFootnote\localfootnote\language\@language}%
6584 \BabelFootnote\mainfootnote\@mainfootnote}%
6585 {}

```

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.

```
6586 \IfBabelLayout{extras}%
6587   {\let\bbl@OL@underline\underline
6588    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6589    \let\bbl@OL@LaTeX2e\LaTeX2e
6590    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6591      \if b\expandafter\@car\@series\@nil\boldmath\fi
6592      \babelsublr}%
6593      \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}}}}
6594   {}
6595 \end{luatex}
```

13.10 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (`<l>`, `<r>` or `<al>`).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```
6596 (*basic-r)
6597 Babel = Babel or {}
6598
6599 Babel.bidi_enabled = true
6600
6601 require('babel-data-bidi.lua')
6602
6603 local characters = Babel.characters
```

```

6604 local ranges = Babel.ranges
6605
6606 local DIR = node.id("dir")
6607
6608 local function dir_mark(head, from, to, outer)
6609   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6610   local d = node.new(DIR)
6611   d.dir = '+' .. dir
6612   node.insert_before(head, from, d)
6613   d = node.new(DIR)
6614   d.dir = '-' .. dir
6615   node.insert_after(head, to, d)
6616 end
6617
6618 function Babel.bidi(head, ispar)
6619   local first_n, last_n          -- first and last char with nums
6620   local last_es                  -- an auxiliary 'last' used with nums
6621   local first_d, last_d          -- first and last char in L/R block
6622   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):

```

6623   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6624   local strong_lr = (strong == 'l') and 'l' or 'r'
6625   local outer = strong
6626
6627   local new_dir = false
6628   local first_dir = false
6629   local inmath = false
6630
6631   local last_lr
6632
6633   local type_n = ''
6634
6635   for item in node.traverse(head) do
6636
6637     -- three cases: glyph, dir, otherwise
6638     if item.id == node.id'glyph'
6639       or (item.id == 7 and item.subtype == 2) then
6640
6641       local itemchar
6642       if item.id == 7 and item.subtype == 2 then
6643         itemchar = item.replace.char
6644       else
6645         itemchar = item.char
6646       end
6647       local chardata = characters[itemchar]
6648       dir = chardata and chardata.d or nil
6649       if not dir then
6650         for nn, et in ipairs(ranges) do
6651           if itemchar < et[1] then
6652             break
6653           elseif itemchar <= et[2] then
6654             dir = et[3]
6655             break
6656           end
6657         end
6658       end

```

```

6659     dir = dir or 'l'
6660     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).

```

6661     if new_dir then
6662         attr_dir = 0
6663         for at in node.traverse(item.attr) do
6664             if at.number == luatexbase.registernumber'bbl@attr@dir' then
6665                 attr_dir = at.value % 3
6666             end
6667         end
6668         if attr_dir == 1 then
6669             strong = 'r'
6670         elseif attr_dir == 2 then
6671             strong = 'al'
6672         else
6673             strong = 'l'
6674         end
6675         strong_lr = (strong == 'l') and 'l' or 'r'
6676         outer = strong_lr
6677         new_dir = false
6678     end
6679
6680     if dir == 'nsm' then dir = strong end -- W1

```

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

```

6681     dir_real = dir -- We need dir_real to set strong below
6682     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:

```

6683     if strong == 'al' then
6684         if dir == 'en' then dir = 'an' end -- W2
6685         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6686         strong_lr = 'r' -- W3
6687     end

```

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

```

6688     elseif item.id == node.id'dir' and not inmath then
6689         new_dir = true
6690         dir = nil
6691     elseif item.id == node.id'math' then
6692         inmath = (item.subtype == 0)
6693     else
6694         dir = nil -- Not a char
6695     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>.

```

6696     if dir == 'en' or dir == 'an' or dir == 'et' then
6697         if dir ~= 'et' then
6698             type_n = dir
6699         end

