

Babel

Version 3.61.2424
2021/07/05

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	8
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	16
1.12	The base option	18
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	33
1.18	Dates	34
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Transforms	38
1.22	Selection based on BCP 47 tags	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	49
1.31	Current and future work	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	80
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	159
12	Font handling with fontspec	163

13	Hooks for XeTeX and LuaTeX	168
13.1	XeTeX	168
13.2	Layout	170
13.3	LuaTeX	171
13.4	Southeast Asian scripts	177
13.5	CJK line breaking	179
13.6	Arabic justification	181
13.7	Common stuff	185
13.8	Automatic fonts and ids switching	185
13.9	Layout	199
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)	214
16.1	Not renaming hyphen.tex	214
16.2	Emulating some L _A T _E X features	215
16.3	General tools	215
16.4	Encoding related macros	219
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 xet_{ex} and lua_{tex}, the following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and `\today` in Danish and Vietnamese. No additional packages are required.

LUATEX/XETEX

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

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

1.3 Mostly monolingual documents

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

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

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

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

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

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

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

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

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

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

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

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

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

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

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

1.9 More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

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

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

⁶Thanks to Enrico Gregorio

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

1.11 Package options

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

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

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

activegrave Same for `.

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

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

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

safe= none | ref | bib

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

math= active | normal

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

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

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

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

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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

1.12 The base option

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

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

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

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

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 $\text{T}_{\text{E}}\text{X}$ 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 work as expected.

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

LUATEX/XETEX

```
\documentclass{book}

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

```

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

\begin{document}

\tableofcontents

\chapter{სამხარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამხარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

```

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

```

Or also:

```

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

```

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

```

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

```

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

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

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

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

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

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1໐ 1໙ 1໑ 1໘ 1໗} % 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}	bg	Bulgarian ^{ul}
agq	Aghem	bm	Bambara
ak	Akan	bn	Bangla ^{ul}
am	Amharic ^{ul}	bo	Tibetan ^u
ar	Arabic ^{ul}	brx	Bodo
ar-DZ	Arabic ^{ul}	bs-Cyrl	Bosnian
ar-MA	Arabic ^{ul}	bs-Latn	Bosnian ^{ul}
ar-SY	Arabic ^{ul}	bs	Bosnian ^{ul}
as	Assamese	ca	Catalan ^{ul}
asa	Asu	ce	Chechen
ast	Asturian ^{ul}	cgg	Chiga
az-Cyrl	Azerbaijani	chr	Cherokee
az-Latn	Azerbaijani	ckb	Central Kurdish
az	Azerbaijani ^{ul}	cop	Coptic
bas	Basaa	cs	Czech ^{ul}
be	Belarusian ^{ul}	cu	Church Slavic
bem	Bemba	cu-Cyrs	Church Slavic
bez	Bena	cu-Glag	Church Slavic

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

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

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

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

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

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

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

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

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

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

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

Modifying and adding values to ini files

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

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

This is *not* 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 import'ed from `ini` files, you can modify the values of specific keys, like:

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

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

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

```
\addto\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}
```

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

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

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads $([\text{t}\acute{o}])$, the replacement could be $\{1|\text{t}\acute{o}|\text{t}\acute{o}\}$, which maps $\text{t}\acute{}$ to t , and \acute{o} to $\text{t}\acute{o}$, 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 `rl` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

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

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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ parameters (`#1`, `#2`, `#3`), with the meaning given:

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.27 Languages supported by babel with ldf files

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

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

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

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

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

```

\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`), L^AT_EX 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, send it to me. Feel free to ask for help or to make feature requests.

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

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

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 \TeX to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

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

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

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

`\bbl@add@special` The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]
`\bbl@remove@special` It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

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

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

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

3.6 Support for extending macros

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

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

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

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

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

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

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=3.61.2424>>
2 <<date=2021/07/05>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8   {\def#1{#2}}%
9   {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@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\bbbl@trace[1]{\message{^^J[ #1 ]}}}%
206   \let\bbbl@debug\@firstofone
207   \ifx\directlua\@undefined\else
208     \directlua{ Babel = Babel or {}
209       Babel.debug = true }%
210   \fi}
211 {\providecommand\bbbl@trace[1]{}%
212   \let\bbbl@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\bbbl@error#1#2{%
220   \begingroup
221     \def\{\{\MessageBreak}%
222     \PackageError{babel}{#1}{#2}%
223   \endgroup}
224 \def\bbbl@warning#1{%
225   \begingroup
226     \def\{\{\MessageBreak}%
227     \PackageWarning{babel}{#1}%
228   \endgroup}
229 \def\bbbl@infowarn#1{%
230   \begingroup
231     \def\{\{\MessageBreak}%
232     \GenericWarning
233       {(babel) \@spaces\@spaces\@spaces}%
234       {Package babel Info: #1}%
235   \endgroup}
236 \def\bbbl@info#1{%
237   \begingroup
238     \def\{\{\MessageBreak}%
239     \PackageInfo{babel}{#1}%
240   \endgroup}
241 \def\bbbl@nocaption{\protect\bbbl@nocaption@i}
242 % TODO - Wrong for \today !!! Must be a separate macro.
243 \def\bbbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
244   \global\@namedef{#2}{\textbf{?#1?}}%
245   \@nameuse{#2}%
246   \edef\bbbl@tempa{#1}%
247   \bbbl@sreplace\bbbl@tempa{name}{}}%
248   \bbbl@warning{%
249     \@backslashchar#1 not set for '\language' name'. Please,\%
250     define it after the language has been loaded\%
251     (typically in the preamble) with\%
252     \string\setlocalecaption{\language name}{\bbbl@tempa}{..}\%
253     Reported}}
254 \def\bbbl@tentative{\protect\bbbl@tentative@i}
255 \def\bbbl@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 \fi
407 %

```

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=...`

```

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

```

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

```

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

```

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

```

429   \edef\bbbl@opt@shorthands{%
430     \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.

```

431   \bbbl@ifshorthand{'}%
432     {\PassOptionsToPackage{activeacute}{babel}}{}
433   \bbbl@ifshorthand{`}%
434     {\PassOptionsToPackage{activegrave}{babel}}{}
435 \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.

```

436 \ifx\bbbl@opt@headfoot\@nnil\else
437   \g@addto@macro\@resetactivechars{%
438     \set@typeset@protect
439     \expandafter\select@language@x\expandafter{\bbbl@opt@headfoot}%
440     \let\protect\noexpand}
441 \fi

```

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

```

442 \ifx\bbbl@opt@safe\undefined
443   \def\bbbl@opt@safe{BR}
444 \fi
445 \ifx\bbbl@opt@main\@nnil\else
446   \edef\bbbl@language@opts{%
447     \ifx\bbbl@language@opts\@empty\else\bbbl@language@opts,\fi
448     \bbbl@opt@main}
449 \fi

```

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

```

450 \bbbl@trace{Defining IfBabelLayout}
451 \ifx\bbbl@opt@layout\@nnil
452   \newcommand\IfBabelLayout[3]{#3}%
453 \else
454   \newcommand\IfBabelLayout[1]{%
455     \@expandtwoargs\in@{.#1.}{.\bbbl@opt@layout.}%
456     \ifin@
457       \expandafter\@firstoftwo
458     \else
459       \expandafter\@secondoftwo
460     \fi}
461 \fi

```

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

```
462 \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.

```

463 <<{*More package options}> \equiv
464 \DeclareOption{safe=none}{\let\bbbl@opt@safe\@empty}
465 \DeclareOption{safe=bib}{\def\bbbl@opt@safe{B}}
466 \DeclareOption{safe=ref}{\def\bbbl@opt@safe{R}}
467 <</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.

```

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

```
478 \CheckCommand*\@testdef[3]{%
479   \def\reserved@a{#3}%
480   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
481   \else
482     \@tempswatrue
483   \fi}
```

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

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

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

```
499 \bbl@xin@{R}\bbl@opt@safe
500 \ifin@
501   \bbl@redefineroobust\ref#1{%
502     \@safe@activetrue\org@ref{#1}\@safe@activetrue}
503   \bbl@redefineroobust\pageref#1{%
504     \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
505 \else
506   \let\org@ref\ref
507   \let\org@pageref\pageref
508 \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.

```
509 \bbl@xin@{B}\bbl@opt@safe
510 \ifin@
511   \bbl@redefine\@citex[#1]#2{%
512     \@safe@activetrue\edef\@tempa{#2}\@safe@activetrue}
513   \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.

```
514 \AtBeginDocument{%
515   \@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.)

```
516 \def\@citex[#1][#2]#3{%
517   \@safe@activetrue\edef\@tempa{#3}\@safe@activetruefalse
518   \org@@citex[#1][#2]{\@tempa}}%
519 }{}}
```

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

```
520 \AtBeginDocument{%
521   \@ifpackageloaded{cite}{%
522     \def\@citex[#1]#2{%
523       \@safe@activetrue\org@@citex[#1][#2]\@safe@activetruefalse}%
524     }{}}
```

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

```
525 \bbl@redefine\nocite#1{%
526   \@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.

```
527 \bbl@redefine\bibcite{%
528   \bbl@cite@choice
529   \bibcite}
```

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

```
530 \def\bbl@bibcite#1#2{%
531   \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.

```
532 \def\bbl@cite@choice{%
533   \global\let\bibcite\bbl@bibcite
534   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
535   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
536   \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.

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

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

```
538 \bbl@redefine\@bibitem#1{%
539   \@safe@activetrue\org@@bibitem{#1}\@safe@activetruefalse}
540 \else
541   \let\org@nocite\nocite
542   \let\org@@citex\@citex
543   \let\org@bibcite\bibcite
544   \let\org@@bibitem\@bibitem
545 \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.

```

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

```

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

`\@mkboth`

```

567   \ifx\@mkboth\markboth
568     \def\bbl@tempc{\let\@mkboth\markboth}
569   \else
570     \def\bbl@tempc{}
571   \fi
572   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
573   \markboth#1#2{%
574     \protected@edef\bbl@tempb##1{%
575       \protect\foreignlanguage
576       {\language}{\protect\bbl@restore@actives##1}%
577     \bbl@ifblank{#1}%
578     {\toks@{}}%
579     {\toks@\expandafter{\bbl@tempb{#1}}}%
580     \bbl@ifblank{#2}%
581     {\@temptokena{}}%
582     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
583     \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
584     \bbl@tempc
585   \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.

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

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`.

```
609 \AtBeginDocument{%
610   \@ifpackageloaded{varioref}{%
611     \bbl@redefine\@@vpageref#1[#2]#3{%
612       \@safe@activestrue
613       \org@@@vpageref{#1}{#2}{#3}%
614       \@safe@activesfalse}%
615     \bbl@redefine\vrefpagemum#1#2{%
616       \@safe@activestrue
617       \org@vrefpagemum{#1}{#2}%
618       \@safe@activesfalse}%
619   }
```

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.

```
619 \expandafter\def\csname Ref \endcsname#1{%
620 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
621 }{}%
622 }
623 \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.

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

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.

```
633 % \AtBeginDocument{%
634 % \ifx\pdfstringdefDisableCommands\@undefined\else
635 % \pdfstringdefDisableCommands{\languageshorthands{system}}%
636 % \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.

```
637 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
638 \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 provided by `LATEX`.

```
639 \def\substitutefontfamily#1#2#3{%
640 \lowercase{\immediate\openout15=#1#2.fd\relax}%
641 \immediate\write15{%
642 \string\ProvidesFile{#1#2.fd}%
643 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
644 \space generated font description file]^^J
```

```

645 \string\DeclareFontFamily{#1}{#2}{}^^J
646 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
647 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
648 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
649 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
650 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
651 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
652 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
653 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
654 }%
655 \closeout15
656 }
657 \@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`

```

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

```



```
693 \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.

```
694 \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.

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

```
719 \DeclareRobustCommand{\latintext}{%
720   \fontencoding{\latinencoding}\selectfont
721   \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.

```
722 \ifx\@undefined\DeclareTextFontCommand
723   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
724 \else
725   \DeclareTextFontCommand{\textlatin}{\latintext}
726 \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.

```

727 \ifodd\bbl@engine
728   \def\bbl@activate@preotf{%
729     \let\bbl@activate@preotf\relax % only once
730     \directlua{
731       Babel = Babel or {}
732       %
733       function Babel.pre_otfload_v(head)
734         if Babel.numbers and Babel.digits_mapped then
735           head = Babel.numbers(head)
736         end
737         if Babel.bidi_enabled then
738           head = Babel.bidi(head, false, dir)
739         end
740         return head
741       end
742       %
743       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
744         if Babel.numbers and Babel.digits_mapped then
745           head = Babel.numbers(head)
746         end
747         if Babel.bidi_enabled then
748           head = Babel.bidi(head, false, dir)
749         end
750         return head
751       end
752       %
753       luatexbase.add_to_callback('pre_linebreak_filter',
754         Babel.pre_otfload_v,
755         'Babel.pre_otfload_v',
756         luatexbase.priority_in_callback('pre_linebreak_filter',
757           'luaotfload.node_processor') or nil)
758       %
759       luatexbase.add_to_callback('hpack_filter',
760         Babel.pre_otfload_h,
761         'Babel.pre_otfload_h',
762         luatexbase.priority_in_callback('hpack_filter',
763           'luaotfload.node_processor') or nil)
764     }}
765 \fi

```

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

```

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

```

```

822 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
823 \fi
824 \AtEndOfPackage{%
825 \EnableBabelHook{babel-bidi}%
826 \ifodd\bbl@engine\else
827 \bbl@xebidipar
828 \fi}
829 \fi

```

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

```

830 \bbl@trace{Macros to switch the text direction}
831 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
832 \def\bbl@rscripts{% TODO. Base on codes ??
833 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
834 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
835 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
836 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
837 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
838 Old South Arabian,}%
839 \def\bbl@provide@dirs#1{%
840 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
841 \ifin@
842 \global\bbl@csarg\chardef{wdir@#1}\@ne
843 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
844 \ifin@
845 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
846 \fi
847 \else
848 \global\bbl@csarg\chardef{wdir@#1}\z@
849 \fi
850 \ifodd\bbl@engine
851 \bbl@csarg\ifcase{wdir@#1}%
852 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
853 \or
854 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
855 \or
856 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
857 \fi
858 \fi}
859 \def\bbl@switchdir{%
860 \bbl@ifunset{bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
861 \bbl@ifunset{bbl@wdir\languagename}{\bbl@provide@dirs{\languagename}}{}%
862 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
863 \def\bbl@setdirs#1{% TODO - math
864 \ifcase\bbl@select@type % TODO - strictly, not the right test
865 \bbl@bodydir{#1}%
866 \bbl@pardir{#1}%
867 \fi
868 \bbl@texmdir{#1}}
869 % TODO. Only if \bbl@bidimode > 0?:
870 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
871 \DisableBabelHook{babel-bidi}

```

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

```

872 \ifodd\bbl@engine % luatex=1
873 \chardef\bbl@thetexmdir\z@
874 \chardef\bbl@thepardir\z@
875 \def\bbl@getluadir#1{%

```

```

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

```

```

935      {I'll insert a new group, but expect wrong results.}%
936      \bgroup\aftergroup#2\aftergroup\egroup
937    \else
938      \ifcase\currentgrouptype\or % 0 bottom
939        \aftergroup#2% 1 simple {}
940      \or
941        \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
942      \or
943        \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
944      \or\or\or % vbox vtop align
945      \or
946        \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
947      \or\or\or\or\or\or % output math disc insert vcent mathchoice
948      \or
949        \aftergroup#2% 14 \beginngroup
950      \else
951        \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
952      \fi
953    \fi
954    \bbl@dirlevel\currentgrouplevel
955  \fi
956  #1%
957  \fi}
958  \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
959  \let\bbl@bodydir\@gobble
960  \let\bbl@pagedir\@gobble
961  \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).

```

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

```

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

```

988 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
989 \AtBeginDocument{%
990   \ifx\pdfstringdefDisableCommands\@undefined\else
991     \ifx\pdfstringdefDisableCommands\relax\else
992       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
993     \fi
994   \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`.

```

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

```

1006 \bbl@trace{Language options}
1007 \let\bbl@afterlang\relax
1008 \let\BabelModifiers\relax
1009 \let\bbl@loaded\@empty
1010 \def\bbl@load@language#1{%
1011   \InputIfFileExists{#1.ldf}%
1012   {\edef\bbl@loaded{\CurrentOption
1013     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1014     \expandafter\let\expandafter\bbl@afterlang
1015       \csname\CurrentOption.ldf-h@@k\endcsname
1016     \expandafter\let\expandafter\BabelModifiers
1017       \csname bbl@mod@\CurrentOption\endcsname}%
1018   {\bbl@error{%
1019     Unknown option '\CurrentOption'. Either you misspelled it\\%
1020     or the language definition file \CurrentOption.ldf was not found}{%
1021     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1022     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1023     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.

```

1024 \def\bbl@try@load@lang#1#2#3{%
1025   \IfFileExists{\CurrentOption.ldf}%
1026   {\bbl@load@language{\CurrentOption}}}%
1027   {#1\bbl@load@language{#2}#3}}

```

```

1028 \DeclareOption{hebrew}{%
1029   \input{rlbabel.def}%
1030   \bbl@load@language{hebrew}}
1031 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1032 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1033 \DeclareOption{ nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1034 \DeclareOption{polutonikogreek}{%
1035   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1036 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1037 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1038 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

Another way to extend the list of ‘known’ options for babel was to create the file `bblopts.cfg` in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new `.ldf` file loading the actual one. You can also set the name of the file with the package option `config=<name>`, which will load `<name>.cfg` instead.

```

1039 \ifx\bbl@opt@config\@nnil
1040   \@ifpackagewith{babel}{noconfigs}{}%
1041     {\InputIfFileExists{bblopts.cfg}%
1042       {\typeout{*****^J%
1043         * Local config file bblopts.cfg used^^J%
1044         *}}}%
1045     }{}%
1046 \else
1047   \InputIfFileExists{\bbl@opt@config.cfg}%
1048     {\typeout{*****^J%
1049       * Local config file \bbl@opt@config.cfg used^^J%
1050       *}}}%
1051     {\bbl@error{%
1052       Local config file '\bbl@opt@config.cfg' not found}{%
1053       Perhaps you misspelled it.}}%
1054 \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.

```

1055 \let\bbl@tempc\relax
1056 \bbl@foreach\bbl@language@opts{%
1057   \ifcase\bbl@iniflag % Default
1058     \bbl@ifunset{ds@#1}%
1059     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1060     {}%
1061   \or % provide=*
1062     \@gobble % case 2 same as 1
1063   \or % provide+=*
1064     \bbl@ifunset{ds@#1}%
1065     {\IfFileExists{#1.ldf}{}%
1066      {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}%
1067      {}%
1068     \bbl@ifunset{ds@#1}%
1069     {\def\bbl@tempc{#1}%
1070      \DeclareOption{#1}{%
1071        \ifnum\bbl@iniflag>\@ne
1072          \bbl@ldfinit
1073          \babelprovide[import]{#1}%
1074          \bbl@afterldf}%
1075        \else
1076          \bbl@load@language{#1}%

```



```

1077     \fi}}%
1078   {}%
1079   \or    % provide*=*
1080     \def\bbbl@tempc{#1}%
1081     \bbbl@ifunset{ds@#1}%
1082     {\DeclareOption{#1}{%
1083       \bbbl@ldfinit
1084       \babelprovide[import]{#1}%
1085       \bbbl@afterldf{}}}%
1086     {}%
1087   \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.

```

1088 \let\bbbl@tempb\@nnil
1089 \bbbl@foreach\@classoptionslist{%
1090   \bbbl@ifunset{ds@#1}%
1091   {\IfFileExists{#1.ldf}%
1092     {\def\bbbl@tempb{#1}%
1093       \DeclareOption{#1}{%
1094         \ifnum\bbbl@iniflag>\@ne
1095           \bbbl@ldfinit
1096           \babelprovide[import]{#1}%
1097           \bbbl@afterldf{}}%
1098         \else
1099           \bbbl@load@language{#1}%
1100         \fi}}%
1101     {\IfFileExists{babel-#1.tex}% TODO. Copypaste pattern
1102       {\def\bbbl@tempb{#1}%
1103         \DeclareOption{#1}{%
1104           \ifnum\bbbl@iniflag>\@ne
1105             \bbbl@ldfinit
1106             \babelprovide[import]{#1}%
1107             \bbbl@afterldf{}}%
1108           \else
1109             \bbbl@load@language{#1}%
1110           \fi}}%
1111       {}}}%
1112   {}}

```

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

```

1113 \ifnum\bbbl@iniflag=\z@\else
1114   \ifx\bbbl@opt@main\@nnil
1115     \ifx\bbbl@tempc\relax
1116       \let\bbbl@opt@main\bbbl@tempb
1117     \else
1118       \let\bbbl@opt@main\bbbl@tempc
1119     \fi
1120   \fi
1121 \fi
1122 \ifx\bbbl@opt@main\@nnil\else
1123   \expandafter
1124   \let\expandafter\bbbl@loadmain\csname ds@\bbbl@opt@main\endcsname
1125   \expandafter\let\csname ds@\bbbl@opt@main\endcsname\@empty
1126 \fi

```

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

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

```

1127 \def\AfterBabelLanguage#1{%
1128   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1129 \DeclareOption*{}
1130 \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`.

```

1131 \bbl@trace{Option 'main'}
1132 \ifx\bbl@opt@main\@nnil
1133   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1134   \let\bbl@tempc\@empty
1135   \bbl@for\bbl@tempb\bbl@tempa{%
1136     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1137     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1138   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1139   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1140   \ifx\bbl@tempb\bbl@tempc\else
1141     \bbl@warning{%
1142       Last declared language option is '\bbl@tempc',\%
1143       but the last processed one was '\bbl@tempb'.\%
1144       The main language can't be set as both a global\%
1145       and a package option. Use 'main=\bbl@tempc' as\%
1146       option. Reported}%
1147   \fi
1148 \else
1149   \ifodd\bbl@iniflag % case 1,3
1150     \bbl@ldfinit
1151     \let\CurrentOption\bbl@opt@main
1152     \ifx\bbl@opt@provide\@nnil
1153       \bbl@exp{\@babelprovide[import,main]{\bbl@opt@main}}%
1154     \else
1155       \bbl@exp{\@babelforkv{\@nameuse{@raw@opt@babel.sty}}}{%
1156         \bbl@xin@{,provide,}{, #1,}%
1157         \ifin@
1158           \def\bbl@opt@provide{#2}%
1159           \bbl@replace\bbl@opt@provide{;}{,}%
1160         \fi}%
1161       \bbl@exp{%
1162         \@babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
1163       \fi
1164     \bbl@afterldf}%
1165   \else % case 0,2
1166     \chardef\bbl@iniflag\z@ % Force ldf
1167     \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1168     \ExecuteOptions{\bbl@opt@main}
1169     \DeclareOption*{}%
1170     \ProcessOptions*
1171   \fi
1172 \fi
1173 \def\AfterBabelLanguage{%
1174   \bbl@error
1175   {Too late for \string\AfterBabelLanguage}%
1176   {Languages have been loaded, so I can do nothing}}

```

In order to catch the case where the user forgot to specify a language we check whether

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

```

1177 \ifx\bbl@main@language\@undefined
1178   \bbl@info{%
1179     You haven't specified a language. I'll use 'nil'\%
1180     as the main language. Reported}
1181   \bbl@load@language{nil}
1182 \fi
1183 \</package>
1184 \*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

```

1185 \ifx\ldf@quit\@undefined\else
1186 \endinput\fi % Same line!
1187 <<Make sure ProvidesFile is defined>>
1188 \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).

```

1189 \ifx\AtBeginDocument\@undefined % TODO. change test.
1190   <<Emulate LaTeX>>
1191   \def\languagename{english}%
1192   \let\bbl@opt@shorthands\@nnil
1193   \def\bbl@ifshorthand#1#2#3{#2}%
1194   \let\bbl@language@opts\@empty
1195   \ifx\babeloptionstrings\@undefined
1196     \let\bbl@opt@strings\@nnil
1197   \else
1198     \let\bbl@opt@strings\babeloptionstrings
1199   \fi
1200   \def\BabelStringsDefault{generic}
1201   \def\bbl@tempa{normal}
1202   \ifx\babeloptionmath\bbl@tempa
1203     \def\bbl@mathnormal{\noexpand\textormath}
1204   \fi
1205   \def\AfterBabelLanguage#1#2{}
1206   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1207   \let\bbl@afterlang\relax
1208   \def\bbl@opt@safe{BR}
1209   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1210   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi

```

```

1211 \expandafter\newif\csname ifbbl@single\endcsname
1212 \chardef\bbl@bidimode\z@
1213 \fi

```

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

```

1214 \ifx\bbl@trace\@undefined
1215 \let\LdfInit\endinput
1216 \def\ProvidesLanguage#1{\endinput}
1217 \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.

```

1218 <<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.

```

1219 \def\bbl@version{\<version>}
1220 \def\bbl@date{\<date>}
1221 \def\adddialect#1#2{%
1222   \global\chardef#1#2\relax
1223   \bbl@usehooks{adddialect}{#1}{#2}}%
1224   \begingroup
1225     \count@#1\relax
1226     \def\bbl@elt##1##2##3##4{%
1227       \ifnum\count@=##2\relax
1228         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
1229         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
1230           set to \expandafter\string\csname l@##1\endcsname\\%
1231           (\string\language\the\count@). Reported}%
1232         \def\bbl@elt####1####2####3####4}%
1233       \fi}%
1234   \bbl@cs{languages}%
1235   \endgroup

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error.

The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```

1236 \def\bbl@fixname#1{%
1237   \begingroup
1238   \def\bbl@tempe{l@}%
1239   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1240   \bbl@tempd
1241   {\lowercase\expandafter{\bbl@tempd}%
1242    {\uppercase\expandafter{\bbl@tempd}%
1243     \@empty
1244     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1245      \uppercase\expandafter{\bbl@tempd}}}%
1246   {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1247    \lowercase\expandafter{\bbl@tempd}}}%
1248   \@empty
1249   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%

```

```

1250 \bbl@tempd
1251 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}
1252 \def\bbl@iflanguage#1{%
1253 \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.

```

1254 \def\bbl@bcpcase#1#2#3#4\@#5{%
1255 \ifx\@empty#3%
1256 \uppercase{\def#5{#1#2}}%
1257 \else
1258 \uppercase{\def#5{#1}}%
1259 \lowercase{\edef#5{#5#2#3#4}}%
1260 \fi}
1261 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1262 \let\bbl@bcp\relax
1263 \lowercase{\def\bbl@tempa{#1}}%
1264 \ifx\@empty#2%
1265 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1266 \else\ifx\@empty#3%
1267 \bbl@bcpcase#2\@empty\@empty\@bbl@tempb
1268 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1269 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1270 {}%
1271 \ifx\bbl@bcp\relax
1272 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1273 \fi
1274 \else
1275 \bbl@bcpcase#2\@empty\@empty\@bbl@tempb
1276 \bbl@bcpcase#3\@empty\@empty\@bbl@tempc
1277 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1278 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1279 {}%
1280 \ifx\bbl@bcp\relax
1281 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1282 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1283 {}%
1284 \fi
1285 \ifx\bbl@bcp\relax
1286 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1287 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1288 {}%
1289 \fi
1290 \ifx\bbl@bcp\relax
1291 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1292 \fi
1293 \fi\fi}
1294 \let\bbl@initoload\relax
1295 \def\bbl@provide@locale{%
1296 \ifx\babelprovide\undefined
1297 \bbl@error{For a language to be defined on the fly 'base'\\%
1298 is not enough, and the whole package must be\\%
1299 loaded. Either delete the 'base' option or\\%
1300 request the languages explicitly}%
1301 {See the manual for further details.}%
1302 \fi

```

```

1303% TODO. Option to search if loaded, with \LocaleForEach
1304 \let\bb1@auxname\language % Still necessary. TODO
1305 \bb1@ifunset{bb1@bcp@map@\language}{}% Move uplevel??
1306 {\edef\language{\@nameuse{bb1@bcp@map@\language}}}%
1307 \ifbb1@bcpallowed
1308 \expandafter\ifx\csname date\language\endcsname\relax
1309 \expandafter
1310 \bb1@bcplookup\language-\@empty-\@empty-\@empty\@@
1311 \ifx\bb1@bcp\relax\else % Returned by \bb1@bcplookup
1312 \edef\language{\bb1@bcp@prefix\bb1@bcp}%
1313 \edef\localename{\bb1@bcp@prefix\bb1@bcp}%
1314 \expandafter\ifx\csname date\language\endcsname\relax
1315 \let\bb1@initoload\bb1@bcp
1316 \bb1@exp{\\\babelprovide[\bb1@autoload@bcptoptions]{\language}}%
1317 \let\bb1@initoload\relax
1318 \fi
1319 \bb1@csarg\xdef{bcp@map@\bb1@bcp}{\localename}%
1320 \fi
1321 \fi
1322 \fi
1323 \expandafter\ifx\csname date\language\endcsname\relax
1324 \IfFileExists{babel-\language.tex}%
1325 {\bb1@exp{\\\babelprovide[\bb1@autoload@options]{\language}}}%
1326 {}%
1327 \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.

```

1328 \def\iflanguage#1{%
1329 \bb1@iflanguage{#1}{%
1330 \ifnum\csname l@#1\endcsname=\language
1331 \expandafter\@firstoftwo
1332 \else
1333 \expandafter\@secondoftwo
1334 \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.

```

1335 \let\bb1@select@type\z@
1336 \edef\selectlanguage{%
1337 \noexpand\protect
1338 \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`.

```

1339 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

1340 \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.

```
1341 \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
1342 \def\bbl@push@language{%
1343   \ifx\language\@undefined\else
1344     \ifx\currentgrouplevel\@undefined
1345       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1346     \else
1347       \ifnum\currentgrouplevel=\z@
1348         \xdef\bbl@language@stack{\language+}%
1349       \else
1350         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1351       \fi
1352     \fi
1353 \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`.

```
1354 \def\bbl@pop@lang#1+#2\@{%
1355   \edef\language{#1}%
1356   \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).

```
1357 \let\bbl@ifrestoring\@secondoftwo
1358 \def\bbl@pop@language{%
1359   \expandafter\bbl@pop@lang\bbl@language@stack\@
1360   \let\bbl@ifrestoring\@firstoftwo
1361   \expandafter\bbl@set@language\expandafter{\language}%
1362   \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).

```
1363 \chardef\localeid\z@
1364 \def\bbl@id@last{0} % No real need for a new counter
1365 \def\bbl@id@assign{%
```

```

1366 \bbl@ifunset{bbl@id@@\language}%
1367 {\count@bbl@id@last\relax
1368 \advance\count@one
1369 \bbl@csarg\chardef{id@@\language}\count@
1370 \edef\bbl@id@last{\the\count@}%
1371 \ifcase\bbl@engine\or
1372 \directlua{
1373     Babel = Babel or {}
1374     Babel.locale_props = Babel.locale_props or {}
1375     Babel.locale_props[\bbl@id@last] = {}
1376     Babel.locale_props[\bbl@id@last].name = '\language'
1377 }%
1378 \fi}%
1379 {}%
1380 \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

1381 \expandafter\def\csname selectlanguage \endcsname#1{%
1382 \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw@fi
1383 \bbl@push@language
1384 \aftergroup\bbl@pop@language
1385 \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).

```

1386 \def\BabelContentsFiles{toc,lof,lot}
1387 \def\bbl@set@language#1{% from selectlanguage, pop@
1388 % The old buggy way. Preserved for compatibility.
1389 \edef\language{%
1390 \ifnum\escapechar=\expandafter`\string#1\@empty
1391 \else\string#1\@empty\fi}%
1392 \ifcat\relax\noexpand#1%
1393 \expandafter\ifx\csname date\language\endcsname\relax
1394 \edef\language{#1}%
1395 \let\localename\language
1396 \else
1397 \bbl@info{Using '\string\language' instead of 'language' is%%
1398 deprecated. If what you want is to use a%%
1399 macro containing the actual locale, make%%
1400 sure it does not not match any language.%%
1401 Reported}%
1402 \ifx\scantokens\@undefined
1403 \def\localename{??}%
1404 \else
1405 \scantokens\expandafter{\expandafter
1406 \def\expandafter\localename\expandafter{\language}}%
1407 \fi
1408 \fi
1409 \else

```



```

1410 \def\localename{#1}% This one has the correct catcodes
1411 \fi
1412 \select@language{\language}%
1413 % write to auxs
1414 \expandafter\ifx\csname date\language\endcsname\relax\else
1415 \if@filesw
1416 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1417 \bbl@savelastskip
1418 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1419 \bbl@restorelastskip
1420 \fi
1421 \bbl@usehooks{write}{}}%
1422 \fi
1423 \fi}
1424 %
1425 \let\bbl@restorelastskip\relax
1426 \def\bbl@savelastskip{%
1427 \let\bbl@restorelastskip\relax
1428 \ifvmode
1429 \ifdim\lastskip=\z@
1430 \let\bbl@restorelastskip\nobreak
1431 \else
1432 \bbl@exp{%
1433 \def\\bbl@restorelastskip{%
1434 \skip@=\the\lastskip
1435 \\nobreak \vskip-\skip@ \vskip\skip@}}%
1436 \fi
1437 \fi}
1438 %
1439 \newif\ifbbl@bcpallowed
1440 \bbl@bcpallowedfalse
1441 \def\select@language#1{% from set@, babel@aux
1442 % set hmap
1443 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1444 % set name
1445 \edef\language{#1}%
1446 \bbl@fixname\language
1447 % TODO. name@map must be here?
1448 \bbl@provide@locale
1449 \bbl@iflanguage\language{%
1450 \expandafter\ifx\csname date\language\endcsname\relax
1451 \bbl@error
1452 {Unknown language '\language'. Either you have\\%
1453 misspelled its name, it has not been installed,\\%
1454 or you requested it in a previous run. Fix its name,\\%
1455 install it or just rerun the file, respectively. In\\%
1456 some cases, you may need to remove the aux file}%
1457 {You may proceed, but expect wrong results}%
1458 \else
1459 % set type
1460 \let\bbl@select@type\z@
1461 \expandafter\bbl@switch\expandafter{\language}%
1462 \fi}}
1463 \def\babel@aux#1#2{%
1464 \select@language{#1}%
1465 \bbl@foreach\BabelContentsFiles{% \relax: don't assume vertical mode
1466 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
1467 \def\babel@toc#1#2{%
1468 \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.

```

1469 \newif\ifbbl@usedategroup
1470 \def\bbl@switch#1{% from select@, foreign@
1471 % make sure there is info for the language if so requested
1472 \bbl@ensureinfo{#1}%
1473 % restore
1474 \originalTeX
1475 \expandafter\def\expandafter\originalTeX\expandafter{%
1476 \csname noextras#1\endcsname
1477 \let\originalTeX\empty
1478 \babel@beginsave}%
1479 \bbl@usehooks{afterreset}{}%
1480 \languageshorthands{none}%
1481 % set the locale id
1482 \bbl@id@assign
1483 % switch captions, date
1484 % No text is supposed to be added here, so we remove any
1485 % spurious spaces.
1486 \bbl@bsphack
1487 \ifcase\bbl@select@type
1488 \csname captions#1\endcsname\relax
1489 \csname date#1\endcsname\relax
1490 \else
1491 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1492 \ifin@
1493 \csname captions#1\endcsname\relax
1494 \fi
1495 \bbl@xin@{,date,}{, \bbl@select@opts,}%
1496 \ifin@ % if \foreign... within \<lang>date
1497 \csname date#1\endcsname\relax
1498 \fi
1499 \fi
1500 \bbl@esphack
1501 % switch extras
1502 \bbl@usehooks{beforeextras}{}%
1503 \csname extras#1\endcsname\relax
1504 \bbl@usehooks{afterextras}{}%
1505 % > babel-ensure
1506 % > babel-sh-<short>
1507 % > babel-bidi
1508 % > babel-fontspec
1509 % hyphenation - case mapping
1510 \ifcase\bbl@opt@hyphenmap\or
1511 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1512 \ifnum\bbl@hymapsel>4\else
1513 \csname\language @bbl@hyphenmap\endcsname
1514 \fi

```

```

1515 \chardef\bbbl@opt@hyphenmap\z@
1516 \else
1517 \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
1518 \csname\language\language @bbbl@hyphenmap\endcsname
1519 \fi
1520 \fi
1521 \let\bbbl@hymapsel\@cclv
1522 % hyphenation - select rules
1523 \ifnum\csname l@\language\endcsname=\l@unhyphenated
1524 \edef\bbbl@tempa{u}%
1525 \else
1526 \edef\bbbl@tempa{\bbbl@cl{lnbrk}}%
1527 \fi
1528 % linebreaking - handle u, e, k (v in the future)
1529 \bbbl@xin@{/u}{/\bbbl@tempa}%
1530 \ifin@else\bbbl@xin@{/e}{/\bbbl@tempa}\fi % elongated forms
1531 \ifin@else\bbbl@xin@{/k}{/\bbbl@tempa}\fi % only kashida
1532 \ifin@else\bbbl@xin@{/v}{/\bbbl@tempa}\fi % variable font
1533 \ifin@
1534 % unhyphenated/kashida/elongated = allow stretching
1535 \language\l@unhyphenated
1536 \babel@savevariable\emergencystretch
1537 \emergencystretch\maxdimen
1538 \babel@savevariable\hbadness
1539 \hbadness\@M
1540 \else
1541 % other = select patterns
1542 \bbbl@patterns{#1}%
1543 \fi
1544 % hyphenation - mins
1545 \babel@savevariable\lefthyphenmin
1546 \babel@savevariable\righthyphenmin
1547 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1548 \set@hyphenmins\tw@\thr@\relax
1549 \else
1550 \expandafter\expandafter\expandafter\set@hyphenmins
1551 \csname #1hyphenmins\endcsname\relax
1552 \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.

```

1553 \long\def\otherlanguage#1{%
1554 \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@\fi
1555 \csname selectlanguage \endcsname{#1}%
1556 \ignorespaces}

```

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

```

1557 \long\def\endotherlanguage{%
1558 \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`.

```

1559 \expandafter\def\csname otherlanguage*\endcsname{%

```

```

1560 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}{
1561 \def\bbl@otherlanguage@s[#1]#2{%
1562 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1563 \def\bbl@select@opts{#1}%
1564 \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”.

```

1565 \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`.

```

1566 \providecommand\bbl@beforeforeign{}
1567 \edef\foreignlanguage{%
1568 \noexpand\protect
1569 \expandafter\noexpand\csname foreignlanguage \endcsname}
1570 \expandafter\def\csname foreignlanguage \endcsname{%
1571 \@ifstar\bbl@foreign@s\bbl@foreign@x}
1572 \providecommand\bbl@foreign@x[3][]{%
1573 \begingroup
1574 \def\bbl@select@opts{#1}%
1575 \let\BabelText\@firstofone
1576 \bbl@beforeforeign
1577 \foreign@language{#2}%
1578 \bbl@usehooks{foreign}{}}%
1579 \BabelText{#3}% Now in horizontal mode!
1580 \endgroup}
1581 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1582 \begingroup
1583 {\par}%
1584 \let\bbl@select@opts\@empty
1585 \let\BabelText\@firstofone
1586 \foreign@language{#1}%
1587 \bbl@usehooks{foreign*}{}}%
1588 \bbl@dirparastext
1589 \BabelText{#2}% Still in vertical mode!
1590 {\par}%
1591 \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`.

```

1592 \def\foreign@language#1{%
1593   % set name
1594   \edef\language#1}%
1595   \ifbbl@usedategroup
1596     \bbl@add\bbl@select@opts{,date,}%
1597     \bbl@usedategroupfalse
1598   \fi
1599   \bbl@fixname\language
1600   % TODO. name@map here?
1601   \bbl@provide@locale
1602   \bbl@iflanguage\language{%
1603     \expandafter\ifx\csname date\language\endcsname\relax
1604       \bbl@warning % TODO - why a warning, not an error?
1605         {Unknown language '#1'. Either you have\\%
1606           misspelled its name, it has not been installed,\\%
1607           or you requested it in a previous run. Fix its name,\\%
1608           install it or just rerun the file, respectively. In\\%
1609           some cases, you may need to remove the aux file.\\%
1610           I'll proceed, but expect wrong results.\\%
1611           Reported}%
1612     \fi
1613     % set type
1614     \let\bbl@select@type\@ne
1615     \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.

```

1616 \let\bbl@hyphlist\@empty
1617 \let\bbl@hyphenation@\relax
1618 \let\bbl@pttnlist\@empty
1619 \let\bbl@patterns@\relax
1620 \let\bbl@hymapsel=\@cclv
1621 \def\bbl@patterns#1{%
1622   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1623     \csname l@#1\endcsname
1624     \edef\bbl@tempa{#1}%
1625   \else
1626     \csname l@#1:\f@encoding\endcsname
1627     \edef\bbl@tempa{#1:\f@encoding}%
1628   \fi
1629   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1630 % > luatex
1631 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1632   \begingroup
1633     \bbl@xin@{\, \number\language,}{, \bbl@hyphlist}%
1634   \ifin\else
1635     \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
1636   \hyphenation{%
1637     \bbl@hyphenation@

```

```

1638      \ifundefined{bbl@hyphenation@#1}%
1639      \empty
1640      {\space\csname bbl@hyphenation@#1\endcsname}}%
1641      \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1642      \fi
1643      \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*`.

```

1644 \def\hyphenrules#1{%
1645   \edef\bbl@tempf{#1}%
1646   \bbl@fixname\bbl@tempf
1647   \bbl@iflanguage\bbl@tempf{%
1648     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1649     \ifx\languageshortands\undefined\else
1650       \languageshortands{none}%
1651     \fi
1652     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1653       \set@hyphenmins\tw@\thr@@\relax
1654     \else
1655       \expandafter\expandafter\expandafter\set@hyphenmins
1656       \csname\bbl@tempf hyphenmins\endcsname\relax
1657     \fi}}
1658 \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.

```

1659 \def\providehyphenmins#1#2{%
1660   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1661     \@namedef{#1hyphenmins}{#2}%
1662   \fi}

```

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

```

1663 \def\set@hyphenmins#1#2{%
1664   \lefthyphenmin#1\relax
1665   \righthyphenmin#2\relax}

```

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

```

1666 \ifx\ProvidesFile\undefined
1667   \def\ProvidesLanguage#1[#2 #3 #4]{%
1668     \wlog{Language: #1 #4 #3 <#2>}%
1669   }
1670 \else
1671   \def\ProvidesLanguage#1{%
1672     \begingroup
1673     \catcode`\ 10 %
1674     \@makeother\%
1675     \@ifnextchar[%]
1676       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1677   \def\@provideslanguage#1[#2]{%
1678     \wlog{Language: #1 #2}%

```

```

1679 \expandafter\edef\csname ver@#1.1df\endcsname{#2}%
1680 \endgroup}
1681 \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.

1682 \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.

1683 \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':

1684 \providecommand\setlocale{%
1685 \bbl@error
1686 {Not yet available}%
1687 {Find an armchair, sit down and wait}}
1688 \let\uselocale\setlocale
1689 \let\locale\setlocale
1690 \let\selectlocale\setlocale
1691 \let\localename\setlocale
1692 \let\textlocale\setlocale
1693 \let\textlanguage\setlocale
1694 \let\languagegettext\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 $\text{\LaTeX 2}_{\epsilon}$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

1695 \edef\bbl@nulllanguage{\string\language=0}
1696 \ifx\PackageError\undefined % TODO. Move to Plain
1697 \def\bbl@error#1#2{%
1698 \begingroup
1699 \newlinechar=`^^J
1700 \def\{^^J(babel) }%
1701 \errhelp{#2}\errmessage{\{#1}%
1702 \endgroup}
1703 \def\bbl@warning#1{%
1704 \begingroup
1705 \newlinechar=`^^J
1706 \def\{^^J(babel) }%
1707 \message{\{#1}%
1708 \endgroup}
1709 \let\bbl@infowarn\bbl@warning
1710 \def\bbl@info#1{%
1711 \begingroup
1712 \newlinechar=`^^J
1713 \def\{^^J}%
1714 \wlog{#1}%

```

```

1715 \endgroup}
1716 \fi
1717 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1718 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1719 \global\@namedef{#2}{\textbf{?#1?}}}%
1720 \@nameuse{#2}%
1721 \edef\bbl@tempa{#1}%
1722 \bbl@sreplace\bbl@tempa{name}{}%
1723 \bbl@warning{% TODO.
1724 \@backslashchar#1 not set for '\language'. Please,\%
1725 define it after the language has been loaded\%
1726 (typically in the preamble) with:\%
1727 \string\setlocalecaption{\language}{\bbl@tempa}{..\%
1728 Reported}}
1729 \def\bbl@tentative{\protect\bbl@tentative@i}
1730 \def\bbl@tentative@i#1{%
1731 \bbl@warning{%
1732 Some functions for '#1' are tentative.\%
1733 They might not work as expected and their behavior\%
1734 could change in the future.\%
1735 Reported}}
1736 \def\@nolanerr#1{%
1737 \bbl@error
1738 {You haven't defined the language '#1' yet.\%
1739 Perhaps you misspelled it or your installation\%
1740 is not complete}%
1741 {Your command will be ignored, type <return> to proceed}}
1742 \def\@nopatterns#1{%
1743 \bbl@warning
1744 {No hyphenation patterns were preloaded for\%
1745 the language '#1' into the format.\%
1746 Please, configure your TeX system to add them and\%
1747 rebuild the format. Now I will use the patterns\%
1748 preloaded for \bbl@nulllanguage\space instead}}
1749 \let\bbl@usehooks\@gobbletwo
1750 \ifx\bbl@onlyswitch\@empty\endinput\fi
1751 % Here ended switch.def

Here ended switch.def.

1752 \ifx\directlua\@undefined\else
1753 \ifx\bbl@luapatterns\@undefined
1754 \input luababel.def
1755 \fi
1756 \fi
1757 <<Basic macros>>
1758 \bbl@trace{Compatibility with language.def}
1759 \ifx\bbl@languages\@undefined
1760 \ifx\directlua\@undefined
1761 \openin1 = language.def % TODO. Remove hardcoded number
1762 \ifeof1
1763 \closein1
1764 \message{I couldn't find the file language.def}
1765 \else
1766 \closein1
1767 \begingroup
1768 \def\addlanguage#1#2#3#4#5{%
1769 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1770 \global\expandafter\let\csname l@#1\endcsname
1771 \csname lang@#1\endcsname

```



```

1772      \fi}%
1773      \def\uselanguage#1{%
1774      \input language.def
1775      \endgroup
1776      \fi
1777      \fi
1778      \chardef\l@english\z@
1779      \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.

```

1780 \def\addto#1#2{%
1781   \ifx#1\undefined
1782     \def#1{#2}%
1783   \else
1784     \ifx#1\relax
1785       \def#1{#2}%
1786     \else
1787       {\toks@\expandafter{#1#2}%
1788        \xdef#1{\the\toks@}}%
1789     \fi
1790   \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?

```

1791 \def\bbl@withactive#1#2{%
1792   \begingroup
1793   \lccode`~=#2\relax
1794   \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`.

```

1795 \def\bbl@redefine#1{%
1796   \edef\bbl@tempa{\bbl@stripslash#1}%
1797   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1798   \expandafter\def\csname\bbl@tempa\endcsname{
1799   \@onlypreamble\bbl@redefine

```

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

```

1800 \def\bbl@redefine@long#1{%
1801   \edef\bbl@tempa{\bbl@stripslash#1}%
1802   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1803   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1804   \@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_`.

```

1805 \def\bbl@redefineroobust#1{%
1806   \edef\bbl@tempa{\bbl@stripslash#1}%
1807   \bbl@ifunset{\bbl@tempa\space}%
1808   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%

```

```

1809 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1810 {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1811 \@namedef{\bbl@tempa\space}}
1812 \@onlypreamble\bbl@redefineroast

```

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.

```

1813 \bbl@trace{Hooks}
1814 \newcommand\AddBabelHook[3][\%
1815 \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{\%
1816 \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}}%
1817 \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1818 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1819 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1820 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1821 \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1822 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1823 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1824 \def\bbl@usehooks#1#2{%
1825 \def\bbl@elth##1{%
1826 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}}%
1827 \bbl@cs{ev@#1@}%
1828 \ifx\language\@undefined\else % Test required for Plain (?)
1829 \def\bbl@elth##1{%
1830 \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}}%
1831 \bbl@cl{ev@#1}%
1832 \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).

```

1833 \def\bbl@evargs{,% <- don't delete this comma
1834 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1835 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1836 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1837 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1838 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.

```

1839 \bbl@trace{Defining babelensure}
1840 \newcommand\babelensure[2][\% TODO - revise test files
1841 \AddBabelHook{babel-ensure}{afterextras}{\%
1842 \ifcase\bbl@select@type
1843 \bbl@cl{e}%
1844 \fi}%
1845 \begingroup
1846 \let\bbl@ens@include\@empty

```

```

1847 \let\bb1@ens@exclude\@empty
1848 \def\bb1@ens@fontenc{\relax}%
1849 \def\bb1@tempb##1{%
1850   \ifx\@empty##1\else\noexpand##1\expandafter\bb1@tempb\fi}%
1851 \edef\bb1@tempa{\bb1@tempb#1\@empty}%
1852 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ens@##1}{##2}}%
1853 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
1854 \def\bb1@tempc{\bb1@ensure}%
1855 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1856   \expandafter{\bb1@ens@include}}%
1857 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1858   \expandafter{\bb1@ens@exclude}}%
1859 \toks@\expandafter{\bb1@tempc}%
1860 \bb1@exp{%
1861 \endgroup
1862 \def\<bb1@e@#2>{\the\toks@{\bb1@ens@fontenc}}}%
1863 \def\bb1@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1864 \def\bb1@tempb##1{% elt for (excluding) \bb1@captionslist list
1865   \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1866     \edef##1{\noexpand\bb1@nocaption
1867       {\bb1@stripslash##1}{\language\bb1@stripslash##1}}%
1868   \fi
1869   \ifx##1\@empty\else
1870     \in@{##1}{#2}%
1871     \ifin@\else
1872       \bb1@ifunset{bb1@ensure@\language}%
1873       {\bb1@exp{%
1874         \\\DeclareRobustCommand\<bb1@ensure@\language>[1]{%
1875           \\\foreignlanguage{\language}%
1876             {\ifx\relax#3\else
1877               \\\fontencoding{#3}\selectfont
1878               \fi
1879               #####1}}}%
1880       }%
1881       \toks@\expandafter{##1}%
1882       \edef##1{%
1883         \bb1@csarg\noexpand{ensure@\language}%
1884         {\the\toks@}}%
1885       \fi
1886       \expandafter\bb1@tempb
1887       \fi}%
1888 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
1889 \def\bb1@tempa##1{% elt for include list
1890   \ifx##1\@empty\else
1891     \bb1@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1892     \ifin@\else
1893       \bb1@tempb##1\@empty
1894     \fi
1895     \expandafter\bb1@tempa
1896     \fi}%
1897 \bb1@tempa#1\@empty}
1898 \def\bb1@captionslist{%
1899 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1900 \contentsname\listfigurename\listtablename\indexname\figurename
1901 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1902 \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 at-sign. We make sure that it is a 'letter' during the processing of the file. We also save its name as the last called option, even if not loaded.

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

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

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

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

Finally we check `\originalTeX`.

```

1903 \bbl@trace{Macros for setting language files up}
1904 \def\bbl@ldfinit{%
1905   \let\bbl@screset\@empty
1906   \let\BabelStrings\bbl@opt@string
1907   \let\BabelOptions\@empty
1908   \let\BabelLanguages\relax
1909   \ifx\originalTeX\@undefined
1910     \let\originalTeX\@empty
1911   \else
1912     \originalTeX
1913   \fi}
1914 \def\LdfInit#1#2{%
1915   \chardef\atcatcode=\catcode`\@
1916   \catcode`\@=11\relax
1917   \chardef\eqcatcode=\catcode`\=
1918   \catcode`\==12\relax
1919   \expandafter\if\expandafter\@backslashchar
1920     \expandafter\@car\string#2\@nil
1921     \ifx#2\@undefined\else
1922       \ldf@quit{#1}%
1923     \fi
1924   \else
1925     \expandafter\ifx\csname#2\endcsname\relax\else
1926       \ldf@quit{#1}%
1927     \fi
1928   \fi
1929   \bbl@ldfinit}

```

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

```

1930 \def\ldf@quit#1{%
1931   \expandafter\main@language\expandafter{#1}%
1932   \catcode`\@=\atcatcode \let\atcatcode\relax
1933   \catcode`\==\eqcatcode \let\eqcatcode\relax
1934   \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.

```

1935 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1936   \bbl@afterlang
1937   \let\bbl@afterlang\relax
1938   \let\BabelModifiers\relax
1939   \let\bbl@screset\relax}%
1940 \def\ldf@finish#1{%
1941   \ifx\loadlocalcfg@undefined\else % For LaTeX 209
1942     \loadlocalcfg{#1}%
1943   \fi
1944   \bbl@afterldf{#1}%
1945   \expandafter\main@language\expandafter{#1}%
1946   \catcode`\@=\atcatcode \let\atcatcode\relax
1947   \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 .

```

1948 \@onlypreamble\LdfInit
1949 \@onlypreamble\ldf@quit
1950 \@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.

```

1951 \def\main@language#1{%
1952   \def\bbl@main@language{#1}%
1953   \let\language\let\bbl@main@language % TODO. Set localename
1954   \bbl@id@assign
1955   \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.

```

1956 \def\bbl@beforestart{%
1957   \def\@nolanerr##1{%
1958     \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1959   \bbl@usehooks{beforestart}{}%
1960   \global\let\bbl@beforestart\relax}
1961 \AtBeginDocument{%
1962   {\@nameuse{bbl@beforestart}}% Group!
1963   \if@filesw
1964     \providecommand\babel@aux[2]{}%
1965     \immediate\write\@mainaux{%
1966       \string\providecommand\string\babel@aux[2]{}%
1967       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1968   \fi
1969   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1970   \ifbbl@single % must go after the line above.
1971     \renewcommand\selectlanguage[1]{}%
1972     \renewcommand\foreignlanguage[2]{#2}%
1973     \global\let\babel@aux\@gobbles % Also as flag
1974   \fi
1975   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1976 \def\select@language@x#1{%
1977   \ifcase\bbl@select@type

```

```

1978 \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1979 \else
1980 \select@language{#1}%
1981 \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 \TeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1982 \bbl@trace{Shorhands}
1983 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1984 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1985 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1986 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1987 \begingroup
1988 \catcode`#1\active
1989 \nfss@catcodes
1990 \ifnum\catcode`#1=\active
1991 \endgroup
1992 \bbl@add\nfss@catcodes{\@makeother#1}%
1993 \else
1994 \endgroup
1995 \fi
1996 \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.

```

1997 \def\bbl@remove@special#1{%
1998 \begingroup
1999 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
2000 \else\noexpand##1\noexpand##2\fi}%
2001 \def\do{\x\do}%
2002 \def\@makeother{\x\@makeother}%
2003 \edef\x{\endgroup
2004 \def\noexpand\dospecials{\dospecials}%
2005 \expandafter\ifx\csname @sanitize\endscname\relax\else
2006 \def\noexpand\@sanitize{\@sanitize}%
2007 \fi}%
2008 \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` ($\langle char \rangle$) to expand to the character in its 'normal state' and it defines the active character to expand to `\normal@char` ($\langle char \rangle$) by default ($\langle char \rangle$ being the character to be made active). Later its definition can be changed to expand to `\active@char` ($\langle char \rangle$) by calling `\bbl@activate{\langle char \rangle}`. 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).

```

2009 \def\bbl@active@def#1#2#3#4{%
2010   \@namedef{#3#1}{%
2011     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
2012       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
2013     \else
2014       \bbl@afterfi\csname#2@sh@#1\endcsname
2015     \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.