```

```

6700     first_n = first_n or item
6701     last_n = last_es or item
6702     last_es = nil
6703   elseif dir == 'es' and last_n then -- W3+W6
6704     last_es = item
6705   elseif dir == 'cs' then           -- it's right - do nothing
6706   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6707     if strong_lr == 'r' and type_n ~= '' then
6708       dir_mark(head, first_n, last_n, 'r')
6709     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6710       dir_mark(head, first_n, last_n, 'r')
6711       dir_mark(head, first_d, last_d, outer)
6712       first_d, last_d = nil, nil
6713     elseif strong_lr == 'l' and type_n ~= '' then
6714       last_d = last_n
6715     end
6716     type_n = ''
6717     first_n, last_n = nil, nil
6718   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:

```

6719   if dir == 'l' or dir == 'r' then
6720     if dir ~= outer then
6721       first_d = first_d or item
6722       last_d = item
6723     elseif first_d and dir ~= strong_lr then
6724       dir_mark(head, first_d, last_d, outer)
6725       first_d, last_d = nil, nil
6726     end
6727   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.

```

6728   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6729     item.char = characters[item.char] and
6730       characters[item.char].m or item.char
6731   elseif (dir or new_dir) and last_lr ~= item then
6732     local mir = outer .. strong_lr .. (dir or outer)
6733     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6734       for ch in node.traverse(node.next(last_lr)) do
6735         if ch == item then break end
6736         if ch.id == node.id'glyph' and characters[ch.char] then
6737           ch.char = characters[ch.char].m or ch.char
6738         end
6739       end
6740     end
6741   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).

```

6742   if dir == 'l' or dir == 'r' then
6743     last_lr = item
6744     strong = dir_real           -- Don't search back - best save now

```

```

6745     strong_lr = (strong == 'l') and 'l' or 'r'
6746     elseif new_dir then
6747         last_lr = nil
6748     end
6749 end

```

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

```

6750 if last_lr and outer == 'r' then
6751     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6752         if characters[ch.char] then
6753             ch.char = characters[ch.char].m or ch.char
6754         end
6755     end
6756 end
6757 if first_n then
6758     dir_mark(head, first_n, last_n, outer)
6759 end
6760 if first_d then
6761     dir_mark(head, first_d, last_d, outer)
6762 end

```

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

```

6763 return node.prev(head) or head
6764 end
6765 </basic-r>

```

And here the Lua code for bidi=basic:

```

6766 <(*basic)
6767 Babel = Babel or {}
6768
6769 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6770
6771 Babel.fontmap = Babel.fontmap or {}
6772 Babel.fontmap[0] = {}      -- l
6773 Babel.fontmap[1] = {}      -- r
6774 Babel.fontmap[2] = {}      -- al/an
6775
6776 Babel.bidi_enabled = true
6777 Babel.mirroring_enabled = true
6778
6779 require('babel-data-bidi.lua')
6780
6781 local characters = Babel.characters
6782 local ranges = Babel.ranges
6783
6784 local DIR = node.id('dir')
6785 local GLYPH = node.id('glyph')
6786
6787 local function insert_implicit(head, state, outer)
6788     local new_state = state
6789     if state.sim and state.eim and state.sim ~= state.eim then
6790         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6791         local d = node.new(DIR)
6792         d.dir = '+' .. dir
6793         node.insert_before(head, state.sim, d)
6794         local d = node.new(DIR)
6795         d.dir = '-' .. dir
6796         node.insert_after(head, state.eim, d)

```