```

2016   \long\@namedef{#3@arg#1}##1{%
2017     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
2018       \bbl@afterelse\csname#4#1\endcsname##1%
2019     \else
2020       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
2021     \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.

```

2022 \def\initiate@active@char#1{%
2023   \bbl@ifunset{active@char\string#1}%
2024   {\bbl@withactive
2025     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
2026   {}}

```

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).

```

2027 \def\@initiate@active@char#1#2#3{%
2028   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
2029   \ifx#1\@undefined
2030     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
2031   \else
2032     \bbl@csarg\let{oridef@@#2}#1%
2033     \bbl@csarg\edef{oridef@#2}{%
2034       \let\noexpand#1%
2035       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
2036   \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).

```

2037   \ifx#1#3\relax
2038     \expandafter\let\csname normal@char#2\endcsname#3%
2039   \else
2040     \bbl@info{Making #2 an active character}%
2041     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2042     \@namedef{normal@char#2}{%
2043       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
2044   \else
2045     \@namedef{normal@char#2}{#3}%
2046   \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).

```

2047 \bbl@restoreactive{#2}%
2048 \AtBeginDocument{%
2049   \catcode`#2\active
2050   \if@filesw
2051     \immediate\write\@mainaux{\catcode`\string#2\active}%
2052   \fi}%
2053 \expandafter\bbl@add@special\csname#2\endcsname
2054 \catcode`#2\active
2055 \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}`).

```

2056 \let\bbl@tempa\@firstoftwo
2057 \if\string^#2%
2058   \def\bbl@tempa{\noexpand\textormath}%
2059 \else
2060   \ifx\bbl@mathnormal\@undefined\else
2061     \let\bbl@tempa\bbl@mathnormal
2062   \fi
2063 \fi
2064 \expandafter\edef\csname active@char#2\endcsname{%
2065   \bbl@tempa
2066     {\noexpand\if@safe@actives
2067       \noexpand\expandafter
2068         \expandafter\noexpand\csname normal@char#2\endcsname
2069       \noexpand\else
2070         \noexpand\expandafter
2071         \expandafter\noexpand\csname bbl@doactive#2\endcsname
2072       \noexpand\fi}%
2073   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2074 \bbl@csarg\edef{doactive#2}{%
2075   \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!).

```

2076 \bbl@csarg\edef{active@#2}{%
2077   \noexpand\active@prefix\noexpand#1%
2078   \expandafter\noexpand\csname active@char#2\endcsname}%
2079 \bbl@csarg\edef{normal@#2}{%
2080   \noexpand\active@prefix\noexpand#1%
2081   \expandafter\noexpand\csname normal@char#2\endcsname}%
2082 \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.

```

2083 \bbl@active@def#2\user@group{user@active}{language@active}%

```



```

2084 \bbl@active@def#2\language@group{language@active}{system@active}%
2085 \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.

```

2086 \expandafter\edef\csname\user@group @sh@#2@\endcsname
2087 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2088 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
2089 {\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.

```

2090 \if\string'#2%
2091 \let\prim@s\bbl@prim@s
2092 \let\active@math@prime#1%
2093 \fi
2094 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

2095 <<{*More package options}>> ≡
2096 \DeclareOption{math=active}{}
2097 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2098 <</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.

```

2099 \@ifpackagewith{babel}{KeepShorthandsActive}%
2100 {\let\bbl@restoreactive\@gobble}%
2101 {\def\bbl@restoreactive#1{%
2102   \bbl@exp{%
2103     \\\AfterBabelLanguage\\\CurrentOption
2104     {\catcode`#1=\the\catcode`#1\relax}%
2105     \\\AtEndOfPackage
2106     {\catcode`#1=\the\catcode`#1\relax}}}%
2107 \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.

```

2108 \def\bbl@sh@select#1#2{%
2109   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2110     \bbl@afterelse\bbl@scndcs
2111   \else
2112     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2113   \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 `\ifincname` is available. If there is, the expansion will be more robust.

```

2114 \begingroup
2115 \bbl@ifunset{ifincname}% TODO. Ugly. Correct?
2116 {\gdef\active@prefix#1{%
2117   \ifx\protect\@typeset@protect
2118   \else
2119     \ifx\protect\@unexpandable@protect
2120     \noexpand#1%
2121     \else
2122       \protect#1%
2123       \fi
2124     \expandafter\@gobble
2125   \fi}}
2126 {\gdef\active@prefix#1{%
2127   \ifincname
2128   \string#1%
2129   \expandafter\@gobble
2130   \else
2131     \ifx\protect\@typeset@protect
2132     \else
2133       \ifx\protect\@unexpandable@protect
2134       \noexpand#1%
2135       \else
2136         \protect#1%
2137         \fi
2138       \expandafter\expandafter\expandafter\@gobble
2139     \fi
2140   \fi}}
2141 \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⟩`.

```

2142 \newif\if@safe@actives
2143 \@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.

```

2144 \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`.

```

2145 \chardef\bbl@activated\z@
2146 \def\bbl@activate#1{%
2147   \chardef\bbl@activated\@ne
2148   \bbl@withactive{\expandafter\let\expandafter}#1%
2149   \csname bbl@active@\string#1\endcsname}
2150 \def\bbl@deactivate#1{%
2151   \chardef\bbl@activated\tw@
2152   \bbl@withactive{\expandafter\let\expandafter}#1%
2153   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2154 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2155 \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.

```

2156 \def\babel@texpdf#1#2#3#4{%
2157   \ifx\texorpdfstring\@undefined
2158     \textormath{#1}{#3}%
2159   \else
2160     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2161     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2162   \fi}
2163 %
2164 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2165 \def\@decl@short#1#2#3\@nil#4{%
2166   \def\bbl@tempa{#3}%
2167   \ifx\bbl@tempa\@empty
2168     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2169     \bbl@ifunset{#1@sh@\string#2@}{}%
2170     {\def\bbl@tempa{#4}%
2171      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2172      \else
2173        \bbl@info
2174        {Redefining #1 shorthand \string#2\\%
2175         in language \CurrentOption}%
2176      \fi}%
2177     \@namedef{#1@sh@\string#2@}{#4}%
2178   \else
2179     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2180     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2181     {\def\bbl@tempa{#4}%
2182      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2183      \else
2184        \bbl@info
2185        {Redefining #1 shorthand \string#2\string#3\\%
2186         in language \CurrentOption}%
2187      \fi}%
2188     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2189   \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.

```

2190 \def\textormath{%
2191   \ifmmode
2192     \expandafter\@secondoftwo
2193   \else
2194     \expandafter\@firstoftwo
2195   \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’.

```

2196 \def\user@group{user}
2197 \def\language@group{english} % TODO. I don't like defaults
2198 \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.

2199 \def\useshorthands{%
2200   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2201 \def\bbl@usesh@s#1{%
2202   \bbl@usesh@x
2203   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2204   {#1}}
2205 \def\bbl@usesh@x#1#2{%
2206   \bbl@ifshorthand{#2}%
2207   {\def\user@group{user}%
2208     \initiate@active@char{#2}%
2209     #1%
2210     \bbl@activate{#2}}%
2211   {\bbl@error
2212     {I can't declare a shorthand turned off (\string#2)}
2213     {Sorry, but you can't use shorthands which have been\\
2214       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.

2215 \def\user@language@group{user@\language@group}
2216 \def\bbl@set@user@generic#1#2{%
2217   \bbl@ifunset{user@generic@active#1}%
2218   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2219     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2220     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2221       \expandafter\noexpand\csname normal@char#1\endcsname}%
2222     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2223       \expandafter\noexpand\csname user@active#1\endcsname}}%
2224   \@empty}
2225 \newcommand\defineshorthand[3][user]{%
2226   \edef\bbl@tempa{\zap@space#1 \@empty}%
2227   \bbl@for\bbl@tempb\bbl@tempa{%
2228     \if*\expandafter\car\bbl@tempb\@nil
2229       \edef\bbl@tempb{user@\expandafter@gobble\bbl@tempb}%
2230       \@expandtwoargs
2231       \bbl@set@user@generic{\expandafter\string\car#2\@nil}\bbl@tempb
2232     \fi
2233     \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].

2234 \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".

2235 \def\aliasshorthand#1#2{%

```

```

2236 \bbl@ifshorthand{#2}%
2237 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2238 \ifx\document\@notprerr
2239 \notshorthand{#2}%
2240 \else
2241 \initiate@active@char{#2}%
2242 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2243 \csname active@char\string#1\endcsname
2244 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2245 \csname normal@char\string#1\endcsname
2246 \bbl@activate{#2}%
2247 \fi
2248 \fi}%
2249 {\bbl@error
2250 {Cannot declare a shorthand turned off (\string#2)}
2251 {Sorry, but you cannot use shorthands which have been\\%
2252 turned off in the package options}}}

```

\@notshorthand

```

2253 \def\@notshorthand#1{%
2254 \bbl@error{%
2255 The character '\string #1' should be made a shorthand character;\\%
2256 add the command \string\usesshorthands\string{#1\string} to
2257 the preamble.\\%
2258 I will ignore your instruction}%
2259 {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.

```

2260 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2261 \DeclareRobustCommand*\shorthandoff{%
2262 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2263 \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.

```

2264 \def\bbl@switch@sh#1#2{%
2265 \ifx#2\@nnil\else
2266 \bbl@ifunset{\bbl@active@\string#2}%
2267 {\bbl@error
2268 {I can't switch '\string#2' on or off--not a shorthand}%
2269 {This character is not a shorthand. Maybe you made\\%
2270 a typing mistake? I will ignore your instruction.}}%
2271 {\ifcase#1% off, on, off*
2272 \catcode`#212\relax
2273 \or
2274 \catcode`#2\active
2275 \bbl@ifunset{\bbl@shdef@\string#2}%
2276 {}%
2277 {\bbl@withactive{\expandafter\let\expandafter}#2%
2278 \csname bbl@shdef@\string#2\endcsname
2279 \bbl@csarg\let{shdef@\string#2}\relax}%
2280 \ifcase\bbl@activated\or

```

```

2281         \bbl@activate{#2}%
2282     \else
2283         \bbl@deactivate{#2}%
2284     \fi
2285 \or
2286     \bbl@ifunset{bbl@shdef@\string#2}%
2287     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}{#2}%
2288     }%
2289     \csname bbl@oricat@\string#2\endcsname
2290     \csname bbl@oridef@\string#2\endcsname
2291 \fi}%
2292 \bbl@afterfi\bbl@switch@sh#1%
2293 \fi}

```

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

```

2294 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2295 \def\bbl@putsh#1{%
2296     \bbl@ifunset{bbl@active@\string#1}%
2297     {\bbl@putsh@i#1\@empty\@nnil}%
2298     {\csname bbl@active@\string#1\endcsname}}
2299 \def\bbl@putsh@i#1#2\@nnil{%
2300     \csname\language@group @sh@\string#1@%
2301     \ifx\@empty#2\else\string#2\fi\endcsname}
2302 \ifx\bbl@opt@shorthands\@nnil\else
2303     \let\bbl@s@initiate@active@char\initiate@active@char
2304     \def\initiate@active@char#1{%
2305         \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2306     \let\bbl@s@switch@sh\bbl@switch@sh
2307     \def\bbl@switch@sh#1#2{%
2308         \ifx#2\@nnil\else
2309             \bbl@afterfi
2310             \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2311         \fi}
2312     \let\bbl@s@activate\bbl@activate
2313     \def\bbl@activate#1{%
2314         \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2315     \let\bbl@s@deactivate\bbl@deactivate
2316     \def\bbl@deactivate#1{%
2317         \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2318 \fi

```

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

```

2319 \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.

```

2320 \def\bbl@prim@s{%
2321     \prime\futurelet\@let@token\bbl@pr@m@s}
2322 \def\bbl@if@primes#1#2{%
2323     \ifx#1\@let@token
2324         \expandafter\@firstoftwo
2325     \else\ifx#2\@let@token
2326         \bbl@afterelse\expandafter\@firstoftwo
2327     \else
2328         \bbl@afterfi\expandafter\@secondoftwo
2329     \fi\fi}

```

```

2330 \begingroup
2331 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2332 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\'
2333 \lowercase{%
2334 \gdef\bbl@pr@m@s{%
2335 \bbl@if@primes""%
2336 \pr@@@s
2337 {\bbl@if@primes*^\pr@@@t\egroup}}
2338 \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).

```

2339 \initiate@active@char{~}
2340 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2341 \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.

```

2342 \expandafter\def\csname OT1dqpos\endcsname{127}
2343 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

2344 \ifx\f@encoding\undefined
2345 \def\f@encoding{OT1}
2346 \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.

```

2347 \bbl@trace{Language attributes}
2348 \newcommand\languageattribute[2]{%
2349 \def\bbl@tempc{#1}%
2350 \bbl@fixname\bbl@tempc
2351 \bbl@iflanguage\bbl@tempc{%
2352 \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.

```

2353 \ifx\bbl@known@attribs\undefined
2354 \in@false
2355 \else
2356 \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,%
2357 \fi
2358 \ifin@
2359 \bbl@warning{%
2360 You have more than once selected the attribute '##1'\%
2361 for language #1. Reported}%
2362 \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.

```

2363      \bbl@exp{%
2364        \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2365      \edef\bbl@tempa{\bbl@tempc-##1}%
2366      \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2367      {\csname\bbl@tempc @attr@##1\endcsname}%
2368      {\@attrerr{\bbl@tempc}{##1}}%
2369      \fi}}
2370 \@onlypreamble\languageattribute

```

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

```

2371 \newcommand*{\@attrerr}[2]{%
2372   \bbl@error
2373   {The attribute #2 is unknown for language #1.}%
2374   {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}`.

```

2375 \def\bbl@declare@ttribute#1#2#3{%
2376   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2377   \ifin@
2378     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2379   \fi
2380   \bbl@add@list\bbl@attributes{#1-#2}%
2381   \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

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

```

2382 \def\bbl@ifattributeset#1#2#3#4{%
2383   \ifx\bbl@known@attribs\undefined
2384     \in@false
2385   \else
2386     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2387   \fi
2388   \ifin@
2389     \bbl@afterelse#3%
2390   \else
2391     \bbl@afterfi#4%
2392   \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.

```

2393 \def\bbl@ifknown@ttrib#1#2{%
2394   \let\bbl@tempa\@secondoftwo
2395   \bbl@loopx\bbl@tempb{#2}{%
2396     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2397   \ifin@
2398     \let\bbl@tempa\@firstoftwo
2399   \else

```



```

2400 \fi}%
2401 \bbl@tempa}

```

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

```

2402 \def\bbl@clear@ttribs{%
2403 \ifx\bbl@attributes\undefined\else
2404 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2405 \expandafter\bbl@clear@ttrib\bbl@tempa.
2406 }%
2407 \let\bbl@attributes\undefined
2408 \fi}
2409 \def\bbl@clear@ttrib#1-#2.{%
2410 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2411 \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`

```

2412 \bbl@trace{Macros for saving definitions}
2413 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2414 \newcount\babel@savecnt
2415 \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.

```

2416 \def\babel@save#1{%
2417 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2418 \toks@\expandafter{\originalTeX\let#1=}%
2419 \bbl@exp{%
2420 \def\\originalTeX{\the\toks@<babel@number\babel@savecnt>\relax}}%
2421 \advance\babel@savecnt@one}
2422 \def\babel@savevariable#1{%
2423 \toks@\expandafter{\originalTeX #1=}%
2424 \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.

```

2425 \def\bbl@frenchspacing{%
2426 \ifnum\the\sffcode`.=\@m

```

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

```

2427 \let\bbl@nonfrenchspacing\relax
2428 \else
2429 \frenchspacing
2430 \let\bbl@nonfrenchspacing\nonfrenchspacing
2431 \fi}
2432 \let\bbl@nonfrenchspacing\nonfrenchspacing
2433 \let\bbl@elt\relax
2434 \edef\bbl@fs@chars{%
2435 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2436 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2437 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
2438 \def\bbl@pre@fs{%
2439 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
2440 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
2441 \def\bbl@post@fs{%
2442 \bbl@save@sfcodes
2443 \edef\bbl@tempa{\bbl@cl{frspc}}%
2444 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
2445 \if u\bbl@tempa % do nothing
2446 \else\if n\bbl@tempa % non french
2447 \def\bbl@elt##1##2##3{%
2448 \ifnum\sfcode`##1=##2\relax
2449 \babel@savevariable{\sfcode`##1}%
2450 \sfcode`##1=##3\relax
2451 \fi}%
2452 \bbl@fs@chars
2453 \else\if y\bbl@tempa % french
2454 \def\bbl@elt##1##2##3{%
2455 \ifnum\sfcode`##1=##3\relax
2456 \babel@savevariable{\sfcode`##1}%
2457 \sfcode`##1=##2\relax
2458 \fi}%
2459 \bbl@fs@chars
2460 \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.

```

2461 \bbl@trace{Short tags}
2462 \def\babeltags#1{%
2463 \edef\bbl@tempa{\zap@space#1 \@empty}%
2464 \def\bbl@tempb##1=##2\@{#}%
2465 \edef\bbl@tempc{%
2466 \noexpand\newcommand
2467 \expandafter\noexpand\csname ##1\endcsname{%
2468 \noexpand\protect
2469 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2470 \noexpand\newcommand
2471 \expandafter\noexpand\csname text##1\endcsname{%
2472 \noexpand\foreignlanguage{##2}}}
2473 \bbl@tempc}%
2474 \bbl@for\bbl@tempa\bbl@tempa{%
2475 \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.

```

2476 \bbl@trace{Hyphens}
2477 \@onlypreamble\babelhyphenation
2478 \AtEndOfPackage{%
2479   \newcommand\babelhyphenation[2][\@empty]{%
2480     \ifx\bbl@hyphenation@ \relax
2481       \let\bbl@hyphenation@\@empty
2482     \fi
2483     \ifx\bbl@hyphlist\@empty\else
2484       \bbl@warning{%
2485         You must not intermingle \string\selectlanguage\space and\\%
2486         \string\babelhyphenation\space or some exceptions will not\\%
2487         be taken into account. Reported}%
2488       \fi
2489       \ifx\@empty#1%
2490         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2491       \else
2492         \bbl@vforeach{#1}{%
2493           \def\bbl@tempa{##1}%
2494           \bbl@fixname\bbl@tempa
2495           \bbl@iflanguage\bbl@tempa{%
2496             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2497               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2498               {}%
2499               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2500               #2}}}%
2501         \fi}}

```

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

```

2502 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2503 \def\bbl@t@one{T1}
2504 \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`.

```

2505 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2506 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2507 \def\bbl@hyphen{%
2508   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2509 \def\bbl@hyphen@i#1#2{%
2510   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2511   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2512   {\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.

```

2513 \def\bbl@usehyphen#1{%
2514   \leavevmode
2515   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2516   \nobreak\hskip\z@skip}
2517 \def\bbl@usehyphen#1{%
2518   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2519 \def\bbl@hyphenchar{%
2520   \ifnum\hyphenchar\font=\m@ne
2521     \babe\nullhyphen
2522   \else
2523     \char\hyphenchar\font
2524   \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.

```

2525 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2526 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2527 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2528 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2529 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2530 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2531 \def\bbl@hy@repeat{%
2532   \bbl@usehyphen{%
2533     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2534 \def\bbl@hy@repeat{%
2535   \bbl@usehyphen{%
2536     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2537 \def\bbl@hy@empty{\hskip\z@skip}
2538 \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.

```

2539 \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.

```

2540 \bbl@trace{Multiencoding strings}
2541 \def\bbl@tglobal#1{\global\let#1#1}
2542 \def\bbl@recatcode#1{% TODO. Used only once?
2543   \@tempcnta="7F
2544   \def\bbl@tempa{%
2545     \ifnum\@tempcnta>"FF\else
2546       \catcode\@tempcnta=#1\relax
2547       \advance\@tempcnta\@ne
2548       \expandafter\bbl@tempa
2549     \fi}%
2550   \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).

```
2551 \ifpackagewith{babel}{nocase}%
2552   {\let\bbl@patchuclc\relax}%
2553   {\def\bbl@patchuclc{%
2554     \global\let\bbl@patchuclc\relax
2555     \g@addto@macro\@uclclist{\reserved@b\reserved@b\bbl@uclc}}%
2556     \gdef\bbl@uclc##1{%
2557       \let\bbl@encoded\bbl@encoded@uclc
2558       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2559       {##1}%
2560       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2561         \csname\language @bbl@uclc\endcsname}%
2562       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2563     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2564     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2565 <<(*More package options)>> ≡
2566 \DeclareOption{nocase}{}
2567 <</More package options>>
```

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

```
2568 <<(*More package options)>> ≡
2569 \let\bbl@opt@strings\@nnil % accept strings=value
2570 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2571 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2572 \def\BabelStringsDefault{generic}
2573 <</More package options>>
```

Main command This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

```
2574 \@onlypreamble\StartBabelCommands
2575 \def\StartBabelCommands{%
2576   \begingroup
2577   \bbl@recatcode{11}%
2578   <<Macros local to BabelCommands>>
2579   \def\bbl@provstring##1##2{%
2580     \providecommand##1{##2}%
2581     \bbl@tglobal##1}%
2582   \global\let\bbl@scafter\@empty
2583   \let\StartBabelCommands\bbl@startcmds
2584   \ifx\BabelLanguages\relax
2585     \let\BabelLanguages\CurrentOption
2586   \fi
2587   \begingroup
2588   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2589   \StartBabelCommands}
2590 \def\bbl@startcmds{%
2591   \ifx\bbl@screset\@nnil\else
2592     \bbl@usehooks{stopcommands}{}%
2593   \fi
```

```

2594 \endgroup
2595 \begingroup
2596 \@ifstar
2597 {\ifx\bbbl@opt@strings\@nnil
2598   \let\bbbl@opt@strings\BabelStringsDefault
2599   \fi
2600   \bbbl@startcmds@i}%
2601   \bbbl@startcmds@i}
2602 \def\bbbl@startcmds@i#1#2{%
2603   \edef\bbbl@L{\zap@space#1 \@empty}%
2604   \edef\bbbl@G{\zap@space#2 \@empty}%
2605   \bbbl@startcmds@ii}
2606 \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.

```

2607 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
2608   \let\SetString@gobbletwo
2609   \let\bbbl@stringdef@gobbletwo
2610   \let\AfterBabelCommands@gobble
2611   \ifx\@empty#1%
2612     \def\bbbl@sc@label{generic}%
2613     \def\bbbl@encstring##1##2{%
2614       \ProvideTextCommandDefault##1{##2}%
2615       \bbbl@toglobal##1%
2616       \expandafter\bbbl@toglobal\csname\string?\string##1\endcsname}%
2617     \let\bbbl@sctest\in@true
2618   \else
2619     \let\bbbl@sc@charset\space % <- zapped below
2620     \let\bbbl@sc@fontenc\space % <- " "
2621     \def\bbbl@tempa##1=##2\@nil{%
2622       \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }%
2623       \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
2624       \def\bbbl@tempa##1 ##2{% space -> comma
2625         ##1%
2626         \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
2627       \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
2628       \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
2629       \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
2630       \def\bbbl@encstring##1##2{%
2631         \bbbl@foreach\bbbl@sc@fontenc{%
2632           \bbbl@ifunset{T@###1}%
2633           {}%
2634           {\ProvideTextCommand##1{####1}{##2}%
2635             \bbbl@toglobal##1%
2636             \expandafter
2637             \bbbl@toglobal\csname###1\string##1\endcsname}}}%
2638       \def\bbbl@sctest{%
2639         \bbbl@xin@{\bbbl@opt@strings,}{,\bbbl@sc@label,\bbbl@sc@fontenc,}}%
2640     \fi
2641     \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
2642     \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded

```

```

2643 \let\AfterBabelCommands\bbl@aftercmds
2644 \let\SetString\bbl@setstring
2645 \let\bbl@stringdef\bbl@encstring
2646 \else % ie, strings=value
2647 \bbl@sctest
2648 \ifin@
2649 \let\AfterBabelCommands\bbl@aftercmds
2650 \let\SetString\bbl@setstring
2651 \let\bbl@stringdef\bbl@provstring
2652 \fi\fi\fi
2653 \bbl@scswitch
2654 \ifx\bbl@G\@empty
2655 \def\SetString##1##2{%
2656 \bbl@error{Missing group for string \string##1}%
2657 {You must assign strings to some category, typically\\%
2658 captions or extras, but you set none}}%
2659 \fi
2660 \ifx\@empty#1%
2661 \bbl@usehooks{defaultcommands}{}%
2662 \else
2663 \@expandtwoargs
2664 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2665 \fi}

```

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

```

2666 \def\bbl@forlang#1#2{%
2667 \bbl@for#1\bbl@L{%
2668 \bbl@xin@{, #1,}{, \BabelLanguages,}%
2669 \ifin@#2\relax\fi}}
2670 \def\bbl@scswitch{%
2671 \bbl@forlang\bbl@tempa{%
2672 \ifx\bbl@G\@empty\else
2673 \ifx\SetString\@gobbles\else
2674 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2675 \bbl@xin@{, \bbl@GL,}{, \bbl@screset,}%
2676 \ifin@\else
2677 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2678 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
2679 \fi
2680 \fi
2681 \fi}}
2682 \AtEndOfPackage{%
2683 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2684 \let\bbl@scswitch\relax}
2685 \onlypreamble\EndBabelCommands
2686 \def\EndBabelCommands{%
2687 \bbl@usehooks{stopcommands}{}%
2688 \endgroup
2689 \endgroup
2690 \bbl@scafter}
2691 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside `\StartBabelCommands`.

Strings The following macro is the actual definition of `\SetString` when it is “active”. First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like `\providescommand`). With the event `stringprocess` you can preprocess the string by manipulating the value of `\BabelString`. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

2692 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2693   \bbl@forlang\bbl@tempa{%
2694     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2695     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2696       {\bbl@exp{%
2697         \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2698       }%
2699     \def\BabelString{#2}%
2700     \bbl@usehooks{stringprocess}{}%
2701     \expandafter\bbl@stringdef
2702     \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`.

```

2703 \ifx\bbl@opt@strings\relax
2704   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2705   \bbl@patchuclc
2706   \let\bbl@encoded\relax
2707   \def\bbl@encoded@uclc#1{%
2708     \@inmathwarn#1%
2709     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2710       \expandafter\ifx\csname ?\string#1\endcsname\relax
2711         \TextSymbolUnavailable#1%
2712       \else
2713         \csname ?\string#1\endcsname
2714       \fi
2715     \else
2716       \csname\cf@encoding\string#1\endcsname
2717     \fi}
2718 \else
2719   \def\bbl@scset#1#2{\def#1{#2}}
2720 \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.

```

2721 <<*Macros local to BabelCommands>> ≡
2722 \def\SetStringLoop##1##2{%
2723   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2724   \count@\z@
2725   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2726     \advance\count@\@ne
2727     \toks@\expandafter{\bbl@tempa}%
2728     \bbl@exp{%
2729       \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2730       \count@=\the\count@\relax}}}%
2731 <</Macros local to BabelCommands>>
```

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

```

2732 \def\bbl@aftercmds#1{%
2733   \toks@\expandafter{\bbl@scafter#1}%
2734   \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.

```
2735 <<*Macros local to BabelCommands>> ≡
2736 \newcommand\SetCase[3][]{%
2737   \bbl@patchuclc
2738   \bbl@forlang\bbl@tempa{%
2739     \expandafter\bbl@encstring
2740     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2741     \expandafter\bbl@encstring
2742     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2743     \expandafter\bbl@encstring
2744     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2745 <</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.