```

6797 end
6798 new_state.sim, new_state.eim = nil, nil
6799 return head, new_state
6800 end
6801
6802 local function insert_numeric(head, state)
6803   local new
6804   local new_state = state
6805   if state.san and state.ean and state.san ~= state.ean then
6806     local d = node.new(DIR)
6807     d.dir = '+TLT'
6808     _, new = node.insert_before(head, state.san, d)
6809     if state.san == state.sim then state.sim = new end
6810     local d = node.new(DIR)
6811     d.dir = '-TLT'
6812     _, new = node.insert_after(head, state.ean, d)
6813     if state.ean == state.eim then state.eim = new end
6814   end
6815   new_state.san, new_state.ean = nil, nil
6816   return head, new_state
6817 end
6818
6819 -- TODO - \hbox with an explicit dir can lead to wrong results
6820 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6821 -- was made to improve the situation, but the problem is the 3-dir
6822 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6823 -- well.
6824
6825 function Babel.bidi(head, ispar, hdir)
6826   local d -- d is used mainly for computations in a loop
6827   local prev_d = ''
6828   local new_d = false
6829
6830   local nodes = {}
6831   local outer_first = nil
6832   local inmath = false
6833
6834   local glue_d = nil
6835   local glue_i = nil
6836
6837   local has_en = false
6838   local first_et = nil
6839
6840   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6841
6842   local save_outer
6843   local temp = node.get_attribute(head, ATDIR)
6844   if temp then
6845     temp = temp % 3
6846     save_outer = (temp == 0 and 'l') or
6847                 (temp == 1 and 'r') or
6848                 (temp == 2 and 'al')
6849   elseif ispar then -- Or error? Shouldn't happen
6850     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6851   else -- Or error? Shouldn't happen
6852     save_outer = ('TRT' == hdir) and 'r' or 'l'
6853   end
6854   -- when the callback is called, we are just _after_ the box,
6855   -- and the textdir is that of the surrounding text

```



```

6856 -- if not ispar and hdir ~= tex.textdir then
6857 --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6858 -- end
6859 local outer = save_outer
6860 local last = outer
6861 -- 'al' is only taken into account in the first, current loop
6862 if save_outer == 'al' then save_outer = 'r' end
6863
6864 local fontmap = Babel.fontmap
6865
6866 for item in node.traverse(head) do
6867
6868   -- In what follows, #node is the last (previous) node, because the
6869   -- current one is not added until we start processing the neutrals.
6870
6871   -- three cases: glyph, dir, otherwise
6872   if item.id == GLYPH
6873     or (item.id == 7 and item.subtype == 2) then
6874
6875     local d_font = nil
6876     local item_r
6877     if item.id == 7 and item.subtype == 2 then
6878       item_r = item.replace -- automatic discs have just 1 glyph
6879     else
6880       item_r = item
6881     end
6882     local chardata = characters[item_r.char]
6883     d = chardata and chardata.d or nil
6884     if not d or d == 'nsm' then
6885       for nn, et in ipairs(ranges) do
6886         if item_r.char < et[1] then
6887           break
6888         elseif item_r.char <= et[2] then
6889           if not d then d = et[3]
6890           elseif d == 'nsm' then d_font = et[3]
6891           end
6892           break
6893         end
6894       end
6895     end
6896     d = d or 'l'
6897
6898     -- A short 'pause' in bidi for mapfont
6899     d_font = d_font or d
6900     d_font = (d_font == 'l' and 0) or
6901              (d_font == 'nsm' and 0) or
6902              (d_font == 'r' and 1) or
6903              (d_font == 'al' and 2) or
6904              (d_font == 'an' and 2) or nil
6905     if d_font and fontmap and fontmap[d_font][item_r.font] then
6906       item_r.font = fontmap[d_font][item_r.font]
6907     end
6908
6909     if new_d then
6910       table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6911       if inmath then
6912         attr_d = 0
6913       else
6914         attr_d = node.get_attribute(item, ATDIR)

```

```

6915         attr_d = attr_d % 3
6916     end
6917     if attr_d == 1 then
6918         outer_first = 'r'
6919         last = 'r'
6920     elseif attr_d == 2 then
6921         outer_first = 'r'
6922         last = 'al'
6923     else
6924         outer_first = 'l'
6925         last = 'l'
6926     end
6927     outer = last
6928     has_en = false
6929     first_et = nil
6930     new_d = false
6931 end
6932
6933 if glue_d then
6934     if (d == 'l' and 'l' or 'r') ~= glue_d then
6935         table.insert(nodes, {glue_i, 'on', nil})
6936     end
6937     glue_d = nil
6938     glue_i = nil
6939 end
6940
6941 elseif item.id == DIR then
6942     d = nil
6943     new_d = true
6944
6945 elseif item.id == node.id'glue' and item.subtype == 13 then
6946     glue_d = d
6947     glue_i = item
6948     d = nil
6949
6950 elseif item.id == node.id'math' then
6951     inmath = (item.subtype == 0)
6952
6953 else
6954     d = nil
6955 end
6956
6957 -- AL <= EN/ET/ES      -- W2 + W3 + W6
6958 if last == 'al' and d == 'en' then
6959     d = 'an'          -- W3
6960 elseif last == 'al' and (d == 'et' or d == 'es') then
6961     d = 'on'          -- W6
6962 end
6963
6964 -- EN + CS/ES + EN      -- W4
6965 if d == 'en' and #nodes >= 2 then
6966     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6967         and nodes[#nodes-1][2] == 'en' then
6968         nodes[#nodes][2] = 'en'
6969     end
6970 end
6971
6972 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6973 if d == 'an' and #nodes >= 2 then

```

```

6974     if (nodes[#nodes][2] == 'cs')
6975         and nodes[#nodes-1][2] == 'an' then
6976             nodes[#nodes][2] = 'an'
6977         end
6978     end
6979
6980     -- ET/EN          -- W5 + W7->1 / W6->on
6981     if d == 'et' then
6982         first_et = first_et or (#nodes + 1)
6983     elseif d == 'en' then
6984         has_en = true
6985         first_et = first_et or (#nodes + 1)
6986     elseif first_et then      -- d may be nil here !
6987         if has_en then
6988             if last == 'l' then
6989                 temp = 'l'      -- W7
6990             else
6991                 temp = 'en'     -- W5
6992             end
6993         else
6994             temp = 'on'        -- W6
6995         end
6996         for e = first_et, #nodes do
6997             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6998         end
6999         first_et = nil
7000         has_en = false
7001     end
7002
7003     -- Force mathdir in math if ON (currently works as expected only
7004     -- with 'l')
7005     if inmath and d == 'on' then
7006         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7007     end
7008
7009     if d then
7010         if d == 'al' then
7011             d = 'r'
7012             last = 'al'
7013         elseif d == 'l' or d == 'r' then
7014             last = d
7015         end
7016         prev_d = d
7017         table.insert(nodes, {item, d, outer_first})
7018     end
7019
7020     outer_first = nil
7021
7022 end
7023
7024 -- TODO -- repeated here in case EN/ET is the last node. Find a
7025 -- better way of doing things:
7026 if first_et then      -- dir may be nil here !
7027     if has_en then
7028         if last == 'l' then
7029             temp = 'l'      -- W7
7030         else
7031             temp = 'en'     -- W5
7032         end

```

```

7033     else
7034         temp = 'on'      -- W6
7035     end
7036     for e = first_et, #nodes do
7037         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7038     end
7039 end
7040
7041 -- dummy node, to close things
7042 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7043
7044 ----- NEUTRAL -----
7045
7046 outer = save_outer
7047 last = outer
7048
7049 local first_on = nil
7050
7051 for q = 1, #nodes do
7052     local item
7053
7054     local outer_first = nodes[q][3]
7055     outer = outer_first or outer
7056     last = outer_first or last
7057
7058     local d = nodes[q][2]
7059     if d == 'an' or d == 'en' then d = 'r' end
7060     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7061
7062     if d == 'on' then
7063         first_on = first_on or q
7064     elseif first_on then
7065         if last == d then
7066             temp = d
7067         else
7068             temp = outer
7069         end
7070         for r = first_on, q - 1 do
7071             nodes[r][2] = temp
7072             item = nodes[r][1]      -- MIRRORING
7073             if Babel.mirroring_enabled and item.id == GLYPH
7074                 and temp == 'r' and characters[item.char] then
7075                 local font_mode = font.fonts[item.font].properties.mode
7076                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7077                     item.char = characters[item.char].m or item.char
7078                 end
7079             end
7080         end
7081         first_on = nil
7082     end
7083
7084     if d == 'r' or d == 'l' then last = d end
7085 end
7086
7087 ----- IMPLICIT, REORDER -----
7088
7089 outer = save_outer
7090 last = outer
7091

```