```
2746 <<*Macros local to BabelCommands>> ≡
2747 \newcommand\SetHyphenMap[1]{%
2748   \bbl@forlang\bbl@tempa{%
2749     \expandafter\bbl@stringdef
2750     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2751 <</Macros local to BabelCommands>>
```

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

```
2752 \newcommand\BabelLower[2]{% one to one.
2753   \ifnum\lccode#1=#2\else
2754     \babel@savevariable{\lccode#1}%
2755     \lccode#1=#2\relax
2756   \fi}
2757 \newcommand\BabelLowerMM[4]{% many-to-many
2758   \@tempcnta=#1\relax
2759   \@tempcntb=#4\relax
2760   \def\bbl@tempa{%
2761     \ifnum\@tempcnta>#2\else
2762       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2763       \advance\@tempcnta#3\relax
2764       \advance\@tempcntb#3\relax
2765       \expandafter\bbl@tempa
2766     \fi}%
2767   \bbl@tempa}
2768 \newcommand\BabelLowerM0[4]{% many-to-one
2769   \@tempcnta=#1\relax
2770   \def\bbl@tempa{%
2771     \ifnum\@tempcnta>#2\else
2772       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2773       \advance\@tempcnta#3
2774       \expandafter\bbl@tempa
2775     \fi}%
2776   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2777 <<*More package options>> ≡
2778 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2779 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2780 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2781 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2782 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
```

2783 <</More package options>>

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

```
2784 \AtEndOfPackage{%
2785   \ifx\bbbl@opt@hyphenmap\undefined
2786     \bbbl@xin@{,}{\bbbl@language@opts}%
2787     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
2788   \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.

```
2789 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2790   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
2791 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
2792   \bbbl@trim@def\bbbl@tempa{#2}%
2793   \bbbl@xin@{.template}{\bbbl@tempa}%
2794   \ifin@
2795     \bbbl@ini@captions@template{#3}{#1}%
2796   \else
2797     \edef\bbbl@tempd{%
2798       \expandafter\expandafter\expandafter
2799       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2800     \bbbl@xin@
2801       {\expandafter\string\csname #2name\endcsname}%
2802     {\bbbl@tempd}%
2803   \ifin@ % Renew caption
2804     \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
2805   \ifin@
2806     \bbbl@exp{%
2807       \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2808       {\bbbl@scset\<#2name>\<#1#2name>}%
2809     }%
2810   \else % Old way converts to new way
2811     \bbbl@ifunset{#1#2name}%
2812     {\bbbl@exp{%
2813       \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2814     \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2815     {\def\<#2name>\<#1#2name>}}%
2816     {}%
2817   }%
2818   \fi
2819 \else
2820   \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2821   \ifin@ % New way
2822     \bbbl@exp{%
2823       \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
2824       \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2825       {\bbbl@scset\<#2name>\<#1#2name>}%
2826     }%
2827   \else % Old way, but defined in the new way
2828     \bbbl@exp{%
2829       \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2830     \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2831     {\def\<#2name>\<#1#2name>}}%
2832     {}%
2833   \fi%
2834 \fi
2835 \@namedef{#1#2name}{#3}%
```

```

2836 \toks@\expandafter{\bbl@captionslist}%
2837 \bbl@exp{\in@{<#2name>}{the\toks@}}%
2838 \ifin@else
2839 \bbl@exp{\bbl@add\bbl@captionslist{<#2name>}}%
2840 \bbl@tglobal\bbl@captionslist
2841 \fi
2842 \fi}
2843 % \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.

```

2844 \bbl@trace{Macros related to glyphs}
2845 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2846 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2847 \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.

```

2848 \def\save@sfc@q#1{\leavevmode
2849 \begingroup
2850 \edef\SF{\spacefactor\the\spacefactor}#1\SF
2851 \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.

```

2852 \ProvideTextCommand{\quotedblbase}{OT1}{%
2853 \save@sfc@q{\set@low@box{\textquotedblright\}}%
2854 \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.

```

2855 \ProvideTextCommandDefault{\quotedblbase}{%
2856 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2857 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2858 \save@sfc@q{\set@low@box{\textquoteright\}}%
2859 \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.

```

2860 \ProvideTextCommandDefault{\quotesinglbase}{%
2861 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2862 \ProvideTextCommand{\guillemetleft}{OT1}{%
2863 \ifmmode
2864 \ll
2865 \else

```

```

2866 \save@sf@q{\nobreak
2867 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2868 \fi}
2869 \ProvideTextCommand{\guillemetright}{OT1}{%
2870 \ifmmode
2871 \gg
2872 \else
2873 \save@sf@q{\nobreak
2874 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2875 \fi}
2876 \ProvideTextCommand{\guillemotleft}{OT1}{%
2877 \ifmmode
2878 \ll
2879 \else
2880 \save@sf@q{\nobreak
2881 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2882 \fi}
2883 \ProvideTextCommand{\guillemotright}{OT1}{%
2884 \ifmmode
2885 \gg
2886 \else
2887 \save@sf@q{\nobreak
2888 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2889 \fi}

```

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

```

2890 \ProvideTextCommandDefault{\guillemetleft}{%
2891 \UseTextSymbol{OT1}{\guillemetleft}}
2892 \ProvideTextCommandDefault{\guillemetright}{%
2893 \UseTextSymbol{OT1}{\guillemetright}}
2894 \ProvideTextCommandDefault{\guillemotleft}{%
2895 \UseTextSymbol{OT1}{\guillemotleft}}
2896 \ProvideTextCommandDefault{\guillemotright}{%
2897 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2898 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2899 \ifmmode
2900 <%
2901 \else
2902 \save@sf@q{\nobreak
2903 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2904 \fi}
2905 \ProvideTextCommand{\guilsinglright}{OT1}{%
2906 \ifmmode
2907 >%
2908 \else
2909 \save@sf@q{\nobreak
2910 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2911 \fi}

```

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

```

2912 \ProvideTextCommandDefault{\guilsinglleft}{%
2913 \UseTextSymbol{OT1}{\guilsinglleft}}
2914 \ProvideTextCommandDefault{\guilsinglright}{%
2915 \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.

```
2916 \DeclareTextCommand{\ij}{OT1}{%
2917   i\kern-0.02em\bbl@allowhyphens j}
2918 \DeclareTextCommand{\IJ}{OT1}{%
2919   I\kern-0.02em\bbl@allowhyphens J}
2920 \DeclareTextCommand{\ij}{T1}{\char188}
2921 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2922 \ProvideTextCommandDefault{\ij}{%
2923   \UseTextSymbol{OT1}{\ij}}
2924 \ProvideTextCommandDefault{\IJ}{%
2925   \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).

```
2926 \def\crrtic@{\hrule height0.1ex width0.3em}
2927 \def\crttic@{\hrule height0.1ex width0.33em}
2928 \def\ddj@{%
2929   \setbox0\hbox{d}\dimen@=\ht0
2930   \advance\dimen@1ex
2931   \dimen@.45\dimen@
2932   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2933   \advance\dimen@ii.5ex
2934   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2935 \def\DDJ@{%
2936   \setbox0\hbox{D}\dimen@=.55\ht0
2937   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2938   \advance\dimen@ii.15ex % correction for the dash position
2939   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2940   \dimen\thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2941   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2942 %
2943 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2944 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2945 \ProvideTextCommandDefault{\dj}{%
2946   \UseTextSymbol{OT1}{\dj}}
2947 \ProvideTextCommandDefault{\DJ}{%
2948   \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.

```
2949 \DeclareTextCommand{\SS}{OT1}{SS}
2950 \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 2951 \ProvideTextCommandDefault{\glq}{%
2952   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2953 \ProvideTextCommand{\grq}{T1}{%
2954   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2955 \ProvideTextCommand{\grq}{TU}{%
2956   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2957 \ProvideTextCommand{\grq}{OT1}{%
2958   \save@sf@q{\kern-.0125em
2959     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2960     \kern.07em\relax}}
2961 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2962 \ProvideTextCommandDefault{\glqq}{%
2963   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2964 \ProvideTextCommand{\grqq}{T1}{%
2965   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2966 \ProvideTextCommand{\grqq}{TU}{%
2967   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2968 \ProvideTextCommand{\grqq}{OT1}{%
2969   \save@sf@q{\kern-.07em
2970     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2971     \kern.07em\relax}}
2972 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2973 \ProvideTextCommandDefault{\flq}{%
2974   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2975 \ProvideTextCommandDefault{\frq}{%
2976   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2977 \ProvideTextCommandDefault{\flqq}{%
2978   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2979 \ProvideTextCommandDefault{\frqq}{%
2980   \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).

```
2981 \def\umlauthigh{%
2982   \def\bbl@umlauta##1{\leavevmode\bgroup%
2983     \expandafter\accent\csname\fontencoding dpos\endcsname
2984     ##1\bbl@allowhyphens\egroup}%
2985   \let\bbl@umlaute\bbl@umlauta}
2986 \def\umlautlow{%
2987   \def\bbl@umlauta{\protect\lower@umlaut}}
```

```

2988 \def\umlaute\lower@umlaute{\protect\lower@umlaute}}
2989 \def\bbl@umlaute{\protect\lower@umlaute}}
2990 \umlauthigh

```

`\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.

```

2991 \expandafter\ifx\csname U@D\endcsname\relax
2992 \csname newdimen\endcsname\U@D
2993 \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.

```

2994 \def\lower@umlaute#1{%
2995   \leavevmode\bggroup
2996     \U@D 1ex%
2997     {\setbox\z@\hbox{%
2998       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2999       \dimen@ -.45ex\advance\dimen@\ht\z@
3000       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
3001     \expandafter\accent\csname\fontencoding dqpos\endcsname
3002     \fontdimen5\font\U@D #1%
3003   \egroup}

```

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

```

3004 \AtBeginDocument{%
3005   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlaut{a}}%
3006   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
3007   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
3008   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
3009   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlaut{o}}%
3010   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlaut{u}}%
3011   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlaut{A}}%
3012   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
3013   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
3014   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlaut{O}}%
3015   \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`.

```

3016 \ifx\l@english\undefined
3017   \chardef\l@english\z@
3018 \fi
3019 % The following is used to cancel rules in ini files (see Amharic).
3020 \ifx\l@unhyphenated\undefined
3021   \newlanguage\l@unhyphenated
3022 \fi

```

9.13 Layout

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

```
3023 \bbl@trace{Bidi layout}
3024 \providecommand\IfBabelLayout[3]{#3}%
3025 \newcommand\BabelPatchSection[1]{%
3026   \@ifundefined{#1}{}{%
3027     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3028     \@namedef{#1}{%
3029       \@ifstar{\bbl@presec@s{#1}}%
3030       {\@dblarg{\bbl@presec@x{#1}}}}}%
3031 \def\bbl@presec@x#1[#2]#3{%
3032   \bbl@exp{%
3033     \\\select@language@x{\bbl@main@language}%
3034     \\\bbl@cs{sspre@#1}%
3035     \\\bbl@cs{ss@#1}%
3036     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
3037     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3038     \\\select@language@x{\language}}}%
3039 \def\bbl@presec@s#1#2{%
3040   \bbl@exp{%
3041     \\\select@language@x{\bbl@main@language}%
3042     \\\bbl@cs{sspre@#1}%
3043     \\\bbl@cs{ss@#1}*%
3044     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3045     \\\select@language@x{\language}}}%
3046 \IfBabelLayout{sectioning}%
3047   {\BabelPatchSection{part}%
3048    \BabelPatchSection{chapter}%
3049    \BabelPatchSection{section}%
3050    \BabelPatchSection{subsection}%
3051    \BabelPatchSection{subsubsection}%
3052    \BabelPatchSection{paragraph}%
3053    \BabelPatchSection{subparagraph}%
3054    \def\babel@toc#1{%
3055      \select@language@x{\bbl@main@language}}}%
3056 \IfBabelLayout{captions}%
3057   {\BabelPatchSection{caption}}}
```

9.14 Load engine specific macros

```
3058 \bbl@trace{Input engine specific macros}
3059 \ifcase\bbl@engine
3060   \input txtbabel.def
3061 \or
3062   \input luababel.def
3063 \or
3064   \input xebabel.def
3065 \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.

```
3066 \bbl@trace{Creating languages and reading ini files}
3067 \let\bbl@extend@ini\@gobble
3068 \newcommand\babelprovide[2][]{%
3069   \let\bbl@savelangname\language
```



```

3070 \edef\bbl@savelocaleid{\the\localeid}%
3071 % Set name and locale id
3072 \edef\languagenamename{#2}%
3073 \bbl{id@assign
3074 % Initialize keys
3075 \let\bbl@KVP@captions\@nil
3076 \let\bbl@KVP@date\@nil
3077 \let\bbl@KVP@import\@nil
3078 \let\bbl@KVP@main\@nil
3079 \let\bbl@KVP@script\@nil
3080 \let\bbl@KVP@language\@nil
3081 \let\bbl@KVP@hyphenrules\@nil
3082 \let\bbl@KVP@linebreaking\@nil
3083 \let\bbl@KVP@justification\@nil
3084 \let\bbl@KVP@mapfont\@nil
3085 \let\bbl@KVP@maparabic\@nil
3086 \let\bbl@KVP@mapdigits\@nil
3087 \let\bbl@KVP@intraspace\@nil
3088 \let\bbl@KVP@intrapenalty\@nil
3089 \let\bbl@KVP@onchar\@nil
3090 \let\bbl@KVP@transforms\@nil
3091 \global\let\bbl@release@transforms\@empty
3092 \let\bbl@KVP@alph\@nil
3093 \let\bbl@KVP@Alph\@nil
3094 \let\bbl@KVP@labels\@nil
3095 \bbl@csarg\let{KVP@labels*}\@nil
3096 \global\let\bbl@inidata\@empty
3097 \global\let\bbl@extend@ini\@gobble
3098 \gdef\bbl@key@list{;}%
3099 \bbl@forkv{#1}{% TODO - error handling
3100   \in@{/}{##1}%
3101   \ifin@
3102     \global\let\bbl@extend@ini\bbl@extend@ini@aux
3103     \bbl@renewinikey##1\@{##2}%
3104   \else
3105     \bbl@csarg\def{KVP@##1}{##2}%
3106   \fi}%
3107 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
3108 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel#2}\@n\@tw@}%
3109 % == init ==
3110 \ifx\bbl@screset\@undefined
3111   \bbl@ldfinit
3112 \fi
3113 % ==
3114 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3115 \ifcase\bbl@howloaded
3116   \let\bbl@lbkflag\@empty % new
3117 \else
3118   \ifx\bbl@KVP@hyphenrules\@nil\else
3119     \let\bbl@lbkflag\@empty
3120   \fi
3121   \ifx\bbl@KVP@import\@nil\else
3122     \let\bbl@lbkflag\@empty
3123   \fi
3124 \fi
3125 % == import, captions ==
3126 \ifx\bbl@KVP@import\@nil\else
3127   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
3128   {\ifx\bbl@initoload\relax

```

```

3129         \begingroup
3130         \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3131         \bb1@input@texini{#2}%
3132     \endgroup
3133     \else
3134         \xdef\bb1@KVP@import{\bb1@initoload}%
3135     \fi}%
3136 {}%
3137 \fi
3138 \ifx\bb1@KVP@captions\@nil
3139     \let\bb1@KVP@captions\bb1@KVP@import
3140 \fi
3141 % ==
3142 \ifx\bb1@KVP@transforms\@nil\else
3143     \bb1@replace\bb1@KVP@transforms{ },}%
3144 \fi
3145 % == Load ini ==
3146 \ifcase\bb1@howloaded
3147     \bb1@provide@new{#2}%
3148 \else
3149     \bb1@ifblank{#1}%
3150     {}% With \bb1@load@basic below
3151     {\bb1@provide@renew{#2}}%
3152 \fi
3153 % Post tasks
3154 % -----
3155 % == subsequent calls after the first provide for a locale ==
3156 \ifx\bb1@inidata\@empty\else
3157     \bb1@extend@ini{#2}%
3158 \fi
3159 % == ensure captions ==
3160 \ifx\bb1@KVP@captions\@nil\else
3161     \bb1@ifunset{\bb1@extracaps@#2}%
3162     {\bb1@exp{\labelensure[exclude=\\today]{#2}}}%
3163     {\toks@ \expandafter \expandafter \expandafter
3164      {\csname \bb1@extracaps@#2\endcsname}%
3165      \bb1@exp{\labelensure[exclude=\\today,include=\\the\toks@]{#2}}}%
3166     \bb1@ifunset{\bb1@ensure@\\language}%
3167     {\bb1@exp%
3168      \\DeclareRobustCommand\<\bb1@ensure@\\language>[1]{%
3169       \\foreignlanguage{\\language}%
3170       {####1}}}%
3171     {}%
3172     \bb1@exp{%
3173       \\bb1@tglobal\<\bb1@ensure@\\language>%
3174       \\bb1@tglobal\<\bb1@ensure@\\language\space>}%
3175 \fi
3176 % ==
3177 % At this point all parameters are defined if 'import'. Now we
3178 % execute some code depending on them. But what about if nothing was
3179 % imported? We just set the basic parameters, but still loading the
3180 % whole ini file.
3181 \bb1@load@basic{#2}%
3182 % == script, language ==
3183 % Override the values from ini or defines them
3184 \ifx\bb1@KVP@script\@nil\else
3185     \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
3186 \fi
3187 \ifx\bb1@KVP@language\@nil\else

```

```

3188 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
3189 \fi
3190 % == onchar ==
3191 \ifx\bbl@KVP@onchar\@nil\else
3192 \bbl@luahyphenate
3193 \directlua{
3194   if Babel.locale_mapped == nil then
3195     Babel.locale_mapped = true
3196     Babel.linebreaking.add_before(Babel.locale_map)
3197     Babel.loc_to_scr = {}
3198     Babel.chr_to_loc = Babel.chr_to_loc or {}
3199   end}%
3200 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3201 \ifin@
3202 \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
3203 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}}%
3204 \fi
3205 \bbl@exp{\bbl@add\bbl@starthyphens
3206   {\bbl@patterns@lua{\languagename}}}%
3207 % TODO - error/warning if no script
3208 \directlua{
3209   if Babel.script_blocks['\bbl@cl{sbc}'] then
3210     Babel.loc_to_scr[\the\localeid] =
3211       Babel.script_blocks['\bbl@cl{sbc}']
3212     Babel.locale_props[\the\localeid].lc = \the\localeid\space
3213     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
3214   end
3215 }%
3216 \fi
3217 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3218 \ifin@
3219 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
3220 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
3221 \directlua{
3222   if Babel.script_blocks['\bbl@cl{sbc}'] then
3223     Babel.loc_to_scr[\the\localeid] =
3224       Babel.script_blocks['\bbl@cl{sbc}']
3225   end}%
3226 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
3227 \AtBeginDocument{%
3228   \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3229   {\selectfont}}%
3230 \def\bbl@mapselect{%
3231   \let\bbl@mapselect\relax
3232   \edef\bbl@prefontid{\fontid\font}}%
3233 \def\bbl@mapdir##1{%
3234   {\def\languagename{##1}%
3235     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3236     \bbl@switchfont
3237     \directlua{
3238       Babel.locale_props[\the\csname bbl@id@##1\endcsname]
3239         [\bbl@prefontid] = \fontid\font\space}}}%
3240   \fi
3241   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
3242   \fi
3243   % TODO - catch non-valid values
3244 \fi
3245 % == mapfont ==
3246 % For bidi texts, to switch the font based on direction

```

```

3247 \ifx\bb1@KVP@mapfont\@nil\else
3248   \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}}}%
3249   {\bb1@error{Option '\bb1@KVP@mapfont' unknown for\%
3250     mapfont. Use 'direction'.%
3251     {See the manual for details.}}}%
3252   \bb1@ifunset{\bb1@lsys@\language}\bb1@provide@lsys{\language}}}%
3253   \bb1@ifunset{\bb1@wdir@\language}\bb1@provide@dirs{\language}}}%
3254   \ifx\bb1@mapselect\@undefined % TODO. See onchar
3255     \AtBeginDocument{%
3256       \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
3257       {\selectfont}}}%
3258     \def\bb1@mapselect{%
3259       \let\bb1@mapselect\relax
3260       \edef\bb1@prefontid{\fontid\font}}%
3261     \def\bb1@mapdir##1{%
3262       {\def\language{##1}%
3263       \let\bb1@ifrestoring\@firstoftwo % avoid font warning
3264       \bb1@switchfont
3265       \directlua{Babel.fontmap
3266         [\the\csname \bb1@wdir@##1\endcsname]%
3267         [\bb1@prefontid]=\fontid\font}}}%
3268     \fi
3269     \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%
3270   \fi
3271   % == Line breaking: intraspace, intrapenalty ==
3272   % For CJK, East Asian, Southeast Asian, if interspace in ini
3273   \ifx\bb1@KVP@intraspace\@nil\else % We can override the ini or set
3274     \bb1@csarg\edef{intsp@#2}{\bb1@KVP@intraspace}%
3275   \fi
3276   \bb1@provide@intraspace
3277   % == Line breaking: CJK quotes ==
3278   \ifcase\bb1@engine\or
3279     \bb1@xin@{/c}{/\bb1@cl{\lnbrk}}}%
3280   \ifin@
3281     \bb1@ifunset{\bb1@quote@\language}}}%
3282     {\directlua{
3283       Babel.locale_props[\the\localeid].cjk_quotes = {}
3284       local cs = 'op'
3285       for c in string.utfvalues(
3286         [[\csname \bb1@quote@\language\endcsname]]) do
3287         if Babel.cjk_characters[c].c == 'qu' then
3288           Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
3289         end
3290         cs = (cs == 'op') and 'cl' or 'op'
3291       end
3292     }}%
3293   \fi
3294   \fi
3295   % == Line breaking: justification ==
3296   \ifx\bb1@KVP@justification\@nil\else
3297     \let\bb1@KVP@linebreaking\bb1@KVP@justification
3298   \fi
3299   \ifx\bb1@KVP@linebreaking\@nil\else
3300     \bb1@xin@{,\bb1@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3301   \ifin@
3302     \bb1@csarg\xdef
3303       {\lnbrk@\language}{\expandafter\@car\bb1@KVP@linebreaking\@nil}%
3304   \fi
3305   \fi

```

```

3306 \bbl@xin@{/e}{/\bbl@c1{lbrk}}%
3307 \ifin@else\bbl@xin@{/k}{/\bbl@c1{lbrk}}\fi
3308 \ifin@bbl@arabicjust\fi
3309 % == Line breaking: hyphenate.other.(locale|script) ==
3310 \ifx\bbl@lbrkflag\empty
3311 \bbl@ifunset{bbl@hyotl@language}{}%
3312 {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}}%
3313 \bbl@startcommands*\language{}%
3314 \bbl@csarg\bbl@foreach{hyotl@language}{%
3315 \ifcase\bbl@engine
3316 \ifnum##1<257
3317 \SetHyphenMap{\BabelLower{##1}{##1}}%
3318 \fi
3319 \else
3320 \SetHyphenMap{\BabelLower{##1}{##1}}%
3321 \fi}%
3322 \bbl@endcommands}%
3323 \bbl@ifunset{bbl@hyots@language}{}%
3324 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}}%
3325 \bbl@csarg\bbl@foreach{hyots@language}{%
3326 \ifcase\bbl@engine
3327 \ifnum##1<257
3328 \global\lccode##1=##1\relax
3329 \fi
3330 \else
3331 \global\lccode##1=##1\relax
3332 \fi}}%
3333 \fi
3334 % == Counters: maparabic ==
3335 % Native digits, if provided in ini (TeX level, xe and lua)
3336 \ifcase\bbl@engine\else
3337 \bbl@ifunset{bbl@dgnat@language}{}%
3338 {\expandafter\ifx\csname bbl@dgnat@language\endcsname\empty\else
3339 \expandafter\expandafter\expandafter
3340 \bbl@setdigits\csname bbl@dgnat@language\endcsname
3341 \ifx\bbl@KVP@maparabic\@nil\else
3342 \ifx\bbl@latinarabic\@undefined
3343 \expandafter\let\expandafter\@arabic
3344 \csname bbl@counter@language\endcsname
3345 \else % ie, if layout=counters, which redefines \@arabic
3346 \expandafter\let\expandafter\bbl@latinarabic
3347 \csname bbl@counter@language\endcsname
3348 \fi
3349 \fi
3350 \fi}%
3351 \fi
3352 % == Counters: mapdigits ==
3353 % Native digits (lua level).
3354 \ifodd\bbl@engine
3355 \ifx\bbl@KVP@mapdigits\@nil\else
3356 \bbl@ifunset{bbl@dgnat@language}{}%
3357 {\RequirePackage{luatexbase}%
3358 \bbl@activate@preotf
3359 \directlua{
3360 Babel = Babel or {} %% -> presets in luababel
3361 Babel.digits_mapped = true
3362 Babel.digits = Babel.digits or {}
3363 Babel.digits[\the\localeid] =
3364 table.pack(string.utfvalue('\bbl@c1{dgnat}'))

```

```

3365         if not Babel.numbers then
3366             function Babel.numbers(head)
3367                 local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3368                 local GLYPH = node.id'glyph'
3369                 local inmath = false
3370                 for item in node.traverse(head) do
3371                     if not inmath and item.id == GLYPH then
3372                         local temp = node.get_attribute(item, LOCALE)
3373                         if Babel.digits[temp] then
3374                             local chr = item.char
3375                             if chr > 47 and chr < 58 then
3376                                 item.char = Babel.digits[temp][chr-47]
3377                             end
3378                         end
3379                     elseif item.id == node.id'math' then
3380                         inmath = (item.subtype == 0)
3381                     end
3382                 end
3383                 return head
3384             end
3385         end
3386     } }%
3387 \fi
3388 \fi
3389 % == Counters: alph, Alph ==
3390 % What if extras<lang> contains a \babel@save\@alph? It won't be
3391 % restored correctly when exiting the language, so we ignore
3392 % this change with the \bbl@alph@saved trick.
3393 \ifx\bbl@KVP@alph\@nil\else
3394     \bbl@extras@wrap{\\bbl@alph@saved}%
3395     {\let\bbl@alph@saved\@alph}%
3396     {\let\@alph\bbl@alph@saved
3397     \babel@save\@alph}%
3398     \bbl@exp{%
3399         \\bbl@add\<extras\language\name>{%
3400             \let\\@alph<bbl@cntr@bbl@KVP@alph @\language\name>}}%
3401 \fi
3402 \ifx\bbl@KVP@Alph\@nil\else
3403     \bbl@extras@wrap{\\bbl@Alph@saved}%
3404     {\let\bbl@Alph@saved\@Alph}%
3405     {\let\@Alph\bbl@Alph@saved
3406     \babel@save\@Alph}%
3407     \bbl@exp{%
3408         \\bbl@add\<extras\language\name>{%
3409             \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language\name>}}%
3410 \fi
3411 % == require.babel in ini ==
3412 % To load or reload the babel-*.tex, if require.babel in ini
3413 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3414     \bbl@ifunset{bbl@rqtex@\language\name}{}%
3415     {\xdef\babel@beforestart{
3416         \let\BabelBeforeIni\@gobbletwo
3417         \chardef\atcatcode=\catcode`\@
3418         \catcode`\@=11\relax
3419         \bbl@input@texini{\bbl@cs{rqtex@\language\name}}%
3420         \catcode`\@=\atcatcode
3421         \let\atcatcode\relax
3422         \global\bbl@csarg\let{rqtex@\language\name}\relax
3423     }}%

```

```

3424 \fi
3425 % == frenchspacing ==
3426 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
3427 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
3428 \ifin@
3429 \bbbl@extras@wrap{\bbbl@pre@fs}%
3430 {\bbbl@pre@fs}%
3431 {\bbbl@post@fs}%
3432 \fi
3433 % == Release saved transforms ==
3434 \bbbl@release@transforms\relax % \relax closes the last item.
3435 % == main ==
3436 \ifx\bbbl@KVP@main\@nil % Restore only if not 'main'
3437 \let\language\bbbl@savelangname
3438 \chardef\localeid\bbbl@savelocaleid\relax
3439 \fi}

```

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

```

3440 \def\bbbl@provide@new#1{%
3441 \namedef{date#1}}% marks lang exists - required by \StartBabelCommands
3442 \namedef{extras#1}}%
3443 \namedef{noextras#1}}%
3444 \bbbl@startcommands*{#1}{captions}%
3445 \ifx\bbbl@KVP@captions\@nil % and also if import, implicit
3446 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
3447 \ifx##1\@empty\else
3448 \bbbl@exp{%
3449 \SetString\##1{%
3450 \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
3451 \expandafter\bbbl@tempb
3452 \fi}%
3453 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
3454 \else
3455 \ifx\bbbl@initoload\relax
3456 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
3457 \else
3458 \bbbl@read@ini{\bbbl@initoload}2% % Same
3459 \fi
3460 \fi
3461 \StartBabelCommands*{#1}{date}%
3462 \ifx\bbbl@KVP@import\@nil
3463 \bbbl@exp{%
3464 \SetString\today{\bbbl@nocaption{today}{#1today}}}%
3465 \else
3466 \bbbl@savetoday
3467 \bbbl@savestate
3468 \fi
3469 \bbbl@endcommands
3470 \bbbl@load@basic{#1}%
3471 % == hyphenmins == (only if new)
3472 \bbbl@exp{%
3473 \gdef\<#1hyphenmins>{%
3474 {\bbbl@ifunset{\bbbl@lftm#1}{2}{\bbbl@cs{lftm#1}}}%
3475 {\bbbl@ifunset{\bbbl@rgtm#1}{3}{\bbbl@cs{rgtm#1}}}%
3476 % == hyphenrules (also in renew) ==
3477 \bbbl@provide@hyphens{#1}%
3478 \ifx\bbbl@KVP@main\@nil\else
3479 \expandafter\main@language\expandafter{#1}%

```

```

3480 \fi}
3481 %
3482 \def\bbbl@provide@renew#1{%
3483 \ifx\bbbl@KVP@captions\@nil\else
3484 \StartBabelCommands*{#1}{captions}%
3485 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
3486 \EndBabelCommands
3487 \fi
3488 \ifx\bbbl@KVP@import\@nil\else
3489 \StartBabelCommands*{#1}{date}%
3490 \bbbl@savetoday
3491 \bbbl@savedate
3492 \EndBabelCommands
3493 \fi
3494 % == hyphenrules (also in new) ==
3495 \ifx\bbbl@lbfkflag\@empty
3496 \bbbl@provide@hyphens{#1}%
3497 \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.)

```

3498 \def\bbbl@load@basic#1{%
3499 \ifcase\bbbl@howloaded\or\or
3500 \ifcase\csname bbl@llevel@\language\endcsname
3501 \bbbl@csarg\let{lname@\language}\relax
3502 \fi
3503 \fi
3504 \bbbl@ifunset{\bbbl@lname@#1}%
3505 {\def\BabelBeforeIni##1##2{%
3506 \begingroup
3507 \let\bbbl@ini@captions@aux\@gobbletwo
3508 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3509 \bbbl@read@ini{##1}1%
3510 \ifx\bbbl@initoload\relax\endinput\fi
3511 \endgroup}%
3512 \begingroup % boxed, to avoid extra spaces:
3513 \ifx\bbbl@initoload\relax
3514 \bbbl@input@texini{##1}%
3515 \else
3516 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
3517 \fi
3518 \endgroup}%
3519 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3520 \def\bbbl@provide@hyphens#1{%
3521 \let\bbbl@tempa\relax
3522 \ifx\bbbl@KVP@hyphenrules\@nil\else
3523 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3524 \bbbl@foreach\bbbl@KVP@hyphenrules{%
3525 \ifx\bbbl@tempa\relax % if not yet found
3526 \bbbl@ifsamestring{##1}{+}%
3527 {{\bbbl@exp{\addlanguage\<l@##1>}}}%
3528 {}%
3529 \bbbl@ifunset{l@##1}%
3530 {}%
3531 {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}}%
3532 \fi}%

```



```

3533 \fi
3534 \ifx\bbbl@tempa\relax %           if no opt or no language in opt found
3535   \ifx\bbbl@KVP@import\@nil
3536     \ifx\bbbl@initoload\relax\else
3537       \bbbl@exp{%                   and hyphenrules is not empty
3538         \\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3539         }%
3540         {\let\\bbbl@tempa\<l@\bbbl@cl{hyphr}>}}%
3541   \fi
3542 \else % if importing
3543   \bbbl@exp{%                       and hyphenrules is not empty
3544     \\bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3545     }%
3546     {\let\\bbbl@tempa\<l@\bbbl@cl{hyphr}>}}%
3547 \fi
3548 \fi
3549 \bbbl@ifunset{\bbbl@tempa}%         ie, relax or undefined
3550 {\bbbl@ifunset{l@#1}%              no hyphenrules found - fallback
3551   {\bbbl@exp{\\adddialect\<l@#1>\language}}%
3552   }%                                so, l@<lang> is ok - nothing to do
3553 {\bbbl@exp{\\adddialect\<l@#1>\bbbl@tempa}}% found in opt list or ini

```

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

```

3554 \def\bbbl@input@texini#1{%
3555   \bbbl@bsphack
3556   \bbbl@exp{%
3557     \catcode\\% =14 \catcode\\% =0
3558     \catcode\\% =1 \catcode\\% =2
3559     \lowercase{\\InputIfFileExists{babel-#1.tex}{}}%
3560     \catcode\\% =\the\catcode\\% \relax
3561     \catcode\\% =\the\catcode\\% \relax
3562     \catcode\\% =\the\catcode\\% \relax
3563     \catcode\\% =\the\catcode\\% \relax}%
3564   \bbbl@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 \bbbl@read@ini.

```

3565 \def\bbbl@inline#1\bbbl@inline{%
3566   \@ifnextchar[\bbbl@inisect{\@ifnextchar;\bbbl@iniskip\bbbl@inistore}#1\@@}% ]
3567 \def\bbbl@inisect[#1]#2\@@{\def\bbbl@section{#1}}
3568 \def\bbbl@iniskip#1\@@{%           if starts with ;
3569 \def\bbbl@inistore#1=#2\@@{%       full (default)
3570   \bbbl@trim@def\bbbl@tempa{#1}%
3571   \bbbl@trim\toks@{#2}%
3572   \bbbl@xin@{\bbbl@section/\bbbl@tempa}{\bbbl@key@list}%
3573   \ifin@else
3574     \bbbl@exp{%
3575       \\g@addto@macro\\bbbl@inidata{%
3576         \\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3577   \fi}
3578 \def\bbbl@inistore@min#1=#2\@@{%   minimal (maybe set in \bbbl@read@ini)
3579   \bbbl@trim@def\bbbl@tempa{#1}%
3580   \bbbl@trim\toks@{#2}%
3581   \bbbl@xin@{.identification.}{.\bbbl@section.}%
3582   \ifin@
3583     \bbbl@exp{\\g@addto@macro\\bbbl@inidata{%
3584       \\bbbl@elt{identification}{\bbbl@tempa}{\the\toks@}}}%
3585   \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.

```

3586 \ifx\bbl@readstream\undefined
3587 \csname newread\endcsname\bbl@readstream
3588 \fi
3589 \def\bbl@read@ini#1#2{%
3590   \global\let\bbl@extend@ini\@gobble
3591   \openin\bbl@readstream=babel-#1.ini
3592   \ifeof\bbl@readstream
3593     \bbl@error
3594     {There is no ini file for the requested language\%
3595      (#1). Perhaps you misspelled it or your installation\%
3596      is not complete.}%
3597     {Fix the name or reinstall babel.}%
3598   \else
3599     % == Store ini data in \bbl@inidata ==
3600     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3601     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3602     \bbl@info{Importing
3603               \ifcase#2font and identification \or basic \fi
3604               data for \language\%
3605               from babel-#1.ini. Reported}%
3606     \ifnum#2=\z@
3607       \global\let\bbl@inidata\@empty
3608       \let\bbl@inistore\bbl@inistore@min % Remember it's local
3609     \fi
3610     \def\bbl@section{identification}%
3611     \bbl@exp{\bbl@inistore tag.ini=#1\\@}%
3612     \bbl@inistore load.level=#2\\@@
3613     \loop
3614     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3615       \endlinechar\m@ne
3616       \read\bbl@readstream to \bbl@line
3617       \endlinechar\^^M
3618       \ifx\bbl@line\@empty\else
3619         \expandafter\bbl@iniline\bbl@line\bbl@iniline
3620       \fi
3621     \repeat
3622     % == Process stored data ==
3623     \bbl@csarg\xdef{lini@\language}{#1}%
3624     \bbl@read@ini@aux
3625     % == 'Export' data ==
3626     \bbl@ini@exports{#2}%
3627     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
3628     \global\let\bbl@inidata\@empty
3629     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
3630     \bbl@tglobal\bbl@ini@loaded
3631   \fi}
3632 \def\bbl@read@ini@aux{%
3633   \let\bbl@savestrings\@empty
3634   \let\bbl@savetoday\@empty
3635   \let\bbl@savestate\@empty
3636   \def\bbl@elt##1##2##3{%
3637     \def\bbl@section{##1}%

```

```

3638 \in@{=date.}{=##1}% Find a better place
3639 \ifin@
3640 \bbl@ini@calendar{##1}%
3641 \fi
3642 \bbl@ifunset{bbl@inikv@##1}{}%
3643 {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
3644 \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.

```

3645 \def\bbl@extend@ini@aux#1{%
3646 \bbl@startcommands*{#1}{captions}%
3647 % Activate captions/... and modify exports
3648 \bbl@csarg\def{inikv@captions.licr}##1##2{%
3649 \setlocalecaption{#1}{##1}{##2}}%
3650 \def\bbl@inikv@captions##1##2{%
3651 \bbl@ini@captions@aux{##1}{##2}}%
3652 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3653 \def\bbl@exportkey##1##2##3{%
3654 \bbl@ifunset{bbl@kv@##2}{}%
3655 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
3656 \bbl@exp{\global\let<bbl@##1@languagename>\<bbl@kv@##2>}}%
3657 \fi}}%
3658 % As with \bbl@read@ini, but with some changes
3659 \bbl@read@ini@aux
3660 \bbl@ini@exports\tw@
3661 % Update inidata@lang by pretending the ini is read.
3662 \def\bbl@elt##1##2##3{%
3663 \def\bbl@section{##1}%
3664 \bbl@iniline##2=##3\bbl@iniline}%
3665 \csname bbl@inidata@#1\endcsname
3666 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
3667 \StartBabelCommands*{#1}{date}% And from the import stuff
3668 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3669 \bbl@savetoday
3670 \bbl@savestate
3671 \bbl@endcommands}

```

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

```

3672 \def\bbl@ini@calendar#1{%
3673 \lowercase{\def\bbl@tempa{=#1=}}%
3674 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3675 \bbl@replace\bbl@tempa{=date.}{}%
3676 \in@{.licr=}{#1=}%
3677 \ifin@
3678 \ifcase\bbl@engine
3679 \bbl@replace\bbl@tempa{.licr=}{}%
3680 \else
3681 \let\bbl@tempa\relax
3682 \fi
3683 \fi
3684 \ifx\bbl@tempa\relax\else
3685 \bbl@replace\bbl@tempa{=}{}%
3686 \bbl@exp{%
3687 \def<bbl@inikv@#1>####1####2{%
3688 \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
3689 \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).

```

3690 \def\bbl@renewinikey#1/#2\@#3{%
3691   \edef\bbl@tempa{\zap@space #1 \@empty}%    section
3692   \edef\bbl@tempb{\zap@space #2 \@empty}%    key
3693   \bbl@trim\toks@{#3}%                        value
3694   \bbl@exp{%
3695     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
3696     \\g@addto@macro\\bbl@inidata{%
3697       \\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.

```

3698 \def\bbl@exportkey#1#2#3{%
3699   \bbl@ifunset{bbl@kv@#2}%
3700   {\bbl@csarg\gdef{#1@\language}\@empty}%
3701   {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3702     \bbl@csarg\gdef{#1@\language}\@empty}%
3703   \else
3704     \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3705     \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.

```

3706 \def\bbl@iniwarning#1{%
3707   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3708   {\bbl@warning{%
3709     From babel-\bbl@cs{lini@\language}.ini:\%
3710     \bbl@cs{kv@identification.warning#1}\%
3711     Reported }}}
3712 %
3713 \let\bbl@release@transforms\@empty
3714 %
3715 \def\bbl@ini@exports#1{%
3716   % Identification always exported
3717   \bbl@iniwarning{%
3718     \ifcase\bbl@engine
3719       \bbl@iniwarning{.pdflatex}%
3720     \or
3721       \bbl@iniwarning{.lualatex}%
3722     \or
3723       \bbl@iniwarning{.xelatex}%
3724     \fi%
3725     \bbl@exportkey{llevel}{identification.load.level}{}%
3726     \bbl@exportkey{elname}{identification.name.english}{}%
3727     \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
3728       {\csname bbl@elname@\language\endcsname}}%
3729     \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
3730     \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
3731     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3732     \bbl@exportkey{esname}{identification.script.name}{}%
3733     \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
3734       {\csname bbl@esname@\language\endcsname}}%
3735     \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3736     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3737     % Also maps bcp47 -> language
3738     \ifbbl@bcptoname
3739     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcpl}}{\language}%

```

```

3740 \fi
3741 % Conditional
3742 \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
3743 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3744 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3745 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3746 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3747 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3748 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3749 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3750 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3751 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3752 \bbl@exportkey{chrng}{characters.ranges}{}%
3753 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3754 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3755 \ifnum#1=\tw@          % only (re)new
3756 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3757 \bbl@tglobal\bbl@savetoday
3758 \bbl@tglobal\bbl@savestate
3759 \bbl@savestrings
3760 \fi
3761 \fi}

```

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

```

3762 \def\bbl@inikv#1#2{%      key=value
3763 \toks@{#2}%              This hides #'s from ini values
3764 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3765 \let\bbl@inikv@identification\bbl@inikv
3766 \let\bbl@inikv@typography\bbl@inikv
3767 \let\bbl@inikv@characters\bbl@inikv
3768 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3769 \def\bbl@inikv@counters#1#2{%
3770 \bbl@ifsamestring{#1}{digits}%
3771 {\bbl@error{The counter name 'digits' is reserved for mapping\\
3772 decimal digits}%
3773 {Use another name.}}%
3774 }%
3775 \def\bbl@tempc{#1}%
3776 \bbl@trim@def{\bbl@tempb*}{#2}%
3777 \in@{.1$}{#1$}%
3778 \ifin@
3779 \bbl@replace\bbl@tempc{.1}{}%
3780 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3781 \noexpand\bbl@alphanumeric{\bbl@tempc}%
3782 \fi
3783 \in@{.F.}{#1}%
3784 \ifin@else\in@{.S.}{#1}\fi
3785 \ifin@
3786 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3787 \else
3788 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3789 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3790 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3791 \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.

```

3792 \ifcase\bbl@engine
3793   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3794     \bbl@ini@captions@aux{#1}{#2}}
3795 \else
3796   \def\bbl@inikv@captions#1#2{%
3797     \bbl@ini@captions@aux{#1}{#2}}
3798 \fi

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

3799 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3800   \bbl@replace\bbl@tempa{.template}{}}%
3801   \def\bbl@toreplace{#1}{}}%
3802   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3803   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3804   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3805   \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}%
3806   \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3807   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3808   \ifin@
3809     \@nameuse{bbl@patch\bbl@tempa}%
3810     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3811   \fi
3812   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3813   \ifin@
3814     \toks@{\expandafter{\bbl@toreplace}%
3815     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}}%
3816   \fi}
3817 \def\bbl@ini@captions@aux#1#2{%
3818   \bbl@trim@def\bbl@tempa{#1}%
3819   \bbl@xin@{.template}{\bbl@tempa}%
3820   \ifin@
3821     \bbl@ini@captions@template{#2}\languagename
3822   \else
3823     \bbl@ifblank{#2}%
3824     {\bbl@exp{%
3825       \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3826     {\bbl@trim\toks@{#2}}}%
3827     \bbl@exp{%
3828       \bbl@add\bbl@savestrings{%
3829         \SetString\<\bbl@tempa name>{\the\toks@}}}%
3830     \toks@\expandafter{\bbl@captionslist}%
3831     \bbl@exp{\in{\<\bbl@tempa name>}{\the\toks@}}%
3832     \ifin@else
3833       \bbl@exp{%
3834         \bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3835         \bbl@toglobal\<bbl@extracaps@\languagename>}%
3836     \fi
3837   \fi}

```

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

```

3838 \def\bbl@list@the{%
3839   part,chapter,section,subsection,subsubsection,paragraph,%
3840   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3841   table,page,footnote,mpfootnote,mpfn}
3842 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3843   \bbl@ifunset{bbl@map@#1@\languagename}%

```

```

3844 {\@nameuse{#1}}%
3845 {\@nameuse{bbl@map@#1@\languagename}}%
3846 \def\bbl@inikv@labels#1#2{%
3847 \in@{.map}{#1}%
3848 \ifin@
3849 \ifx\bbl@KVP@labels\@nil\else
3850 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3851 \ifin@
3852 \def\bbl@tempc{#1}%
3853 \bbl@replace\bbl@tempc{.map}{}%
3854 \in@{, #2, }{, arabic, roman, Roman, alph, Alph, fnsymbol,}%
3855 \bbl@exp{%
3856 \gdef\<bbl@map@\bbl@tempc @\languagename>%
3857 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3858 \bbl@foreach\bbl@list@the{%
3859 \bbl@ifunset{the##1}{}%
3860 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3861 \bbl@exp{%
3862 \\bbl@sreplace\<the##1>%
3863 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3864 \\bbl@sreplace\<the##1>%
3865 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3866 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3867 \toks@ \expandafter\expandafter\expandafter{%
3868 \csname the##1\endcsname}%
3869 \expandafter\def\csname the##1\endcsname{{\the\toks@}}%
3870 \fi}}%
3871 \fi
3872 \fi
3873 %
3874 \else
3875 %
3876 % The following code is still under study. You can test it and make
3877 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3878 % language dependent.
3879 \in@{enumerate.}{#1}%
3880 \ifin@
3881 \def\bbl@tempa{#1}%
3882 \bbl@replace\bbl@tempa{enumerate.}{}%
3883 \def\bbl@toreplace{#2}%
3884 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3885 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3886 \bbl@replace\bbl@toreplace{}{\endcsname{}}%
3887 \toks@ \expandafter{\bbl@toreplace}%
3888 % TODO. Execute only once:
3889 \bbl@exp{%
3890 \\bbl@add\<extras\languagename>{%
3891 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3892 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3893 \\bbl@toggle\<extras\languagename>}%
3894 \fi
3895 \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.

```

3896 \def\bbl@chapttype{chapter}
3897 \ifx\@makechapterhead\@undefined

```

```

3898 \let\bbl@patchchapter\relax
3899 \else\ifx\thechapter\@undefined
3900 \let\bbl@patchchapter\relax
3901 \else\ifx\ps@headings\@undefined
3902 \let\bbl@patchchapter\relax
3903 \else
3904 \def\bbl@patchchapter{%
3905 \global\let\bbl@patchchapter\relax
3906 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3907 \bbl@tglobal\appendix
3908 \bbl@sreplace\ps@headings
3909 {\@chapapp\ \thechapter}%
3910 {\bbl@chapterformat}%
3911 \bbl@tglobal\ps@headings
3912 \bbl@sreplace\chaptermark
3913 {\@chapapp\ \thechapter}%
3914 {\bbl@chapterformat}%
3915 \bbl@tglobal\chaptermark
3916 \bbl@sreplace\@makechapterhead
3917 {\@chapapp\space\thechapter}%
3918 {\bbl@chapterformat}%
3919 \bbl@tglobal\@makechapterhead
3920 \gdef\bbl@chapterformat{%
3921 \bbl@ifunset{bbl@\bbl@chapttype fmt@\language}%
3922 {\@chapapp\space\thechapter}
3923 {\@nameuse{bbl@\bbl@chapttype fmt@\language}}}}
3924 \let\bbl@patchappendix\bbl@patchchapter
3925 \fi\fi\fi
3926 \ifx\@part\@undefined
3927 \let\bbl@patchpart\relax
3928 \else
3929 \def\bbl@patchpart{%
3930 \global\let\bbl@patchpart\relax
3931 \bbl@sreplace\@part
3932 {\partname\nobreakspace\thepart}%
3933 {\bbl@partformat}%
3934 \bbl@tglobal\@part
3935 \gdef\bbl@partformat{%
3936 \bbl@ifunset{bbl@partfmt@\language}%
3937 {\partname\nobreakspace\thepart}
3938 {\@nameuse{bbl@partfmt@\language}}}}
3939 \fi

```

Date. TODO. Document

```

3940 % Arguments are _not_ protected.
3941 \let\bbl@calendar\@empty
3942 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3943 \def\bbl@localedate#1#2#3#4{%
3944 \begingroup
3945 \ifx\@empty#1\@empty\else
3946 \let\bbl@ld@calendar\@empty
3947 \let\bbl@ld@variant\@empty
3948 \edef\bbl@tempa{\zap@space#1 \@empty}%
3949 \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld@##1}{##2}}%
3950 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3951 \edef\bbl@calendar{%
3952 \bbl@ld@calendar
3953 \ifx\bbl@ld@variant\@empty\else
3954 .\bbl@ld@variant

```



```

3955     \fi}%
3956     \bbl@replace\bbl@calendar{gregorian}{}%
3957     \fi
3958     \bbl@cased
3959     {\@nameuse\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3960 \endgroup}
3961 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3962 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3963   \bbl@trim@def\bbl@tempa{#1.#2}%
3964   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3965   {\bbl@trim@def\bbl@tempa{#3}%
3966     \bbl@trim\toks@{#5}%
3967     \@temptokena\expandafter{\bbl@savestate}%
3968     \bbl@exp{% Reverse order - in ini last wins
3969       \def\\bbl@savestate{%
3970         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3971         \the\@temptokena}}}%
3972   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3973     {\lowercase{\def\bbl@tempb{#6}}}%
3974     \bbl@trim@def\bbl@toreplace{#5}%
3975     \bbl@TG@@date
3976     \bbl@ifunset\bbl@date@\language @}%
3977     {\bbl@exp{% TODO. Move to a better place.
3978       \gdef\<\language date>{\protect\<\language date >}%
3979       \gdef\<\language date >####1####2####3{%
3980         \\bbl@usedategrouptue
3981         \<bbl@ensure@\language >{%
3982           \\localedate{####1}{####2}{####3}}}%
3983       \\bbl@add\\bbl@savetoday{%
3984         \\SetString\\today{%
3985           \<\language date>%
3986           {\the\year}{\the\month}{\the\day}}}%
3987       }%
3988       \global\bbl@csarg\let{date@\language @}\bbl@toreplace
3989       \ifx\bbl@tempb\@empty\else
3990         \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3991       \fi}%
3992     {}}}

```

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.

```

3993 \let\bbl@calendar\@empty
3994 \newcommand\BabelDateSpace{\nobreakspace}
3995 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3996 \newcommand\BabelDated[1]{\number#1}
3997 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3998 \newcommand\BabelDateM[1]{\number#1}
3999 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
4000 \newcommand\BabelDateMMMM[1]{%
4001   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
4002 \newcommand\BabelDatey[1]{\number#1}%
4003 \newcommand\BabelDateyy[1]{%
4004   \ifnum#1<10 0\number#1 %
4005   \else\ifnum#1<100 \number#1 %
4006   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
4007   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
4008   \else
4009     \bbl@error

```

```

4010      {Currently two-digit years are restricted to the\
4011      range 0-9999.}%
4012      {There is little you can do. Sorry.}%
4013      \fi\fi\fi\fi}}
4014 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
4015 \def\bbl@replace@finish@iii#1{%
4016   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
4017 \def\bbl@TG@date{%
4018   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
4019   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
4020   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
4021   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
4022   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
4023   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
4024   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
4025   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
4026   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
4027   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
4028   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
4029   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
4030   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
4031 % Note after \bbl@replace \toks@ contains the resulting string.
4032 % TODO - Using this implicit behavior doesn't seem a good idea.
4033   \bbl@replace@finish@iii\bbl@toreplace}
4034 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
4035 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

4036 \let\bbl@release@transforms\@empty
4037 \namedef{bbl@inikv@transforms.prehyphenation}{%
4038   \bbl@transforms\babelprehyphenation}
4039 \namedef{bbl@inikv@transforms.posthyphenation}{%
4040   \bbl@transforms\babelposthyphenation}
4041 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
4042 \begingroup
4043   \catcode\%=12
4044   \catcode\&=14
4045   \gdef\bbl@transforms#1#2#3{&%
4046     \ifx\bbl@KVP@transforms\@nil\else
4047       \directlua{
4048         str = [==[#2]==]
4049         str = str:gsub('%.%d+%.%d+$', '')
4050         tex.print([[def\string\babeltempa{]] .. str .. [[]]])
4051       }&%
4052       \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
4053       \ifin@
4054         \in@{.0$}{#2$}&%
4055       \ifin@
4056         \g@addto@macro\bbl@release@transforms{&%
4057           \relax\bbl@transforms@aux#1{\languagename}{#3}}&%
4058       \else
4059         \g@addto@macro\bbl@release@transforms{, {#3}}&%
4060       \fi
4061     \fi
4062   \fi}
4063 \endgroup

```

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

```

4064 \def\bbl@provide@lsys#1{%
4065   \bbl@ifunset{bbl@lname@#1}%
4066     {\bbl@load@info{#1}}%
4067     }%
4068   \bbl@csarg\let{lsys@#1}\empty
4069   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
4070   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
4071   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
4072   \bbl@ifunset{bbl@lname@#1}{}%
4073     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
4074   \ifcase\bbl@engine\or\or
4075     \bbl@ifunset{bbl@prehc@#1}{}%
4076     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
4077       }%
4078     {\ifx\bbl@xenohyph\@undefined
4079       \let\bbl@xenohyph\bbl@xenohyph@d
4080       \ifx\AtBeginDocument\@notprerr
4081         \expandafter\@secondoftwo % to execute right now
4082         \fi
4083         \AtBeginDocument{%
4084           \expandafter\bbl@add
4085           \csname selectfont \endcsname{\bbl@xenohyph}%
4086           \expandafter\selectlanguage\expandafter{\language}%
4087           \expandafter\bbl@tglobal\csname selectfont \endcsname}%
4088         \fi}%
4089     \fi
4090   \bbl@csarg\bbl@tglobal{lsys@#1}}
4091 \def\bbl@xenohyph@d{%
4092   \bbl@ifset{bbl@prehc@language}%
4093     {\ifnum\hyphenchar\font=\defaultthyphenchar
4094       \iffontchar\font\bbl@cl{prehc}\relax
4095       \hyphenchar\font\bbl@cl{prehc}\relax
4096       \else\iffontchar\font"200B
4097       \hyphenchar\font"200B
4098       \else
4099         \bbl@warning
4100         {Neither 0 nor ZERO WIDTH SPACE are available\\%
4101           in the current font, and therefore the hyphen\\%
4102           will be printed. Try changing the fontspec's\\%
4103           'HyphenChar' to another value, but be aware\\%
4104           this setting is not safe (see the manual)}%
4105         \hyphenchar\font\defaultthyphenchar
4106       \fi\fi
4107     \fi}%
4108   {\hyphenchar\font\defaultthyphenchar}}
4109 % \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).

```

4110 \def\bbl@load@info#1{%
4111   \def\BabelBeforeIni##1##2{%
4112     \begingroup
4113       \bbl@read@ini{##1}0%
4114       \endinput          % babel- .tex may contain onlypreamble's
4115       \endgroup}%        boxed, to avoid extra spaces:
4116   {\bbl@input@texini{#1}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat

[illegible]

```

4148 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={
4149   \ifx\\#1%
4150     \bbl@exp{%
4151       \def\\bbl@tempa####1{%
4152         \<ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
4153     \else
4154       \toks@\xexpandafter{\the\toks@\or #1}%
4155       \expandafter\bbl@buildifcase
4156   \fi}

```

```

4157 \newcommand\localenumberal[2]{\bbl@cs{cntnr@#1\@language}{#2}}
4158 \def\bbl@localecntnr#1#2{\localenumberal{#2}{#1}}
4159 \newcommand\localecounter[2]{%
4160   \expandafter\bbl@localecntnr
4161   \expandafter{\number\csname c@#2\endcsname}{#1}}
4162 \def\bbl@alphnumerical#1#2{%
4163   \expandafter\bbl@alphnumerical{i\number#2 76543210\@@{#1}}
4164 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
4165   \ifcase#9\car#8\@nilor % Currently <10000, but prepared for bigger

```

```

4166 \bbl@alphanumeric@ii{#9}00000#1\or
4167 \bbl@alphanumeric@ii{#9}00000#1#2\or
4168 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
4169 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
4170 \bbl@alphnum@invalid{>9999}%
4171 \fi}
4172 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
4173 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
4174 {\bbl@cs{cntr@#1.4@\language}#5%
4175 \bbl@cs{cntr@#1.3@\language}#6%
4176 \bbl@cs{cntr@#1.2@\language}#7%
4177 \bbl@cs{cntr@#1.1@\language}#8%
4178 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4179 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
4180 {\bbl@cs{cntr@#1.S.321@\language}}%
4181 \fi}%
4182 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}
4183 \def\bbl@alphnum@invalid#1{%
4184 \bbl@error{Alphabetic numeral too large (#1)}%
4185 {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.

```

4186 \newcommand\localeinfo[1]{%
4187 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
4188 {\bbl@error{I've found no info for the current locale.\%
4189 The corresponding ini file has not been loaded\%
4190 Perhaps it doesn't exist}%
4191 {See the manual for details.}}%
4192 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}
4193 \@namedef{bbl@info@name.locale}{lname}
4194 \@namedef{bbl@info@tag.ini}{lini}
4195 \@namedef{bbl@info@name.english}{elname}
4196 \@namedef{bbl@info@name.opentype}{lname}
4197 \@namedef{bbl@info@tag.bcp47}{tbc}
4198 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
4199 \@namedef{bbl@info@tag.opentype}{lotf}
4200 \@namedef{bbl@info@script.name}{esname}
4201 \@namedef{bbl@info@script.name.opentype}{sname}
4202 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4203 \@namedef{bbl@info@script.tag.opentype}{sotf}
4204 \let\bbl@ensureinfo\@gobble
4205 \newcommand\BabelEnsureInfo{%
4206 \ifx\InputIfFileExists\undefined\else
4207 \def\bbl@ensureinfo##1{%
4208 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
4209 \fi
4210 \bbl@foreach\bbl@loaded{%
4211 \def\language{##1}%
4212 \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`.