```

7092 local state = {}
7093 state.has_r = false
7094
7095 for q = 1, #nodes do
7096     local item = nodes[q][1]
7097
7098     outer = nodes[q][3] or outer
7099
7100     local d = nodes[q][2]
7101
7102     if d == 'nsm' then d = last end          -- W1
7103     if d == 'en' then d = 'an' end
7104     local isdir = (d == 'r' or d == 'l')
7105
7106     if outer == 'l' and d == 'an' then
7107         state.san = state.san or item
7108         state.ean = item
7109     elseif state.san then
7110         head, state = insert_numeric(head, state)
7111     end
7112
7113     if outer == 'l' then
7114         if d == 'an' or d == 'r' then      -- im -> implicit
7115             if d == 'r' then state.has_r = true end
7116             state.sim = state.sim or item
7117             state.eim = item
7118         elseif d == 'l' and state.sim and state.has_r then
7119             head, state = insert_implicit(head, state, outer)
7120         elseif d == 'l' then
7121             state.sim, state.eim, state.has_r = nil, nil, false
7122         end
7123     else
7124         if d == 'an' or d == 'l' then
7125             if nodes[q][3] then -- nil except after an explicit dir
7126                 state.sim = item -- so we move sim 'inside' the group
7127             else
7128                 state.sim = state.sim or item
7129             end
7130             state.eim = item
7131         elseif d == 'r' and state.sim then
7132             head, state = insert_implicit(head, state, outer)
7133         elseif d == 'r' then
7134             state.sim, state.eim = nil, nil
7135         end
7136     end
7137 end
7138
7139 if isdir then
7140     last = d          -- Don't search back - best save now
7141 elseif d == 'on' and state.san then
7142     state.san = state.san or item
7143     state.ean = item
7144 end
7145
7146 end
7147
7148 return node.prev(head) or head
7149 end
7150 </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.

```
7151 <*nil>
7152 \ProvidesLanguage{nil}[<<date>> <<version>> Nil language]
7153 \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.

```
7154 \ifx\l@nil\undefined
7155   \newlanguage\l@nil
7156   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7157   \let\bbl@elt\relax
7158   \edef\bbl@languages{% Add it to the list of languages
7159     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7160 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
7161 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7162 \let\captionnil\@empty
7163 \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.

```
7164 \ldf@finish{nil}
7165 </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`.

```
7166 <(*bplain | blplain)>
7167 \catcode`\{=1 % left brace is begin-group character
7168 \catcode`\}=2 % right brace is end-group character
7169 \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).

```
7170 \openin 0 hyphen.cfg
7171 \ifeof0
7172 \else
7173   \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.

```
7174   \def\input #1 {%
7175     \let\input\input
7176     \a hyphen.cfg
7177     \let\input\undefined
7178   }
7179 \fi
7180 </bplain | blplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
7181 <bplain>\a plain.tex
7182 <blplain>\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
7183 <bplain>\def\fmtname{babel-plain}
7184 <blplain>\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of \LaTeX 2_ε that are needed for `babel`.

```
7185 <<(*Emulate LaTeX)>> ≡
7186 % == Code for plain ==
7187 \def\@empty{}
7188 \def\loadlocalcfg#1{%
7189   \openin0#1.cfg
7190   \ifeof0
7191     \closein0
7192   \else
7193     \closein0
```

```

7194    {\immediate\write16{*****}%
7195    \immediate\write16{* Local config file #1.cfg used}%
7196    \immediate\write16{*}%
7197    }
7198    \input #1.cfg\relax
7199    \fi
7200    \@endofldef}