```

4213 \newcommand\getlocaleproperty{%
4214 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4215 \def\bbl@getproperty@s#1#2#3{%
4216 \let#1\relax
4217 \def\bbl@elt##1##2##3{%

```

```

4218 \bbl@ifsamestring{##1/##2}{#3}%
4219 {\providecommand#1{##3}%
4220 \def\bbl@elt####1####2####3{}}%
4221 {}}%
4222 \bbl@cs{inidata@#2}}%
4223 \def\bbl@getproperty@x#1#2#3{%
4224 \bbl@getproperty@s{#1}{#2}{#3}%
4225 \ifx#1\relax
4226 \bbl@error
4227 {Unknown key for locale '#2':\%
4228 #3\%
4229 \string#1 will be set to \relax}%
4230 {Perhaps you misspelled it.}%
4231 \fi}
4232 \let\bbl@ini@loaded\@empty
4233 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel behavior

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

```

4234 \newcommand\babeladjust[1]{% TODO. Error handling.
4235 \bbl@forkv{#1}{%
4236 \bbl@ifunset{\bbl@ADJ@##1@##2}%
4237 {\bbl@cs{ADJ@##1}{##2}}%
4238 {\bbl@cs{ADJ@##1@##2}}}
4239 %
4240 \def\bbl@adjust@lua#1#2{%
4241 \ifvmode
4242 \ifnum\currentgrouplevel=\z@
4243 \directlua{ Babel.#2 }%
4244 \expandafter\expandafter\expandafter\@gobble
4245 \fi
4246 \fi
4247 {\bbl@error % The error is gobbled if everything went ok.
4248 {Currently, #1 related features can be adjusted only\%
4249 in the main vertical list.}%
4250 {Maybe things change in the future, but this is what it is.}}}
4251 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
4252 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4253 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
4254 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4255 \@namedef{\bbl@ADJ@bidi.text@on}{%
4256 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4257 \@namedef{\bbl@ADJ@bidi.text@off}{%
4258 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4259 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
4260 \bbl@adjust@lua{bidi}{digits_mapped=true}}
4261 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
4262 \bbl@adjust@lua{bidi}{digits_mapped=false}}
4263 %
4264 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
4265 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4266 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
4267 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4268 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
4269 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4270 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
4271 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}

```

```

4272 \@namedef{bbl@ADJ@justify.arabic@on}{%
4273   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4274 \@namedef{bbl@ADJ@justify.arabic@off}{%
4275   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4276 %
4277 \def\bbl@adjust@layout#1{%
4278   \ifvmode
4279     #1%
4280     \expandafter\@gobble
4281   \fi
4282   {\bbl@error   % The error is gobbled if everything went ok.
4283     {Currently, layout related features can be adjusted only\\%
4284       in vertical mode.}%
4285     {Maybe things change in the future, but this is what it is.}}}
4286 \@namedef{bbl@ADJ@layout.tabular@on}{%
4287   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
4288 \@namedef{bbl@ADJ@layout.tabular@off}{%
4289   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4290 \@namedef{bbl@ADJ@layout.lists@on}{%
4291   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4292 \@namedef{bbl@ADJ@layout.lists@off}{%
4293   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4294 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4295   \bbl@activateposthyphen}
4296 %
4297 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4298   \bbl@bcppallowedtrue}
4299 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4300   \bbl@bcppallowedfalse}
4301 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4302   \def\bbl@bcp@prefix{#1}}
4303 \def\bbl@bcp@prefix{bcp47-}
4304 \@namedef{bbl@ADJ@autoload.options}#1{%
4305   \def\bbl@autoload@options{#1}}
4306 \let\bbl@autoload@bcptoptions\@empty
4307 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4308   \def\bbl@autoload@bcptoptions{#1}}
4309 \newif\ifbbl@bcptname
4310 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4311   \bbl@bcptnametrue}
4312   \BabelEnsureInfo}
4313 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4314   \bbl@bcptnamefalse}
4315 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4316   \directlua{ Babel.ignore_pre_char = function(node)
4317     return (node.lang == \the\csname l@nohyphenation\endcsname)
4318   end }}
4319 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4320   \directlua{ Babel.ignore_pre_char = function(node)
4321     return false
4322   end }}
4323 % TODO: use babel name, override
4324 %
4325 % As the final task, load the code for lua.
4326 %
4327 \ifx\directlua\@undefined\else
4328   \ifx\bbl@luapatterns\@undefined
4329     \input luabelabel.def
4330   \fi

```

```

4331 \fi
4332 </core>

A proxy file for switch.def

4333 <*kernel>
4334 \let\bbl@onlyswitch\@empty
4335 \input babel.def
4336 \let\bbl@onlyswitch\@undefined
4337 </kernel>
4338 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by \LaTeX because it should instruct \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 \LaTeX 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.

```

4339 <<Make sure ProvidesFile is defined>>
4340 \ProvidesFile{hyphen.cfg}[\<date>\<version>] Babel hyphens]
4341 \xdef\bbl@format{\jobname}
4342 \def\bbl@version{\<version>}
4343 \def\bbl@date{\<date>}
4344 \ifx\AtBeginDocument\@undefined
4345   \def\@empty{}
4346   \let\orig@dump\dump
4347   \def\dump{%
4348     \ifx\@ztryfc\@undefined
4349     \else
4350       \toks0=\expandafter{\@preamblecmds}%
4351       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4352       \def\@begindocumenthook{}%
4353     \fi
4354     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4355 \fi
4356 <<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.

```

4357 \def\process@line#1#2 #3 #4 {%
4358   \ifx=#1%
4359     \process@synonym{#2}%
4360   \else
4361     \process@language{#1#2}{#3}{#4}%
4362   \fi
4363   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bbl@languages` is also set to empty.

```

4364 \toks@{}
4365 \def\bbl@languages{}

```


When no languages have been loaded yet, the name following the = will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.) Otherwise the name will be a synonym for the language loaded last. We also need to copy the `hyphenmin` parameters for the synonym.

```

4366 \def\process@synonym#1{%
4367   \ifnum\last@language=\m@ne
4368     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4369   \else
4370     \expandafter\chardef\csname l@#1\endcsname\last@language
4371     \wlog{\string\l@#1=\string\language\the\last@language}%
4372     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4373       \csname\language\hyphenmins\endcsname
4374     \let\bbl@elt\relax
4375     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4376   \fi}

```

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

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

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

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

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

Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with =. Note also the language name can have encoding info.

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

```

4377 \def\process@language#1#2#3{%
4378   \expandafter\addlanguage\csname l@#1\endcsname
4379   \expandafter\language\csname l@#1\endcsname
4380   \edef\language{#1}%
4381   \bbl@hook@everylanguage{#1}%
4382   % > luatex
4383   \bbl@get@enc#1::@@@
4384   \begingroup
4385     \lefthyphenmin\m@ne
4386     \bbl@hook@loadpatterns{#2}%
4387     % > luatex
4388     \ifnum\lefthyphenmin=\m@ne
4389       \else
4390         \expandafter\xdef\csname #1hyphenmins\endcsname{%

```

```

4391 \the\lefthyphenmin\the\righthyphenmin}%
4392 \fi
4393 \endgroup
4394 \def\bbl@tempa{#3}%
4395 \ifx\bbl@tempa\@empty\else
4396 \bbl@hook@loadexceptions{#3}%
4397 % > luatex
4398 \fi
4399 \let\bbl@elt\relax
4400 \edef\bbl@languages{%
4401 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4402 \ifnum\the\language=\z@
4403 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4404 \set@hyphenmins\tw@\thr@@\relax
4405 \else
4406 \expandafter\expandafter\expandafter\set@hyphenmins
4407 \csname #1hyphenmins\endcsname
4408 \fi
4409 \the\toks@
4410 \toks@{}%
4411 \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.

```

4412 \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.

```

4413 \def\bbl@hook@everylanguage#1{}
4414 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4415 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4416 \def\bbl@hook@loadkernel#1{%
4417 \def\addlanguage{\csname newlanguage\endcsname}%
4418 \def\adddialect##1##2{%
4419 \global\chardef##1##2\relax
4420 \wlog{\string##1 = a dialect from \string\language##2}}%
4421 \def\iflanguage##1{%
4422 \expandafter\ifx\csname l@##1\endcsname\relax
4423 \nol@nerr{##1}%
4424 \else
4425 \ifnum\csname l@##1\endcsname=\language
4426 \expandafter\expandafter\expandafter\@firstoftwo
4427 \else
4428 \expandafter\expandafter\expandafter\@secondoftwo
4429 \fi
4430 \fi}%
4431 \def\providehyphenmins##1##2{%
4432 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4433 \n@medef{##1hyphenmins}{##2}%
4434 \fi}%
4435 \def\set@hyphenmins##1##2{%
4436 \lefthyphenmin##1\relax
4437 \righthyphenmin##2\relax}%
4438 \def\selectlanguage{%
4439 \errhelp{Selecting a language requires a package supporting it}%
4440 \errmessage{Not loaded}}%
4441 \let\foreignlanguage\selectlanguage

```

```

4442 \let\otherlanguage\selectlanguage
4443 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4444 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4445 \def\setlocale{%
4446   \errhelp{Find an armchair, sit down and wait}%
4447   \errmessage{Not yet available}}%
4448 \let\uselocale\setlocale
4449 \let\locale\setlocale
4450 \let\selectlocale\setlocale
4451 \let\localename\setlocale
4452 \let\textlocale\setlocale
4453 \let\textlanguage\setlocale
4454 \let\languagetext\setlocale}
4455 \begingroup
4456 \def\AddBabelHook#1#2{%
4457   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4458   \def\next{\toks1}%
4459   \else
4460     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4461   \fi
4462   \next}
4463 \ifx\directlua\@undefined
4464   \ifx\XeTeXinputencoding\@undefined\else
4465     \input xebabel.def
4466   \fi
4467 \else
4468   \input luababel.def
4469   \fi
4470 \openin1 = babel-\bbl@format.cfg
4471 \ifeof1
4472 \else
4473   \input babel-\bbl@format.cfg\relax
4474 \fi
4475 \closein1
4476 \endgroup
4477 \bbl@hook@loadkernel{switch.def}

```

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

```

4478 \openin1 = language.dat

```

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

```

4479 \def\language{english}%
4480 \ifeof1
4481 \message{I couldn't find the file language.dat,\space
4482         I will try the file hyphen.tex}
4483 \input hyphen.tex\relax
4484 \chardef\l@english\z@
4485 \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 .

```

4486 \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.

```

4487 \loop
4488 \endlinechar\m@ne
4489 \read1 to \bbl@line
4490 \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.

```

4491 \if T\ifeof1F\fi T\relax
4492 \ifx\bbl@line\@empty\else
4493 \edef\bbl@line{\bbl@line\space\space\space}%
4494 \expandafter\process@line\bbl@line\relax
4495 \fi
4496 \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.

```

4497 \begingroup
4498 \def\bbl@elt#1#2#3#4{%
4499 \global\language=#2\relax
4500 \gdef\languagename{#1}%
4501 \def\bbl@elt##1##2##3##4{}}%
4502 \bbl@languages
4503 \endgroup
4504 \fi
4505 \closein1

```

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

```

4506 \if/\the\toks@/\else
4507 \errhelp{language.dat loads no language, only synonyms}
4508 \errmessage{Orphan language synonym}
4509 \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.

```

4510 \let\bbl@line\@undefined
4511 \let\process@line\@undefined
4512 \let\process@synonym\@undefined
4513 \let\process@language\@undefined
4514 \let\bbl@get@enc\@undefined
4515 \let\bbl@hyph@enc\@undefined
4516 \let\bbl@tempa\@undefined
4517 \let\bbl@hook@loadkernel\@undefined
4518 \let\bbl@hook@everylanguage\@undefined
4519 \let\bbl@hook@loadpatterns\@undefined
4520 \let\bbl@hook@loadexceptions\@undefined
4521 \</patterns>

```

Here the code for iniTeX ends.

12 Font handling with fontspec

Add the bidi handler just before luaotfload, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```

4522 <(*More package options)> \equiv
4523 \chardef\bbl@bidimode\z@
4524 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4525 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }

```

```

4526 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4527 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4528 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4529 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4530 <</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.

```

4531 <<{*Font selection}>> ≡
4532 \bbl@trace{Font handling with fontspec}
4533 \ifx\ExplSyntaxOn\undefined\else
4534   \ExplSyntaxOn
4535   \catcode\ =10
4536   \def\bbl@loadfontspec{%
4537     \usepackage{fontspec}%
4538     \expandafter
4539     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4540       Font '\l_fontspec_fontname_tl' is using the\\%
4541       default features for language '##1'.\\%
4542       That's usually fine, because many languages\\%
4543       require no specific features, but if the output is\\%
4544       not as expected, consider selecting another font.}
4545     \expandafter
4546     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4547       Font '\l_fontspec_fontname_tl' is using the\\%
4548       default features for script '##2'.\\%
4549       That's not always wrong, but if the output is\\%
4550       not as expected, consider selecting another font.}}
4551   \ExplSyntaxOff
4552 \fi
4553 \@onlypreamble\babelfont
4554 \newcommand\babelfont[2][% 1=langs/scripts 2=fam
4555   \bbl@foreach{#1}{%
4556     \expandafter\ifx\csname date##1\endcsname\relax
4557       \IfFileExists{babel-##1.tex}%
4558       {\babelprovide{##1}}%
4559     }%
4560   \fi}%
4561 \edef\bbl@tempa{#1}%
4562 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4563 \ifx\fontspec\undefined
4564   \bbl@loadfontspec
4565 \fi
4566 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4567 \bbl@bblfont}
4568 \newcommand\bbl@bblfont[2][% 1=features 2=fontname, @font=rm|sf|tt
4569   \bbl@ifunset{\bbl@tempb family}%
4570   {\bbl@providedefam{\bbl@tempb}}%
4571   {\bbl@exp{%
4572     \\bbl@sreplace<\bbl@tempb family >%
4573     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4574   % For the default font, just in case:
4575   \bbl@ifunset{\bbl@lsys@languagenamename}{\bbl@provide@lsys{languagenamename}}}%
4576   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4577   {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@

```

```

4578 \bbl@exp{%
4579 \let<\bbl@bbl@tempb dflt@<\language>\<\bbl@bbl@tempb dflt@>%
4580 \\\bbl@font@set<\bbl@bbl@tempb dflt@<\language>%
4581 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4582 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4583 \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:

```

4584 \def\bbl@providefam#1{%
4585 \bbl@exp{%
4586 \\\newcommand<#1default>{}% Just define it
4587 \\\bbl@add@list\\bbl@font@fams{#1}%
4588 \\\DeclareRobustCommand<#1family>{%
4589 \\\not@math@alphabet<#1family>\relax
4590 \\\fontfamily<#1default>\selectfont}%
4591 \\\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.

```

4592 \def\bbl@nostdfont#1{%
4593 \bbl@ifunset{\bbl@WFF@f@family}%
4594 {\bbl@csarg\gdef{\bbl@WFF@f@family}}}% Flag, to avoid dupl warns
4595 \bbl@infowarn{The current font is not a babel standard family:\\%
4596 #1%
4597 \fontname\font\\%
4598 There is nothing intrinsically wrong with this warning, and\\%
4599 you can ignore it altogether if you do not need these\\%
4600 families. But if they are used in the document, you should be\\%
4601 aware 'babel' will no set Script and Language for them, so\\%
4602 you may consider defining a new family with \string\babelfont.\\%
4603 See the manual for further details about \string\babelfont.\\%
4604 Reported}}
4605 {}}%
4606 \gdef\bbl@switchfont{%
4607 \bbl@ifunset{\bbl@lsys@<\language>}{\bbl@provide@lsys{<\language>}}}%
4608 \bbl@exp{% eg Arabic -> arabic
4609 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4610 \bbl@foreach\bbl@font@fams{%
4611 \bbl@ifunset{\bbl@##1dflt@<\language>% (1) language?
4612 {\bbl@ifunset{\bbl@##1dflt@*\\bbl@tempa}% (2) from script?
4613 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4614 {}% 123=F - nothing!
4615 {\bbl@exp{% 3=T - from generic
4616 \global\let<\bbl@##1dflt@<\language>%
4617 \<\bbl@##1dflt@>}}}%
4618 {\bbl@exp{% 2=T - from script
4619 \global\let<\bbl@##1dflt@<\language>%
4620 \<\bbl@##1dflt@*\\bbl@tempa>}}}%
4621 {}}% 1=T - language, already defined
4622 \def\bbl@tempa{\bbl@nostdfont}}}%
4623 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4624 \bbl@ifunset{\bbl@##1dflt@<\language>%
4625 {\bbl@cs{famrst@##1}%
4626 \global\bbl@csarg\let{famrst@##1}\relax}%
4627 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4628 \\\bbl@add\\originalTeX{%
4629 \\\bbl@font@rst{\bbl@cl{##1dflt}}}%
4630 \<##1default>\<##1family>{##1}}}%
4631 \\\bbl@font@set<\bbl@##1dflt@<\language>% the main part!

```

```

4632 \<##1default>\<##1family>}}}%
4633 \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.

```

4634 \ifx\font@family\undefined\else % if latex
4635 \ifcase\bbl@engine % if pdftex
4636 \let\bbl@cckstdfonts\relax
4637 \else
4638 \def\bbl@cckstdfonts{%
4639 \begingroup
4640 \global\let\bbl@cckstdfonts\relax
4641 \let\bbl@tempa\@empty
4642 \bbl@foreach\bbl@font@fams{%
4643 \bbl@ifunset\bbl@##1dflt@}%
4644 {\nameuse{##1family}%
4645 \bbl@csarg\gdef{WFF@f@family}}}% Flag
4646 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \fontname\font}}}%
4647 \space\space\fontname\font}}}%
4648 \bbl@csarg\xdef{##1dflt@}{f@family}%
4649 \expandafter\xdef\csname ##1default\endcsname{f@family}}}%
4650 {}}%
4651 \ifx\bbl@tempa\@empty\else
4652 \bbl@infowarn{The following font families will use the default\\%
4653 settings for all or some languages:\\%
4654 \bbl@tempa
4655 There is nothing intrinsically wrong with it, but\\%
4656 'babel' will no set Script and Language, which could\\%
4657 be relevant in some languages. If your document uses\\%
4658 these families, consider redefining them with \string\babelfont.\\%
4659 Reported}%
4660 \fi
4661 \endgroup}
4662 \fi
4663 \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.

```

4664 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4665 \bbl@xin@{<>}{#1}%
4666 \ifin@
4667 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4668 \fi
4669 \bbl@exp{%
4670 \def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4671 \bbl@ifsamestring{#2}{f@family}%
4672 {\#3%
4673 \bbl@ifsamestring{f@series}{bfdefault}{bfseries}}}%
4674 \let\bbl@tempa\relax}%
4675 {}}}
4676 % TODO - next should be global?, but even local does its job. I'm
4677 % still not sure -- must investigate:
4678 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4679 \let\bbl@tempe\bbl@mapselect
4680 \let\bbl@mapselect\relax
4681 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4682 \let#4\@empty % Make sure \renewfontfamily is valid

```

```

4683 \bbl@exp{%
4684 \let\bbbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4685 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4686 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4687 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4688 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4689 \renewfontfamily\#4%
4690 [\bbl@cs{lsys@language},#2]{#3}% ie \bbl@exp{.}{#3}
4691 \begingroup
4692 #4%
4693 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4694 \endgroup
4695 \let#4\bbbl@temp@fam
4696 \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4697 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4698 \def\bbl@font@rst#1#2#3#4{%
4699 \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.

```

4700 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preserved 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 :-).

```

4701 \newcommand\babelFSstore[2][{%
4702 \bbl@ifblank{#1}%
4703 {\bbl@csarg\def{sname@#2}{Latin}}%
4704 {\bbl@csarg\def{sname@#2}{#1}}%
4705 \bbl@provide@dirs{#2}%
4706 \bbl@csarg\ifnum{wdir@#2}>\z@
4707 \let\bbl@beforeforeign\leavevmode
4708 \EnableBabelHook{babel-bidi}%
4709 \fi
4710 \bbl@foreach{#2}{%
4711 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4712 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4713 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4714 \def\bbl@FSstore#1#2#3#4{%
4715 \bbl@csarg\edef{#2default#1}{#3}%
4716 \expandafter\addto\csname extras#1\endcsname{%
4717 \let#4#3%
4718 \ifx#3\f@family
4719 \edef#3{\csname bbl@#2default#1\endcsname}%
4720 \fontfamily{#3}\selectfont
4721 \else
4722 \edef#3{\csname bbl@#2default#1\endcsname}%
4723 \fi}%
4724 \expandafter\addto\csname noextras#1\endcsname{%
4725 \ifx#3\f@family
4726 \fontfamily{#4}\selectfont
4727 \fi
4728 \let#3#4}}
4729 \let\bbl@langfeatures\empty
4730 \def\babelFSfeatures{% make sure \fontspec is redefined once
4731 \let\bbl@ori@fontspec\fontspec
4732 \renewcommand\fontspec[1][{%

```



```

4733 \bbl@ori@fontspec[\bbl@langfeatures##1]}
4734 \let\babelFSfeatures\bbl@FSfeatures
4735 \babelFSfeatures}
4736 \def\bbl@FSfeatures#1#2{%
4737 \expandafter\addto\csname extras#1\endcsname{%
4738 \babel@save\bbl@langfeatures
4739 \edef\bbl@langfeatures{#2,}}
4740 <</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.

```

4741 <<{*Footnote changes}>> ≡
4742 \bbl@trace{Bidi footnotes}
4743 \ifnum\bbl@bidimode>\z@
4744 \def\bbl@footnote#1#2#3{%
4745 \@ifnextchar[%
4746 {\bbl@footnote@o{#1}{#2}{#3}}%
4747 {\bbl@footnote@x{#1}{#2}{#3}}}
4748 \long\def\bbl@footnote@x#1#2#3#4{%
4749 \bgroup
4750 \select@language@x{\bbl@main@language}%
4751 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4752 \egroup}
4753 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4754 \bgroup
4755 \select@language@x{\bbl@main@language}%
4756 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4757 \egroup}
4758 \def\bbl@footnotetext#1#2#3{%
4759 \@ifnextchar[%
4760 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4761 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4762 \long\def\bbl@footnotetext@x#1#2#3#4{%
4763 \bgroup
4764 \select@language@x{\bbl@main@language}%
4765 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4766 \egroup}
4767 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4768 \bgroup
4769 \select@language@x{\bbl@main@language}%
4770 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4771 \egroup}
4772 \def\BabelFootnote#1#2#3#4{%
4773 \ifx\bbl@fn@footnote\@undefined
4774 \let\bbl@fn@footnote\footnote
4775 \fi
4776 \ifx\bbl@fn@footnotetext\@undefined
4777 \let\bbl@fn@footnotetext\footnotetext
4778 \fi
4779 \bbl@ifblank{#2}%
4780 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4781 \@namedef{\bbl@stripslash#1text}%
4782 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4783 {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%

```

```

4784 \namedef{\bbl@stripslash#1text}%
4785 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}}
4786 \fi
4787 <</Footnote changes>>

Now, the code.

4788 (*xetex)
4789 \def\BabelStringsDefault{unicode}
4790 \let\xebbl@stop\relax
4791 \AddBabelHook{xetex}{encodedcommands}{%
4792 \def\bbl@tempa{#1}%
4793 \ifx\bbl@tempa\empty
4794 \XeTeXinputencoding"bytes"%
4795 \else
4796 \XeTeXinputencoding"#1"%
4797 \fi
4798 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4799 \AddBabelHook{xetex}{stopcommands}{%
4800 \xebbl@stop
4801 \let\xebbl@stop\relax}
4802 \def\bbl@intraspace#1 #2 #3\@@{%
4803 \bbl@csarg\gdef{\xeisp@\language}%
4804 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4805 \def\bbl@intrapenalty#1\@@{%
4806 \bbl@csarg\gdef{\xeipn@\language}%
4807 {\XeTeXlinebreakpenalty #1\relax}}
4808 \def\bbl@provide@intraspace{%
4809 \bbl@xin@{/s}{\bbl@cl{lnbrk}}}%
4810 \ifin@else\bbl@xin@{/c}{\bbl@cl{lnbrk}}\fi
4811 \ifin@
4812 \bbl@ifunset{\bbl@intsp@\language}{%
4813 {\expandafter\ifx\csname\bbl@intsp@\language\endcsname\empty\else
4814 \ifx\bbl@KVP@intraspace\@nil
4815 \bbl@exp{%
4816 \bbl@intraspace\bbl@cl{intsp}\@@}%
4817 \fi
4818 \ifx\bbl@KVP@intrapenalty\@nil
4819 \bbl@intrapenalty0\@@
4820 \fi
4821 \fi
4822 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4823 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4824 \fi
4825 \ifx\bbl@KVP@intrapenalty\@nil\else
4826 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4827 \fi
4828 \bbl@exp{%
4829 % TODO. Execute only once (but redundant):
4830 \bbl@add\<extras\language>{%
4831 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4832 \<bbl@xeisp@\language>%
4833 \<bbl@xeipn@\language>%
4834 \bbl@toglobal\<extras\language>%
4835 \bbl@add\<noextras\language>{%
4836 \XeTeXlinebreaklocale "en"%
4837 \bbl@toglobal\<noextras\language>}}%
4838 \ifx\bbl@ispace\undefined
4839 \gdef\bbl@ispace{\bbl@cl{\xeisp}}%
4840 \ifx\AtBeginDocument\@notprerr

```

```

4841         \expandafter\@secondoftwo % to execute right now
4842     \fi
4843     \AtBeginDocument{%
4844         \expandafter\bb1@add
4845         \csname selectfont \endcsname{\bb1@ispace size}%
4846         \expandafter\bb1@tglobal\csname selectfont \endcsname}%
4847     \fi}%
4848 \fi}
4849 \ifx\DisableBabelHook\@undefined\endinput\fi
4850 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
4851 \AddBabelHook{babel-fontspec}{beforestart}{\bb1@cckestdfonts}
4852 \DisableBabelHook{babel-fontspec}
4853 <<Font selection>>
4854 \input txtbabel.def
4855 </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.

\bb1@startskip and \bb1@endskip are available to package authors. Thanks to the T_EX expansion mechanism the following constructs are valid: \adim\bb1@startskip, \advance\bb1@startskip\adim, \bb1@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdf_{TE}X and xet_{EX}.

```

4856 <*texxet>
4857 \providecommand\bb1@provide@intraspace{}
4858 \bb1@trace{Redefinitions for bidi layout}
4859 \def\bb1@sspre@caption{%
4860     \bb1@exp{\everybox{\bb1@textdir\bb1@cs{wdir@\bb1@main@language}}}}
4861 \ifx\bb1@opt@layout\@nnil\endinput\fi % No layout
4862 \def\bb1@startskip{\ifcase\bb1@thepardir\leftskip\else\rightskip\fi}
4863 \def\bb1@endskip{\ifcase\bb1@thepardir\rightskip\else\leftskip\fi}
4864 \ifx\bb1@beforeforeign\leavevmode % A poor test for bidi=
4865     \def\@hangfrom#1{%
4866         \setbox\@tempboxa\hbox{#1}%
4867         \hangindent\ifcase\bb1@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4868         \noindent\box\@tempboxa}
4869 \def\raggedright{%
4870     \let\@centercr
4871     \bb1@startskip\z@skip
4872     \@rightskip\@flushglue
4873     \bb1@endskip\@rightskip
4874     \parindent\z@
4875     \parfillskip\bb1@startskip}
4876 \def\raggedleft{%
4877     \let\@centercr
4878     \bb1@startskip\@flushglue
4879     \bb1@endskip\z@skip
4880     \parindent\z@
4881     \parfillskip\bb1@endskip}
4882 \fi
4883 \IfBabelLayout{lists}
4884 {\bb1@sreplace\list
4885     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bb1@listleftmargin}%
4886     \def\bb1@listleftmargin{%
4887         \ifcase\bb1@thepardir\leftmargin\else\rightmargin\fi}%
4888     \ifcase\bb1@engine

```

```

4889 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4890 \def\p@enumiii{\p@enumii}\theenumii{}\%
4891 \fi
4892 \bbl@sreplace\@verbatim
4893 {\leftskip\@totalleftmargin}%
4894 {\bbl@startskip\textwidth
4895 \advance\bbl@startskip-\linewidth}%
4896 \bbl@sreplace\@verbatim
4897 {\rightskip\z@skip}%
4898 {\bbl@endskip\z@skip}%
4899 {}
4900 \IfBabelLayout{contents}
4901 {\bbl@sreplace\@dottedtocline{\leftskip}\bbl@startskip}%
4902 \bbl@sreplace\@dottedtocline{\rightskip}\bbl@endskip}}
4903 {}
4904 \IfBabelLayout{columns}
4905 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}\bbl@outputbox}%
4906 \def\bbl@outputbox#1{%
4907 \hb@xt@\textwidth{%
4908 \hskip\columnwidth
4909 \hfil
4910 {\normalcolor\vrule \@width\columnseprule}%
4911 \hfil
4912 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4913 \hskip-\textwidth
4914 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4915 \hskip\columnsep
4916 \hskip\columnwidth}}}%
4917 {}
4918 <<Footnote changes>>
4919 \IfBabelLayout{footnotes}%
4920 {\BabelFootnote\footnote\language\{}}{}%
4921 \BabelFootnote\localfootnote\language\{}}{}%
4922 \BabelFootnote\mainfootnote\{}}{}%
4923 {}
4924 \IfBabelLayout{counters}%
4925 {\let\bbl@latinarabic=\@arabic
4926 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4927 \let\bbl@asciroman=\@roman
4928 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4929 \let\bbl@asciiRoman=\@Roman
4930 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4931 \texet

```

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.

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 synonyms, 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 file 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`).

```

4932 (*luatex)
4933 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4934 \bbl@trace{Read language.dat}
4935 \ifx\bbl@readstream\undefined
4936   \csname newread\endcsname\bbl@readstream
4937 \fi
4938 \begingroup
4939   \toks@{}
4940   \count@ \z@ % 0=start, 1=0th, 2=normal
4941   \def\bbl@process@line#1#2 #3 #4 {%
4942     \ifx=#1%
4943       \bbl@process@synonym{#2}%
4944     \else
4945       \bbl@process@language{#1#2}{#3}{#4}%
4946     \fi
4947     \ignorespaces}
4948   \def\bbl@manylang{%
4949     \ifnum\bbl@last>\@ne
4950       \bbl@info{Non-standard hyphenation setup}%
4951     \fi
4952     \let\bbl@manylang\relax}
4953   \def\bbl@process@language#1#2#3{%
4954     \ifcase\count@
4955       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4956     \or
4957       \count@\tw@
4958     \fi
4959     \ifnum\count@=\tw@
4960       \expandafter\addlanguage\csname l@#1\endcsname
4961       \language\allocationnumber
4962       \chardef\bbl@last\allocationnumber
4963       \bbl@manylang
4964       \let\bbl@elt\relax
4965       \xdef\bbl@languages{%
4966         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
```

```

4967 \fi
4968 \the\toks@
4969 \toks@{}}
4970 \def\bbl@process@synonym@aux#1#2{%
4971 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4972 \let\bbl@elt\relax
4973 \xdef\bbl@languages{%
4974 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4975 \def\bbl@process@synonym#1{%
4976 \ifcase\count@
4977 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4978 \or
4979 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4980 \else
4981 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4982 \fi}
4983 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4984 \chardef\l@english\z@
4985 \chardef\l@USenglish\z@
4986 \chardef\bbl@last\z@
4987 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4988 \gdef\bbl@languages{%
4989 \bbl@elt{english}{0}{hyphen.tex}}%
4990 \bbl@elt{USenglish}{0}{}}
4991 \else
4992 \global\let\bbl@languages@format\bbl@languages
4993 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4994 \ifnum#2>\z@\else
4995 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4996 \fi}%
4997 \xdef\bbl@languages{\bbl@languages}%
4998 \fi
4999 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
5000 \bbl@languages
5001 \openin\bbl@readstream=language.dat
5002 \ifeof\bbl@readstream
5003 \bbl@warning{I couldn't find language.dat. No additional\\%
5004 patterns loaded. Reported}%
5005 \else
5006 \loop
5007 \endlinechar\m@ne
5008 \read\bbl@readstream to \bbl@line
5009 \endlinechar\^^M
5010 \if T\ifeof\bbl@readstream F\fi T\relax
5011 \ifx\bbl@line\empty\else
5012 \edef\bbl@line{\bbl@line\space\space\space}%
5013 \expandafter\bbl@process@line\bbl@line\relax
5014 \fi
5015 \repeat
5016 \fi
5017 \endgroup
5018 \bbl@trace{Macros for reading patterns files}
5019 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5020 \ifx\babelcatcodetablenum\undefined
5021 \ifx\newcatcodetable\undefined
5022 \def\babelcatcodetablenum{5211}
5023 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5024 \else
5025 \newcatcodetable\babelcatcodetablenum

```

```

5026 \newcatcodetable\bb1@pattcodes
5027 \fi
5028 \else
5029 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5030 \fi
5031 \def\bb1@luapatterns#1#2{%
5032 \bb1@get@enc#1::\@@@
5033 \setbox\z@\hbox\bgroup
5034 \begingroup
5035 \savecatcodetable\babelcatcodetablenum\relax
5036 \initcatcodetable\bb1@pattcodes\relax
5037 \catcodetable\bb1@pattcodes\relax
5038 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5039 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
5040 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5041 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5042 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5043 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
5044 \input #1\relax
5045 \catcodetable\babelcatcodetablenum\relax
5046 \endgroup
5047 \def\bb1@tempa{#2}%
5048 \ifx\bb1@tempa\@empty\else
5049 \input #2\relax
5050 \fi
5051 \egroup}%
5052 \def\bb1@patterns@lua#1{%
5053 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5054 \csname l@#1\endcsname
5055 \edef\bb1@tempa{#1}%
5056 \else
5057 \csname l@#1:\f@encoding\endcsname
5058 \edef\bb1@tempa{#1:\f@encoding}%
5059 \fi\relax
5060 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5061 \@ifundefined{bb1@hyphendata@the\language}%
5062 {\def\bb1@elt##1##2##3##4{%
5063 \ifnum##2=\csname l@bb1@tempa\endcsname % #2=spanish, dutch:OT1...
5064 \def\bb1@tempb{##3}%
5065 \ifx\bb1@tempb\@empty\else % if not a synonymous
5066 \def\bb1@tempc{##3}{##4}%
5067 \fi
5068 \bb1@csarg\xdef{hyphendata@##2}{\bb1@tempc}%
5069 \fi}%
5070 \bb1@languages
5071 \@ifundefined{bb1@hyphendata@the\language}%
5072 {\bb1@info{No hyphenation patterns were set for\%
5073 language '\bb1@tempa'. Reported}}%
5074 {\expandafter\expandafter\expandafter\bb1@luapatterns
5075 \csname bb1@hyphendata@the\language\endcsname}}}%
5076 \endinput\fi
5077 % Here ends \ifx\AddBabelHook\@undefined
5078 % A few lines are only read by hyphen.cfg
5079 \ifx\DisableBabelHook\@undefined
5080 \AddBabelHook{luatex}{everylanguage}{%
5081 \def\process@language##1##2##3{%
5082 \def\process@line####1####2 ####3 ####4 {}}}
5083 \AddBabelHook{luatex}{loadpatterns}{%
5084 \input #1\relax

```

```

5085 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5086 {{#1}}}}
5087 \AddBabelHook{luatex}{loadexceptions}{%
5088 \input #1\relax
5089 \def\bbl@tempb##1##2{{##1}{##2}}}%
5090 \expandafter\def\csname bbl@hyphendata@the\language\endcsname
5091 {\expandafter\expandafter\expandafter\bbl@tempb
5092 \csname bbl@hyphendata@the\language\endcsname}}
5093 \endinput\fi
5094 % Here stops reading code for hyphen.cfg
5095 % The following is read the 2nd time it's loaded
5096 \begingroup % TODO - to a lua file
5097 \catcode`\%=12
5098 \catcode`\'=12
5099 \catcode`\ "=12
5100 \catcode`\:=12
5101 \directlua{
5102 Babel = Babel or {}
5103 function Babel.bytes(line)
5104 return line:gsub(".",
5105 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5106 end
5107 function Babel.begin_process_input()
5108 if luatexbase and luatexbase.add_to_callback then
5109 luatexbase.add_to_callback('process_input_buffer',
5110 Babel.bytes, 'Babel.bytes')
5111 else
5112 Babel.callback = callback.find('process_input_buffer')
5113 callback.register('process_input_buffer', Babel.bytes)
5114 end
5115 end
5116 function Babel.end_process_input ()
5117 if luatexbase and luatexbase.remove_from_callback then
5118 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5119 else
5120 callback.unregister('process_input_buffer', Babel.callback)
5121 end
5122 end
5123 function Babel.addpatterns(pp, lg)
5124 local lg = lang.new(lg)
5125 local pats = lang.patterns(lg) or ''
5126 lang.clear_patterns(lg)
5127 for p in pp:gmatch('[^%s]+') do
5128 ss = ''
5129 for i in string.utfcharacters(p:gsub('%d', '')) do
5130 ss = ss .. '%d?' .. i
5131 end
5132 ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5133 ss = ss:gsub('%.%d%?$', '%%.')
5134 pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5135 if n == 0 then
5136 tex.sprint(
5137 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5138 .. p .. [[]])
5139 pats = pats .. ' ' .. p
5140 else
5141 tex.sprint(
5142 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5143 .. p .. [[]])

```



```

5144     end
5145   end
5146   lang.patterns(lg, pats)
5147 end
5148 }
5149 \endgroup
5150 \ifx\newattribute\@undefined\else
5151   \newattribute\bbl@attr@locale
5152   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5153   \AddBabelHook{luatex}{beforeextras}{%
5154     \setattribute\bbl@attr@locale\localeid}
5155 \fi
5156 \def\BabelStringsDefault{unicode}
5157 \let\luabbl@stop\relax
5158 \AddBabelHook{luatex}{encodedcommands}{%
5159   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5160   \ifx\bbl@tempa\bbl@tempb\else
5161     \directlua{Babel.begin_process_input()}%
5162     \def\luabbl@stop{%
5163       \directlua{Babel.end_process_input()}}%
5164   \fi}%
5165 \AddBabelHook{luatex}{stopcommands}{%
5166   \luabbl@stop
5167   \let\luabbl@stop\relax}
5168 \AddBabelHook{luatex}{patterns}{%
5169   \@ifundefined{bbl@hyphendata@the\language}%
5170     {\def\bbl@elt##1##2##3##4{%
5171       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5172       \def\bbl@tempb{##3}%
5173       \ifx\bbl@tempb\@empty\else % if not a synonymous
5174         \def\bbl@tempc{##3}{##4}%
5175       \fi
5176       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5177     \fi}%
5178   \bbl@languages
5179   \@ifundefined{bbl@hyphendata@the\language}%
5180     {\bbl@info{No hyphenation patterns were set for\%
5181       language '#2'. Reported}}%
5182     {\expandafter\expandafter\expandafter\bbl@luapatterns
5183       \csname bbl@hyphendata@the\language\endcsname}}}%
5184   \@ifundefined{bbl@patterns@}{}%
5185   \begingroup
5186     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5187   \ifin@else
5188     \ifx\bbl@patterns@\@empty\else
5189       \directlua{ Babel.addpatterns(
5190         [[\bbl@patterns@]], \number\language) }%
5191     \fi
5192     \@ifundefined{bbl@patterns@#1}%
5193     \@empty
5194     {\directlua{ Babel.addpatterns(
5195       [[\space\csname bbl@patterns@#1\endcsname]],
5196       \number\language) }%
5197     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5198   \fi
5199   \endgroup}%
5200 \bbl@exp{%
5201   \bbl@ifunset{bbl@prehc@\languagename}{}%
5202   {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%

```

```
5203      {\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.

```
5204 \@onlypreamble\babelpatterns
5205 \AtEndOfPackage{%
5206   \newcommand\babelpatterns[2][\@empty]{%
5207     \ifx\bbl@patterns@\relax
5208       \let\bbl@patterns@\@empty
5209     \fi
5210     \ifx\bbl@pttnlist\@empty\else
5211       \bbl@warning{%
5212         You must not intermingle \string\selectlanguage\space and\%
5213         \string\babelpatterns\space or some patterns will not\%
5214         be taken into account. Reported}%
5215     \fi
5216     \ifx\@empty#1%
5217       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5218     \else
5219       \edef\bbl@tempb{\zap@space#1 \@empty}%
5220       \bbl@for\bbl@tempa\bbl@tempb{%
5221         \bbl@fixname\bbl@tempa
5222         \bbl@iflanguage\bbl@tempa{%
5223           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5224             \@ifundefined{bbl@patterns@\bbl@tempa}%
5225               \@empty
5226               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5227             #2}}}%
5228     \fi}}
```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```
5229% TODO - to a lua file
5230 \directlua{
5231   Babel = Babel or {}
5232   Babel.linebreaking = Babel.linebreaking or {}
5233   Babel.linebreaking.before = {}
5234   Babel.linebreaking.after = {}
5235   Babel.locale = {} % Free to use, indexed by \localeid
5236   function Babel.linebreaking.add_before(func)
5237     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5238     table.insert(Babel.linebreaking.before, func)
5239   end
5240   function Babel.linebreaking.add_after(func)
5241     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5242     table.insert(Babel.linebreaking.after, func)
5243   end
5244 }
5245 \def\bbl@intraspace#1 #2 #3\@{#1%
5246   \directlua{
5247     Babel = Babel or {}
5248     Babel.intraspaces = Babel.intraspaces or {}
5249     Babel.intraspaces['\csname bbl@sbcpr@\language\endcsname'] = %
```

```

5250     {b = #1, p = #2, m = #3}
5251     Babel.locale_props[\the\localeid].intraspace = %
5252     {b = #1, p = #2, m = #3}
5253   }}
5254 \def\bbl@intrapenalty#1\@@{%
5255   \directlua{
5256     Babel = Babel or {}
5257     Babel.intrapenalties = Babel.intrapenalties or {}
5258     Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5259     Babel.locale_props[\the\localeid].intrapenalty = #1
5260   }}
5261 \begingroup
5262 \catcode`\%=12
5263 \catcode`\^=14
5264 \catcode`\'=12
5265 \catcode`\~=12
5266 \gdef\bbl@seaintraspace{^
5267   \let\bbl@seaintraspace\relax
5268   \directlua{
5269     Babel = Babel or {}
5270     Babel.sea_enabled = true
5271     Babel.sea_ranges = Babel.sea_ranges or {}
5272     function Babel.set_chranges (script, chrng)
5273       local c = 0
5274       for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5275         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5276         c = c + 1
5277       end
5278     end
5279     function Babel.sea_disc_to_space (head)
5280       local sea_ranges = Babel.sea_ranges
5281       local last_char = nil
5282       local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5283       for item in node.traverse(head) do
5284         local i = item.id
5285         if i == node.id'glyph' then
5286           last_char = item
5287         elseif i == 7 and item.subtype == 3 and last_char
5288           and last_char.char > 0x0C99 then
5289           quad = font.getfont(last_char.font).size
5290           for lg, rg in pairs(sea_ranges) do
5291             if last_char.char > rg[1] and last_char.char < rg[2] then
5292               lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyr11
5293               local intraspace = Babel.intraspaces[lg]
5294               local intrapenalty = Babel.intrapenalties[lg]
5295               local n
5296               if intrapenalty ~= 0 then
5297                 n = node.new(14, 0) ^% penalty
5298                 n.penalty = intrapenalty
5299                 node.insert_before(head, item, n)
5300               end
5301               n = node.new(12, 13) ^% (glue, spaceskip)
5302               node.setglue(n, intraspace.b * quad,
5303                 intraspace.p * quad,
5304                 intraspace.m * quad)
5305               node.insert_before(head, item, n)
5306               node.remove(head, item)
5307             end
5308           end

```

```

5309         end
5310     end
5311 end
5312 }^^
5313 \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.

```

5314 \catcode`\%=14
5315 \gdef\bbl@cjkintraspacespace{%
5316   \let\bbl@cjkintraspacespace\relax
5317   \directlua{
5318     Babel = Babel or {}
5319     require('babel-data-cjk.lua')
5320     Babel.cjk_enabled = true
5321     function Babel.cjk_linebreak(head)
5322       local GLYPH = node.id'glyph'
5323       local last_char = nil
5324       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5325       local last_class = nil
5326       local last_lang = nil
5327
5328       for item in node.traverse(head) do
5329         if item.id == GLYPH then
5330
5331           local lang = item.lang
5332
5333           local LOCALE = node.get_attribute(item,
5334             luatexbase.registernumber'bbl@attr@locale')
5335           local props = Babel.locale_props[LOCALE]
5336
5337           local class = Babel.cjk_class[item.char].c
5338
5339           if props.cjk_quotes and props.cjk_quotes[item.char] then
5340             class = props.cjk_quotes[item.char]
5341           end
5342
5343           if class == 'cp' then class = 'cl' end % ]] as CL
5344           if class == 'id' then class = 'I' end
5345
5346           local br = 0
5347           if class and last_class and Babel.cjk_breaks[last_class][class] then
5348             br = Babel.cjk_breaks[last_class][class]
5349           end
5350
5351           if br == 1 and props.linebreak == 'c' and
5352             lang ~= \the\l@nohyphenation\space and
5353             last_lang ~= \the\l@nohyphenation then
5354             local intrapenalty = props.intrapenalty
5355             if intrapenalty ~= 0 then
5356               local n = node.new(14, 0)      % penalty
5357               n.penalty = intrapenalty

```

```

5358         node.insert_before(head, item, n)
5359     end
5360     local intraspace = props.intraspace
5361     local n = node.new(12, 13)      % (glue, spaceskip)
5362     node.setglue(n, intraspace.b * quad,
5363                 intraspace.p * quad,
5364                 intraspace.m * quad)
5365     node.insert_before(head, item, n)
5366 end
5367
5368 if font.getfont(item.font) then
5369     quad = font.getfont(item.font).size
5370 end
5371 last_class = class
5372 last_lang = lang
5373 else % if penalty, glue or anything else
5374     last_class = nil
5375 end
5376 end
5377 lang.hyphenate(head)
5378 end
5379 }%
5380 \bbl@luahyphenate}
5381 \gdef\bbl@luahyphenate{%
5382 \let\bbl@luahyphenate\relax
5383 \directlua{
5384     luatexbase.add_to_callback('hyphenate',
5385     function (head, tail)
5386         if Babel.linebreaking.before then
5387             for k, func in ipairs(Babel.linebreaking.before) do
5388                 func(head)
5389             end
5390         end
5391         if Babel.cjk_enabled then
5392             Babel.cjk_linebreak(head)
5393         end
5394         lang.hyphenate(head)
5395         if Babel.linebreaking.after then
5396             for k, func in ipairs(Babel.linebreaking.after) do
5397                 func(head)
5398             end
5399         end
5400         if Babel.sea_enabled then
5401             Babel.sea_disc_to_space(head)
5402         end
5403     end,
5404     'Babel.hyphenate')
5405 }
5406 }
5407 \endgroup
5408 \def\bbl@provide@intraspace{%
5409     \bbl@ifunset{\bbl@intsp@language}{}%
5410     {\expandafter\ifx\csname\bbl@intsp@language\endcsname\@empty\else
5411         \bbl@xin@{/c}{\bbl@cl{lbrk}}}%
5412     \ifin@           % cjk
5413         \bbl@cjk_intraspace
5414     \directlua{
5415         Babel = Babel or {}
5416         Babel.locale_props = Babel.locale_props or {}

```

```

5417         Babel.locale_props[\the\localeid].linebreak = 'c'
5418     }%
5419     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@@}%
5420     \ifx\bbl@KVP@intrapenalty\@nil
5421         \bbl@intrapenalty0\@
5422     \fi
5423 \else           % sea
5424     \bbl@seaintraspace
5425     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@@}%
5426     \directlua{
5427         Babel = Babel or {}
5428         Babel.sea_ranges = Babel.sea_ranges or {}
5429         Babel.set_chranges('\bbl@cl{sbc}',
5430                             '\bbl@cl{chrng}')
5431     }%
5432     \ifx\bbl@KVP@intrapenalty\@nil
5433         \bbl@intrapenalty0\@
5434     \fi
5435 \fi
5436 \fi
5437 \ifx\bbl@KVP@intrapenalty\@nil\else
5438     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5439 \fi}}

```

13.6 Arabic justification

```

5440 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5441 \def\bblar@chars{%
5442     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5443     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5444     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5445 \def\bblar@elongated{%
5446     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5447     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5448     0649,064A}
5449 \begingroup
5450 \catcode\_ =11 \catcode\:=11
5451 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5452 \endgroup
5453 \gdef\bbl@arabicjust{%
5454     \let\bbl@arabicjust\relax
5455     \newattribute\bblar@kashida
5456     \bblar@kashida=\z@
5457     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@parsejalt}}%
5458 \directlua{
5459     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5460     Babel.arabic.elong_map[\the\localeid] = {}
5461     luatexbase.add_to_callback('post_linebreak_filter',
5462         Babel.arabic.justify, 'Babel.arabic.justify')
5463     luatexbase.add_to_callback('hpack_filter',
5464         Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5465 }}%
5466 % Save both node lists to make replacement. TODO. Save also widths to
5467 % make computations
5468 \def\bblar@fetchjalt#1#2#3#4{%
5469     \bbl@exp{\bbl@foreach{#1}}{%
5470         \bbl@ifunset\bblar@JE@##1{%
5471             {\setbox\z@\hbox{\char"###1#2}}%
5472             {\setbox\z@\hbox{\char"###1#2}}%

```

```

5473 \directlua{%
5474     local last = nil
5475     for item in node.traverse(tex.box[0].head) do
5476         if item.id == node.id'glyph' and item.char > 0x600 and
5477            not (item.char == 0x200D) then
5478             last = item
5479         end
5480     end
5481     Babel.arabic.#3['##1#4'] = last.char
5482 }}}}
5483 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5484 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5485 % positioning?
5486 \gdef\bbl@parsejalt{%
5487     \ifx\addfontfeature\undefined\else
5488         \bbl@xin@{/e}{/\bbl@cl{lbrk}}}%
5489     \ifin@
5490         \directlua{%
5491             if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5492                 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5493                 tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5494             end
5495         }%
5496     \fi
5497 \fi}
5498 \gdef\bbl@parsejalti{%
5499     \begingroup
5500         \let\bbl@parsejalt\relax % To avoid infinite loop
5501         \edef\bbl@tempb{\fontid\font}%
5502         \bblar@nofswarn
5503         \bblar@fetchjalt\bblar@elongated{}{from}{}%
5504         \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5505         \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5506         \addfontfeature{RawFeature+=jalt}%
5507         % \@namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5508         \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5509         \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5510         \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5511         \directlua{%
5512             for k, v in pairs(Babel.arabic.from) do
5513                 if Babel.arabic.dest[k] and
5514                    not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5515                     Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5516                     [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5517                 end
5518             end
5519         }%
5520     \endgroup}
5521 %
5522 \begingroup
5523 \catcode`#=11
5524 \catcode`~ =11
5525 \directlua{
5526
5527 Babel.arabic = Babel.arabic or {}
5528 Babel.arabic.from = {}
5529 Babel.arabic.dest = {}
5530 Babel.arabic.justify_factor = 0.95
5531 Babel.arabic.justify_enabled = true

```

```

5532
5533 function Babel.arabic.justify(head)
5534   if not Babel.arabic.justify_enabled then return head end
5535   for line in node.traverse_id(node.id'hlist', head) do
5536     Babel.arabic.justify_hlist(head, line)
5537   end
5538   return head
5539 end
5540
5541 function Babel.arabic.justify_hbox(head, gc, size, pack)
5542   local has_inf = false
5543   if Babel.arabic.justify_enabled and pack == 'exactly' then
5544     for n in node.traverse_id(12, head) do
5545       if n.stretch_order > 0 then has_inf = true end
5546     end
5547     if not has_inf then
5548       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5549     end
5550   end
5551   return head
5552 end
5553
5554 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5555   local d, new
5556   local k_list, k_item, pos_inline
5557   local width, width_new, full, k_curr, wt_pos, goal, shift
5558   local subst_done = false
5559   local elong_map = Babel.arabic.elong_map
5560   local last_line
5561   local GLYPH = node.id'glyph'
5562   local KASHIDA = luatexbase.registernumber'bblar@kashida'
5563   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5564
5565   if line == nil then
5566     line = {}
5567     line.glue_sign = 1
5568     line.glue_order = 0
5569     line.head = head
5570     line.shift = 0
5571     line.width = size
5572   end
5573
5574   % Exclude last line. todo. But-- it discards one-word lines, too!
5575   % ? Look for glue = 12:15
5576   if (line.glue_sign == 1 and line.glue_order == 0) then
5577     elongs = {}      % Stores elongated candidates of each line
5578     k_list = {}      % And all letters with kashida
5579     pos_inline = 0   % Not yet used
5580
5581     for n in node.traverse_id(GLYPH, line.head) do
5582       pos_inline = pos_inline + 1 % To find where it is. Not used.
5583
5584       % Elongated glyphs
5585       if elong_map then
5586         local locale = node.get_attribute(n, LOCALE)
5587         if elong_map[locale] and elong_map[locale][n.font] and
5588           elong_map[locale][n.font][n.char] then
5589           table.insert(elongs, {node = n, locale = locale} )
5590           node.set_attribute(n.prev, KASHIDA, 0)

```