```

16.3 General tools

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

```

7201 \long\def\@firstofone#1{#1}
7202 \long\def\@firstoftwo#1#2{#1}
7203 \long\def\@secondoftwo#1#2{#2}
7204 \def\@nnil{\@nil}
7205 \def\@gobbletwo#1#2{}
7206 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7207 \def\@star@or@long#1{%
7208   \@ifstar
7209   {\let\l@ngrel@x\relax#1}%
7210   {\let\l@ngrel@x\long#1}}
7211 \let\l@ngrel@x\relax
7212 \def\@car#1#2\@nil{#1}
7213 \def\@cdr#1#2\@nil{#2}
7214 \let\@typeset@protect\relax
7215 \let\protected@edef\edef
7216 \long\def\@gobble#1{}
7217 \edef\@backslashchar{\expandafter\@gobble\string\}
7218 \def\strip@prefix#1>{}
7219 \def\g@addto@macro#1#2{%
7220   \toks@\expandafter{#1#2}%
7221   \xdef#1{\the\toks@}}
7222 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7223 \def\@nameuse#1{\csname #1\endcsname}
7224 \def\@ifundefined#1{%
7225   \expandafter\ifx\csname#1\endcsname\relax
7226     \expandafter\@firstoftwo
7227   \else
7228     \expandafter\@secondoftwo
7229   \fi}
7230 \def\@expandtwoargs#1#2#3{%
7231   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7232 \def\zap@space#1 #2{%
7233   #1%
7234   \ifx#2\@empty\else\expandafter\zap@space\fi
7235   #2}
7236 \let\bbl@trace\@gobble

```

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

```

7237 \ifx\@preamblecmds\undefined
7238   \def\@preamblecmds{}
7239 \fi
7240 \def\@onlypreamble#1{%
7241   \expandafter\gdef\expandafter\@preamblecmds\expandafter{
7242     \@preamblecmds\do#1}}
7243 \@onlypreamble\@onlypreamble

```

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


```

7244 \def\begindocument{%
7245   \@begindocumenthook
7246   \global\let\@begindocumenthook\@undefined
7247   \def\do##1{\global\let##1\@undefined}%
7248   \@preamblecmds
7249   \global\let\do\noexpand}

7250 \ifx\@begindocumenthook\@undefined
7251   \def\@begindocumenthook{}
7252 \fi
7253 \@onlypreamble\@begindocumenthook
7254 \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`.

```

7255 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7256 \@onlypreamble\AtEndOfPackage
7257 \def\@endofldf{}
7258 \@onlypreamble\@endofldf
7259 \let\bbl@afterlang\@empty
7260 \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.

```

7261 \catcode`\&=\z@
7262 \ifx&\if@files\@undefined
7263   \expandafter\let\csname if@files\expandafter\endcsname
7264     \csname iffalse\endcsname
7265 \fi
7266 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7267 \def\newcommand{\@star@or@long\new@command}
7268 \def\new@command#1{%
7269   \@testopt{\@newcommand#1}0}
7270 \def\@newcommand#1[#2]{%
7271   \@ifnextchar [{\@xargdef#1[#2]}%
7272     {\@argdef#1[#2]}}
7273 \long\def\@argdef#1[#2]#3{%
7274   \@yargdef#1\@ne{#2}{#3}}
7275 \long\def\@xargdef#1[#2][#3]#4{%
7276   \expandafter\def\expandafter#1\expandafter{%
7277     \expandafter\@protected@testopt\expandafter #1%
7278     \csname\string#1\expandafter\endcsname{#3}}%
7279   \expandafter\@yargdef \csname\string#1\endcsname
7280   \tw@{#2}{#4}}
7281 \long\def\@yargdef#1#2#3{%
7282   \@tempcnta#3\relax
7283   \advance \@tempcnta \@ne
7284   \let\@hash@\relax
7285   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7286   \@tempcntb #2%
7287   \@whilenum\@tempcntb <\@tempcnta
7288   \do{%
7289     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7290     \advance\@tempcntb \@ne}%
7291   \let\@hash@###
7292   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
7293 \def\providecommand{\@star@or@long\provide@command}

```

```

7294 \def\provide@command#1{%
7295   \begingroup
7296     \escapechar\m@ne\xdef\@gtempa{\string#1}%
7297   \endgroup
7298   \expandafter\@ifundefined\@gtempa
7299     {\def\reserved@a{\new@command#1}}%
7300     {\let\reserved@a\relax
7301      \def\reserved@a{\new@command\reserved@a}}%
7302   \reserved@a}%

7303 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7304 \def\declare@robustcommand#1{%
7305   \edef\reserved@a{\string#1}%
7306   \def\reserved@b{#1}%
7307   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7308   \edef#1{%
7309     \ifx\reserved@a\reserved@b
7310       \noexpand\x@protect
7311       \noexpand#1%
7312     \fi
7313     \noexpand\protect
7314     \expandafter\noexpand\csname
7315       \expandafter\@gobble\string#1 \endcsname
7316   }%
7317   \expandafter\new@command\csname
7318     \expandafter\@gobble\string#1 \endcsname
7319 }
7320 \def\x@protect#1{%
7321   \ifx\protect\@typeset@protect\else
7322     \@x@protect#1%
7323   \fi
7324 }
7325 \catcode`\&=\z@ % Trick to hide conditionals
7326 \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`.

```

7327 \def\bbl@tempa{\csname newif\endcsname&fin@}
7328 \catcode`\&=4
7329 \ifx\in@\@undefined
7330   \def\in@#1#2{%
7331     \def\in@##1#1##2##3\in@{%
7332       \ifx\in@##2\in@false\else\in@true\fi}%
7333     \in@##2#1\in@\in@}
7334 \else
7335   \let\bbl@tempa\@empty
7336 \fi
7337 \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).

```

7338 \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.

```

7339 \def\@ifl@aded#1#2#3#4{}

```

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

```
7340 \ifx\@tempcnta\@undefined
7341   \csname newcount\endcsname\@tempcnta\relax
7342 \fi
7343 \ifx\@tempcntb\@undefined
7344   \csname newcount\endcsname\@tempcntb\relax
7345 \fi
```

To prevent wasting two counters in \LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
7346 \ifx\bye\@undefined
7347   \advance\count10 by -2\relax
7348 \fi
7349 \ifx\@ifnextchar\@undefined
7350   \def\@ifnextchar#1#2#3{%
7351     \let\reserved@d=#1%
7352     \def\reserved@a{#2}\def\reserved@b{#3}%
7353     \futurelet\@let@token\@ifnch}
7354   \def\@ifnch{%
7355     \ifx\@let@token\@sptoken
7356       \let\reserved@c\@xifnch
7357     \else
7358       \ifx\@let@token\reserved@d
7359         \let\reserved@c\reserved@a
7360       \else
7361         \let\reserved@c\reserved@b
7362       \fi
7363     \fi
7364     \reserved@c}
7365   \def\:\let\@sptoken= } \: % this makes \@sptoken a space token
7366   \def\:\@xifnch \expandafter\def\: {\futurelet\@let@token\@ifnch}
7367 \fi
7368 \def\@testopt#1#2{%
7369   \@ifnextchar[#{#1}{#1[#2]}}
7370 \def\@protected@testopt#1{%
7371   \ifx\protect\@typeset@protect
7372     \expandafter\@testopt
7373   \else
7374     \@x@protect#1%
7375   \fi}
7376 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7377   #2\relax}\fi}
7378 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7379   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```
7380 \def\DeclareTextCommand{%
7381   \@dec@text@cmd\providecommand
7382 }
7383 \def\ProvideTextCommand{%
7384   \@dec@text@cmd\providecommand
7385 }
7386 \def\DeclareTextSymbol#1#2#3{%
7387   \@dec@text@cmd\chardef#1{#2}#3\relax
```

```

7388 }
7389 \def\@dec@text@cmd#1#2#3{%
7390   \expandafter\def\expandafter#2%
7391     \expandafter{%
7392       \csname#3-cmd\expandafter\endcsname
7393       \expandafter#2%
7394       \csname#3\string#2\endcsname
7395     }%
7396 %   \let\@ifdefinable\@rc@ifdefinable
7397   \expandafter#1\csname#3\string#2\endcsname
7398 }
7399 \def\@current@cmd#1{%
7400   \ifx\protect\@typeset@protect\else
7401     \noexpand#1\expandafter\@gobble
7402   \fi
7403 }
7404 \def\@changed@cmd#1#2{%
7405   \ifx\protect\@typeset@protect
7406     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7407       \expandafter\ifx\csname ?\string#1\endcsname\relax
7408         \expandafter\def\csname ?\string#1\endcsname{%
7409           \@changed@x@err{#1}%
7410         }%
7411       \fi
7412       \global\expandafter\let
7413         \csname\cf@encoding \string#1\expandafter\endcsname
7414         \csname ?\string#1\endcsname
7415     \fi
7416     \csname\cf@encoding\string#1%
7417     \expandafter\endcsname
7418   \else
7419     \noexpand#1%
7420   \fi
7421 }
7422 \def\@changed@x@err#1{%
7423   \errhelp{Your command will be ignored, type <return> to proceed}%
7424   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7425 \def\DeclareTextCommandDefault#1{%
7426   \DeclareTextCommand#1?%
7427 }
7428 \def\ProvideTextCommandDefault#1{%
7429   \ProvideTextCommand#1?%
7430 }
7431 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7432 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7433 \def\DeclareTextAccent#1#2#3{%
7434   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7435 }
7436 \def\DeclareTextCompositeCommand#1#2#3#4{%
7437   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7438   \edef\reserved@b{\string##1}%
7439   \edef\reserved@c{%
7440     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7441   \ifx\reserved@b\reserved@c
7442     \expandafter\expandafter\expandafter\ifx
7443       \expandafter\@car\reserved@a\relax\relax\@nil
7444       \@text@composite
7445     \else
7446       \edef\reserved@b##1{%

```

```

7447         \def\expandafter\noexpand
7448             \csname#2\string#1\endcsname###1{%
7449             \noexpand\@text@composite
7450             \expandafter\noexpand\csname#2\string#1\endcsname
7451             ###1\noexpand\@empty\noexpand\@text@composite
7452             {##1}%
7453         }%
7454     }%
7455     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7456 \fi
7457 \expandafter\def\csname\expandafter\string\csname
7458     #2\endcsname\string#1-\string#3\endcsname{##4}
7459 \else
7460     \errhelp{Your command will be ignored, type <return> to proceed}%
7461     \errmessage{\string\DeclareTextCompositeCommand\space used on
7462         inappropriate command \protect#1}
7463 \fi
7464 }
7465 \def\@text@composite#1#2#3\@text@composite{%
7466     \expandafter\@text@composite@x
7467         \csname\string#1-\string#2\endcsname
7468 }
7469 \def\@text@composite@x#1#2{%
7470     \ifx#1\relax
7471         #2%
7472     \else
7473         #1%
7474     \fi
7475 }
7476 %
7477 \def\@strip@args#1:#2-#3\@strip@args{#2}
7478 \def\DeclareTextComposite#1#2#3#4{%
7479     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7480     \bgroup
7481         \lcode`\@=#4%
7482         \lowercase{%
7483     \egroup
7484         \reserved@a \@%
7485     }%
7486 }
7487 %
7488 \def\UseTextSymbol#1#2{#2}
7489 \def\UseTextAccent#1#2#3{}
7490 \def\@use@text@encoding#1{}
7491 \def\DeclareTextSymbolDefault#1#2{%
7492     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7493 }
7494 \def\DeclareTextAccentDefault#1#2{%
7495     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7496 }
7497 \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.

7498 \DeclareTextAccent{"}{OT1}{127}
7499 \DeclareTextAccent{'}{OT1}{19}
7500 \DeclareTextAccent{^}{OT1}{94}
7501 \DeclareTextAccent`}{OT1}{18}
7502 \DeclareTextAccent{~}{OT1}{126}

```

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

```
7503 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7504 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
7505 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
7506 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7507 \DeclareTextSymbol{\i}{OT1}{16}
7508 \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`.

```
7509 \ifx\scriptsize\undefined
7510   \let\scriptsize\sevenrm
7511 \fi
7512 % End of code for plain
7513 <</Emulate LaTeX>>
```

A proxy file:

```
7514 <plain>
7515 \input babel.def
7516 </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 L^AT_EX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The T_EXbook*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *L^AT_EX, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: *T_EXhax Digest*, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German T_EX*, *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.
- [12] K.F. Treebus, *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).