```

5591         end
5592     end
5593
5594     % Tatwil
5595     if Babel.kashida_wts then
5596         local k_wt = node.get_attribute(n, KASHIDA)
5597         if k_wt > 0 then % todo. parameter for multi inserts
5598             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5599         end
5600     end
5601
5602 end % of node.traverse_id
5603
5604 if #elongs == 0 and #k_list == 0 then goto next_line end
5605 full = line.width
5606 shift = line.shift
5607 goal = full * Babel.arabic.justify_factor % A bit crude
5608 width = node.dimensions(line.head) % The 'natural' width
5609
5610 % == Elongated ==
5611 % Original idea taken from 'chickenize'
5612 while (#elongs > 0 and width < goal) do
5613     subst_done = true
5614     local x = #elongs
5615     local curr = elongs[x].node
5616     local oldchar = curr.char
5617     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5618     width = node.dimensions(line.head) % Check if the line is too wide
5619     % Substitute back if the line would be too wide and break:
5620     if width > goal then
5621         curr.char = oldchar
5622         break
5623     end
5624     % If continue, pop the just substituted node from the list:
5625     table.remove(elongs, x)
5626 end
5627
5628 % == Tatwil ==
5629 if #k_list == 0 then goto next_line end
5630
5631 width = node.dimensions(line.head) % The 'natural' width
5632 k_curr = #k_list
5633 wt_pos = 1
5634
5635 while width < goal do
5636     subst_done = true
5637     k_item = k_list[k_curr].node
5638     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5639         d = node.copy(k_item)
5640         d.char = 0x0640
5641         line.head, new = node.insert_after(line.head, k_item, d)
5642         width_new = node.dimensions(line.head)
5643         if width > goal or width == width_new then
5644             node.remove(line.head, new) % Better compute before
5645             break
5646         end
5647         width = width_new
5648     end
5649     if k_curr == 1 then

```

```

5650         k_curr = #k_list
5651         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5652     else
5653         k_curr = k_curr - 1
5654     end
5655 end
5656
5657 ::next_line::
5658
5659 % Must take into account marks and ins, see luatex manual.
5660 % Have to be executed only if there are changes. Investigate
5661 % what's going on exactly.
5662 if subst_done and not gc then
5663     d = node.hpack(line.head, full, 'exactly')
5664     d.shift = shift
5665     node.insert_before(head, line, d)
5666     node.remove(head, line)
5667 end
5668 end % if process line
5669 end
5670 }
5671 \endgroup
5672 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5673 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5674 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5675 \DisableBabelHook{babel-fontspec}
5676 <<Font selection>>

```

13.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5677 % TODO - to a lua file
5678 \directlua{
5679 Babel.script_blocks = {
5680   ['dflt'] = {},
5681   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5682               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5683   ['Armn'] = {{0x0530, 0x058F}},
5684   ['Beng'] = {{0x0980, 0x09FF}},
5685   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5686   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5687   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5688               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5689   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5690   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5691               {0xAB00, 0xAB2F}},
5692   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5693   % Don't follow strictly Unicode, which places some Coptic letters in
5694   % the 'Greek and Coptic' block
5695   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5696   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF}},

```

```

5697         {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5698         {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5699         {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5700         {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5701         {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5702 ['Hebr'] = {{0x0590, 0x05FF}},
5703 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5704         {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5705 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5706 ['Knda'] = {{0x0C80, 0x0CFF}},
5707 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5708         {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5709         {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5710 ['Laoo'] = {{0x0E80, 0x0EFF}},
5711 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5712         {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5713         {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5714 ['Mahj'] = {{0x11150, 0x1117F}},
5715 ['Mlym'] = {{0x0D00, 0x0D7F}},
5716 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5717 ['Orya'] = {{0x0B00, 0x0B7F}},
5718 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5719 ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5720 ['Taml'] = {{0x0B80, 0x0BFF}},
5721 ['Telu'] = {{0x0C00, 0x0C7F}},
5722 ['Tfng'] = {{0x2D30, 0x2D7F}},
5723 ['Thai'] = {{0x0E00, 0x0E7F}},
5724 ['Tibt'] = {{0x0F00, 0x0FFF}},
5725 ['Vaii'] = {{0xA500, 0xA63F}},
5726 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5727 }
5728
5729 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5730 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5731 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5732
5733 function Babel.locale_map(head)
5734   if not Babel.locale_mapped then return head end
5735
5736   local LOCALE = luatexbase.registernumber'bb1@attr@locale'
5737   local GLYPH = node.id('glyph')
5738   local inmath = false
5739   local toloc_save
5740   for item in node.traverse(head) do
5741     local toloc
5742     if not inmath and item.id == GLYPH then
5743       % Optimization: build a table with the chars found
5744       if Babel.chr_to_loc[item.char] then
5745         toloc = Babel.chr_to_loc[item.char]
5746       else
5747         for lc, maps in pairs(Babel.loc_to_scr) do
5748           for _, rg in pairs(maps) do
5749             if item.char >= rg[1] and item.char <= rg[2] then
5750               Babel.chr_to_loc[item.char] = lc
5751               toloc = lc
5752               break
5753             end
5754           end
5755         end

```

```

5756     end
5757     % Now, take action, but treat composite chars in a different
5758     % fashion, because they 'inherit' the previous locale. Not yet
5759     % optimized.
5760     if not toloc and
5761         (item.char >= 0x0300 and item.char <= 0x036F) or
5762         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5763         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5764         toloc = toloc_save
5765     end
5766     if toloc and toloc > -1 then
5767         if Babel.locale_props[toloc].lg then
5768             item.lang = Babel.locale_props[toloc].lg
5769             node.set_attribute(item, LOCALE, toloc)
5770         end
5771         if Babel.locale_props[toloc]['/'..item.font] then
5772             item.font = Babel.locale_props[toloc]['/'..item.font]
5773         end
5774         toloc_save = toloc
5775     end
5776     elseif not inmath and item.id == 7 then
5777         item.replace = item.replace and Babel.locale_map(item.replace)
5778         item.pre      = item.pre and Babel.locale_map(item.pre)
5779         item.post     = item.post and Babel.locale_map(item.post)
5780     elseif item.id == node.id'math' then
5781         inmath = (item.subtype == 0)
5782     end
5783 end
5784 return head
5785 end
5786 }

```

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

```

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

```

```

5812 }}
5813 \let\bbl@chprop@bc\bbl@chprop@direction
5814 \def\bbl@chprop@mirror#1{%
5815   \directlua{
5816     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5817     Babel.characters[\the\count@]['m'] = '\number#1'
5818   }}
5819 \let\bbl@chprop@bmg\bbl@chprop@mirror
5820 \def\bbl@chprop@linebreak#1{%
5821   \directlua{
5822     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5823     Babel.cjk_characters[\the\count@]['c'] = '#1'
5824   }}
5825 \let\bbl@chprop@lb\bbl@chprop@linebreak
5826 \def\bbl@chprop@locale#1{%
5827   \directlua{
5828     Babel.chr_to_loc = Babel.chr_to_loc or {}
5829     Babel.chr_to_loc[\the\count@] =
5830       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5831   }}

```

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 discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a `utf8` position. With `first`, the last byte can be the leading byte in a `utf8` sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

5832 \begingroup % TODO - to a lua file
5833 \catcode`\~ = 12
5834 \catcode`\# = 12
5835 \catcode`\% = 12
5836 \catcode`\& = 14
5837 \directlua{
5838   Babel.linebreaking.replacements = {}
5839   Babel.linebreaking.replacements[0] = {} &% pre
5840   Babel.linebreaking.replacements[1] = {} &% post
5841
5842   &% Discretionaries contain strings as nodes
5843   function Babel.str_to_nodes(fn, matches, base)
5844     local n, head, last
5845     if fn == nil then return nil end
5846     for s in string.utfvalues(fn(matches)) do
5847       if base.id == 7 then
5848         base = base.replace
5849       end
5850       n = node.copy(base)
5851       n.char = s
5852       if not head then
5853         head = n
5854       else
5855         last.next = n
5856       end

```

```

5857     last = n
5858 end
5859 return head
5860 end
5861
5862 Babel.fetch_subtext = {}
5863
5864 Babel.ignore_pre_char = function(node)
5865     return (node.lang == \the\l@nohyphenation)
5866 end
5867
5868 %% Merging both functions doesn't seem feasible, because there are too
5869 %% many differences.
5870 Babel.fetch_subtext[0] = function(head)
5871     local word_string = ''
5872     local word_nodes = {}
5873     local lang
5874     local item = head
5875     local inmath = false
5876
5877     while item do
5878
5879         if item.id == 11 then
5880             inmath = (item.subtype == 0)
5881         end
5882
5883         if inmath then
5884             %% pass
5885
5886         elseif item.id == 29 then
5887             local locale = node.get_attribute(item, Babel.attr_locale)
5888
5889             if lang == locale or lang == nil then
5890                 lang = lang or locale
5891                 if Babel.ignore_pre_char(item) then
5892                     word_string = word_string .. Babel.us_char
5893                 else
5894                     word_string = word_string .. unicode.utf8.char(item.char)
5895                 end
5896                 word_nodes[#word_nodes+1] = item
5897             else
5898                 break
5899             end
5900
5901         elseif item.id == 12 and item.subtype == 13 then
5902             word_string = word_string .. ' '
5903             word_nodes[#word_nodes+1] = item
5904
5905             %% Ignore leading unrecognized nodes, too.
5906             elseif word_string ~= '' then
5907                 word_string = word_string .. Babel.us_char
5908                 word_nodes[#word_nodes+1] = item %% Will be ignored
5909             end
5910
5911         item = item.next
5912     end
5913
5914     %% Here and above we remove some trailing chars but not the
5915     %% corresponding nodes. But they aren't accessed.

```

```

5916     if word_string:sub(-1) == ' ' then
5917         word_string = word_string:sub(1,-2)
5918     end
5919     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5920     return word_string, word_nodes, item, lang
5921 end
5922
5923 Babel.fetch_subtext[1] = function(head)
5924     local word_string = ''
5925     local word_nodes = {}
5926     local lang
5927     local item = head
5928     local inmath = false
5929
5930     while item do
5931
5932         if item.id == 11 then
5933             inmath = (item.subtype == 0)
5934         end
5935
5936         if inmath then
5937             && pass
5938
5939         elseif item.id == 29 then
5940             if item.lang == lang or lang == nil then
5941                 if (item.char ~= 124) and (item.char ~= 61) then && not =, not |
5942                     lang = lang or item.lang
5943                     word_string = word_string .. unicode.utf8.char(item.char)
5944                     word_nodes[#word_nodes+1] = item
5945                 end
5946             else
5947                 break
5948             end
5949
5950         elseif item.id == 7 and item.subtype == 2 then
5951             word_string = word_string .. '='
5952             word_nodes[#word_nodes+1] = item
5953
5954         elseif item.id == 7 and item.subtype == 3 then
5955             word_string = word_string .. '|'
5956             word_nodes[#word_nodes+1] = item
5957
5958             && (1) Go to next word if nothing was found, and (2) implicitly
5959             && remove leading USs.
5960         elseif word_string == '' then
5961             && pass
5962
5963             && This is the responsible for splitting by words.
5964         elseif (item.id == 12 and item.subtype == 13) then
5965             break
5966
5967         else
5968             word_string = word_string .. Babel.us_char
5969             word_nodes[#word_nodes+1] = item && Will be ignored
5970         end
5971
5972         item = item.next
5973     end
5974

```

```

5975     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5976     return word_string, word_nodes, item, lang
5977 end
5978
5979 function Babel.pre_hyphenate_replace(head)
5980     Babel.hyphenate_replace(head, 0)
5981 end
5982
5983 function Babel.post_hyphenate_replace(head)
5984     Babel.hyphenate_replace(head, 1)
5985 end
5986
5987 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5988     local ss = ''
5989     for pp = 1, 40 do
5990         if wn[pp] then
5991             if wn[pp].id == 29 then
5992                 ss = ss .. unicode.utf8.char(wn[pp].char)
5993             else
5994                 ss = ss .. '{' .. wn[pp].id .. '}'
5995             end
5996         end
5997     end
5998     print('nod', ss)
5999     print('lst_m',
6000         string.rep(' ', unicode.utf8.len(
6001             string.sub(w, 1, last_match))-1) .. '>')
6002     print('str', w)
6003     print('sc', string.rep(' ', sc-1) .. '^')
6004     if first == last then
6005         print('f=l', string.rep(' ', first-1) .. '!!')
6006     else
6007         print('f/l', string.rep(' ', first-1) .. '[' ..
6008             string.rep(' ', last-first-1) .. ']')
6009     end
6010 end
6011
6012 Babel.us_char = string.char(31)
6013
6014 function Babel.hyphenate_replace(head, mode)
6015     local u = unicode.utf8
6016     local lbkr = Babel.linebreaking.replacements[mode]
6017
6018     local word_head = head
6019
6020     while true do    &% for each subtext block
6021
6022         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6023
6024         if Babel.debug then
6025             print()
6026             print((mode == 0) and '@@@@<' or '@@@@>', w)
6027         end
6028
6029         if nw == nil and w == '' then break end
6030
6031         if not lang then goto next end
6032         if not lbkr[lang] then goto next end
6033

```



```

6034    %% For each saved (pre|post)hyphenation. TODO. Reconsider how
6035    %% loops are nested.
6036    for k=1, #lbr[lang] do
6037        local p = lbr[lang][k].pattern
6038        local r = lbr[lang][k].replace
6039
6040        if Babel.debug then
6041            print('*****', p, mode)
6042        end
6043
6044        %% This variable is set in some cases below to the first *byte*
6045        %% after the match, either as found by u.match (faster) or the
6046        %% computed position based on sc if w has changed.
6047        local last_match = 0
6048        local step = 0
6049
6050        %% For every match.
6051        while true do
6052            if Babel.debug then
6053                print('====')
6054            end
6055            local new    %% used when inserting and removing nodes
6056
6057            local matches = { u.match(w, p, last_match) }
6058
6059            if #matches < 2 then break end
6060
6061            %% Get and remove empty captures (with ()'s, which return a
6062            %% number with the position), and keep actual captures
6063            %% (from (...)), if any, in matches.
6064            local first = table.remove(matches, 1)
6065            local last  = table.remove(matches, #matches)
6066            %% Non re-fetched substrings may contain \31, which separates
6067            %% subsubstrings.
6068            if string.find(w:sub(first, last-1), Babel.us_char) then break end
6069
6070            local save_last = last %% with A()BC()D, points to D
6071
6072            %% Fix offsets, from bytes to unicode. Explained above.
6073            first = u.len(w:sub(1, first-1)) + 1
6074            last  = u.len(w:sub(1, last-1)) %% now last points to C
6075
6076            %% This loop stores in n small table the nodes
6077            %% corresponding to the pattern. Used by 'data' to provide a
6078            %% predictable behavior with 'insert' (now w_nodes is modified on
6079            %% the fly), and also access to 'remove'd nodes.
6080            local sc = first-1          %% Used below, too
6081            local data_nodes = {}
6082
6083            for q = 1, last-first+1 do
6084                data_nodes[q] = w_nodes[sc+q]
6085            end
6086
6087            %% This loop traverses the matched substring and takes the
6088            %% corresponding action stored in the replacement list.
6089            %% sc = the position in substr nodes / string
6090            %% rc = the replacement table index
6091            local rc = 0
6092

```

```

6093 while rc < last-first+1 do %% for each replacement
6094     if Babel.debug then
6095         print('.....', rc + 1)
6096     end
6097     sc = sc + 1
6098     rc = rc + 1
6099
6100     if Babel.debug then
6101         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6102         local ss = ''
6103         for itt in node.traverse(head) do
6104             if itt.id == 29 then
6105                 ss = ss .. unicode.utf8.char(itt.char)
6106             else
6107                 ss = ss .. '{' .. itt.id .. '}'
6108             end
6109         end
6110         print('*****', ss)
6111     end
6112
6113     local crep = r[rc]
6114     local item = w_nodes[sc]
6115     local item_base = item
6116     local placeholder = Babel.us_char
6117     local d
6118
6119     if crep and crep.data then
6120         item_base = data_nodes[crep.data]
6121     end
6122
6123     if crep then
6124         step = crep.step or 0
6125     end
6126
6127     if crep and next(crep) == nil then %% = {}
6128         last_match = save_last    %% Optimization
6129         goto next
6130
6131     elseif crep == nil or crep.remove then
6132         node.remove(head, item)
6133         table.remove(w_nodes, sc)
6134         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6135         sc = sc - 1    %% Nothing has been inserted.
6136         last_match = utf8.offset(w, sc+1+step)
6137         goto next
6138
6139     elseif crep and crep.kashida then %% Experimental
6140         node.set_attribute(item,
6141             luatexbase.registernumber'bb1ar@kashida',
6142             crep.kashida)
6143         last_match = utf8.offset(w, sc+1+step)
6144         goto next
6145
6146     elseif crep and crep.string then
6147         local str = crep.string(matches)
6148         if str == '' then    %% Gather with nil
6149             node.remove(head, item)
6150             table.remove(w_nodes, sc)
6151         end

```

```

6152         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6153         sc = sc - 1  %% Nothing has been inserted.
6154     else
6155         local loop_first = true
6156         for s in string.utfvalues(str) do
6157             d = node.copy(item_base)
6158             d.char = s
6159             if loop_first then
6160                 loop_first = false
6161                 head, new = node.insert_before(head, item, d)
6162                 if sc == 1 then
6163                     word_head = head
6164                 end
6165                 w_nodes[sc] = d
6166                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6167             else
6168                 sc = sc + 1
6169                 head, new = node.insert_before(head, item, d)
6170                 table.insert(w_nodes, sc, new)
6171                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6172             end
6173             if Babel.debug then
6174                 print('.....', 'str')
6175                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6176             end
6177             end  %% for
6178             node.remove(head, item)
6179         end  %% if ''
6180         last_match = utf8.offset(w, sc+1+step)
6181         goto next
6182
6183     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6184         d = node.new(7, 0)  %% (disc, discretionary)
6185         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6186         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6187         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6188         d.attr = item_base.attr
6189         if crep.pre == nil then  %% TeXbook p96
6190             d.penalty = crep.penalty or tex.hyphenpenalty
6191         else
6192             d.penalty = crep.penalty or tex.exhyphenpenalty
6193         end
6194         placeholder = '|'
6195         head, new = node.insert_before(head, item, d)
6196
6197     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6198         %% ERROR
6199
6200     elseif crep and crep.penalty then
6201         d = node.new(14, 0)  %% (penalty, userpenalty)
6202         d.attr = item_base.attr
6203         d.penalty = crep.penalty
6204         head, new = node.insert_before(head, item, d)
6205
6206     elseif crep and crep.space then
6207         %% 655360 = 10 pt = 10 * 65536 sp
6208         d = node.new(12, 13)  %% (glue, spaceskip)
6209         local quad = font.getfont(item_base.font).size or 655360
6210         node.setglue(d, crep.space[1] * quad,

```

```

6211             crep.space[2] * quad,
6212             crep.space[3] * quad)
6213         if mode == 0 then
6214             placeholder = ' '
6215         end
6216         head, new = node.insert_before(head, item, d)
6217
6218     elseif crep and crep.spacefactor then
6219         d = node.new(12, 13)      %% (glue, spaceskip)
6220         local base_font = font.getfont(item_base.font)
6221         node.setglue(d,
6222             crep.spacefactor[1] * base_font.parameters['space'],
6223             crep.spacefactor[2] * base_font.parameters['space_stretch'],
6224             crep.spacefactor[3] * base_font.parameters['space_shrink'])
6225         if mode == 0 then
6226             placeholder = ' '
6227         end
6228         head, new = node.insert_before(head, item, d)
6229
6230     elseif mode == 0 and crep and crep.space then
6231         %% ERROR
6232
6233     end    %% ie replacement cases
6234
6235     %% Shared by disc, space and penalty.
6236     if sc == 1 then
6237         word_head = head
6238     end
6239     if crep.insert then
6240         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6241         table.insert(w_nodes, sc, new)
6242         last = last + 1
6243     else
6244         w_nodes[sc] = d
6245         node.remove(head, item)
6246         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6247     end
6248
6249     last_match = utf8.offset(w, sc+1+step)
6250
6251     ::next::
6252
6253 end    %% for each replacement
6254
6255 if Babel.debug then
6256     print('.....', '/')
6257     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6258 end
6259
6260 end    %% for match
6261
6262 end    %% for patterns
6263
6264 ::next::
6265 word_head = nw
6266 end    %% for substring
6267 return head
6268 end
6269

```

```

6270  &% This table stores capture maps, numbered consecutively
6271  Babel.capture_maps = {}
6272
6273  &% The following functions belong to the next macro
6274  function Babel.capture_func(key, cap)
6275      local ret = "[[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..[" .. "]"
6276      local cnt
6277      local u = unicode.utf8
6278      ret, cnt = ret:gsub('{{[0-9]}|([^\]|+)|(.-)}', Babel.capture_func_map)
6279      if cnt == 0 then
6280          ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6281              function (n)
6282                  return u.char(tonumber(n, 16))
6283              end)
6284      end
6285      ret = ret:gsub("%[%[%]]%.", '')
6286      ret = ret:gsub("%.%.%[%[%]]%", '')
6287      return key .. [=function(m) return ]] .. ret .. [=end]]
6288  end
6289
6290  function Babel.capt_map(from, mapno)
6291      return Babel.capture_maps[mapno][from] or from
6292  end
6293
6294  &% Handle the {n|abc|ABC} syntax in captures
6295  function Babel.capture_func_map(capno, from, to)
6296      local u = unicode.utf8
6297      from = u.gsub(from, '{{(%x%x%x%x+)}',
6298          function (n)
6299              return u.char(tonumber(n, 16))
6300          end)
6301      to = u.gsub(to, '{{(%x%x%x%x+)}',
6302          function (n)
6303              return u.char(tonumber(n, 16))
6304          end)
6305      local froms = {}
6306      for s in string.utfcharacters(from) do
6307          table.insert(froms, s)
6308      end
6309      local cnt = 1
6310      table.insert(Babel.capture_maps, {})
6311      local mlen = table.getn(Babel.capture_maps)
6312      for s in string.utfcharacters(to) do
6313          Babel.capture_maps[mlen][froms[cnt]] = s
6314          cnt = cnt + 1
6315      end
6316      return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6317          (mlen) .. ").." .. "[["
6318  end
6319
6320  &% Create/Extend reversed sorted list of kashida weights:
6321  function Babel.capture_kashida(key, wt)
6322      wt = tonumber(wt)
6323      if Babel.kashida_wts then
6324          for p, q in ipairs(Babel.kashida_wts) do
6325              if wt == q then
6326                  break
6327              elseif wt > q then
6328                  table.insert(Babel.kashida_wts, p, wt)

```

```

6329         break
6330     elseif table.getn(Babel.kashida_wts) == p then
6331         table.insert(Babel.kashida_wts, wt)
6332     end
6333 end
6334 else
6335     Babel.kashida_wts = { wt }
6336 end
6337 return 'kashida = ' .. wt
6338 end
6339 }

```

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).

```

6340 \catcode`\#=6
6341 \gdef\babelposthyphenation#1#2#3{&
6342   \bbl@activateposthyphen
6343   \begingroup
6344     \def\babeltempa{\bbl@add@list\babeltempb}&
6345     \let\babeltempb\@empty
6346     \def\bbl@tempa{#3}& TODO. Ugly trick to preserve {}:
6347     \bbl@replace\bbl@tempa{,}{ ,}&
6348     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&
6349       \bbl@ifsamestring{##1}{remove}&
6350       {\bbl@add@list\babeltempb{nil}}&
6351       {\directlua{
6352         local rep = {[#1]=]
6353         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6354         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6355         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6356         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6357         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6358         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
6359         tex.print([[string\babeltempa{[] .. rep .. [[]]])
6360       ]}}&
6361     \directlua{
6362       local lbkr = Babel.linebreaking.replacements[1]
6363       local u = unicode.utf8
6364       local id = \the\csname l@#1\endcsname
6365       & Convert pattern:
6366       local patt = string.gsub([=[#2]=], '%s', '')
6367       if not u.find(patt, '()', nil, true) then
6368         patt = '()' .. patt .. '()'
6369       end
6370       patt = string.gsub(patt, '%(%)^', '^()')
6371       patt = string.gsub(patt, '%$(%)', '()$')
6372       patt = u.gsub(patt, '{(.)}',
6373         function (n)
6374           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6375         end)
6376       patt = u.gsub(patt, '{(%x%x%x%x+)}',
6377         function (n)

```

```

6378         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6379     end)
6380     lbr[id] = lbr[id] or {}
6381     table.insert(lbr[id], { pattern = patt, replace = { \babeltempb } })
6382 }&%
6383 \endgroup}
6384 % TODO. Copy paste pattern.
6385 \gdef\babelprehyphenation#1#2#3{&%
6386   \bbl@activateprehyphen
6387   \begin{group}
6388     \def\babeltempa{\bbl@add@list\babeltempb}&%
6389     \let\babeltempb\@empty
6390     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6391     \bbl@replace\bbl@tempa{,}{ ,}&%
6392     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6393       \bbl@ifsamestring{##1}{remove}&%
6394       {\bbl@add@list\babeltempb{nil}}&%
6395       {\directlua{
6396         local rep = [=#[#1]=]
6397         rep = rep.gsub('^%s*(remove)%s$', 'remove = true')
6398         rep = rep.gsub('^%s*(insert)%s$', 'insert = true, ')
6399         rep = rep.gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6400         rep = rep.gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6401           'space = {' .. '%2, %3, %4' .. '}')
6402         rep = rep.gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6403           'spacefactor = {' .. '%2, %3, %4' .. '}')
6404         rep = rep.gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6405         tex.print([[\\string\babeltempa{}}] .. rep .. [[{}]])
6406       }}&%
6407     \directlua{
6408       local lbr = Babel.linebreaking.replacements[0]
6409       local u = unicode.utf8
6410       local id = \the\csname bbl@id@@#1\endcsname
6411       &% Convert pattern:
6412       local patt = string.gsub(=[#2]=, '%s', '')
6413       local patt = string.gsub(patt, '|', ' ')
6414       if not u.find(patt, '()', nil, true) then
6415         patt = '()' .. patt .. '()'
6416       end
6417       &% patt = string.gsub(patt, '%(%)^', '^()')
6418       &% patt = string.gsub(patt, '([^\%])%$%$', '%1()$')
6419       patt = u.gsub(patt, '{(.)}',
6420         function (n)
6421           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6422         end)
6423       patt = u.gsub(patt, '{(%x%x%x%x+)}',
6424         function (n)
6425           return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6426         end)
6427       lbr[id] = lbr[id] or {}
6428       table.insert(lbr[id], { pattern = patt, replace = { \babeltempb } })
6429     }&%
6430   \endgroup}
6431 \endgroup
6432 \def\bbl@activateposthyphen{%
6433   \let\bbl@activateposthyphen\relax
6434   \directlua{
6435     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6436   }}

```

```

6437 \def\bbl@activateprehyphen{%
6438   \let\bbl@activateprehyphen\relax
6439   \directlua{
6440     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6441   }}

```

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.

```

6442 \bbl@trace{Redefinitions for bidi layout}
6443 \ifx\@eqnnum\undefined\else
6444   \ifx\bbl@attr@dir\undefined\else
6445     \edef\@eqnnum{%
6446       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6447       \unexpanded\expandafter{\@eqnnum}}%
6448   \fi
6449 \fi
6450 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
6451 \ifnum\bbl@bidimode>\z@
6452   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6453     \bbl@exp{%
6454       \mathdir\the\bodydir
6455       #1% Once entered in math, set boxes to restore values
6456       \<ifmmode>%
6457       \everyvbox{%
6458         \the\everyvbox
6459         \bodydir\the\bodydir
6460         \mathdir\the\mathdir
6461         \everyhbox{\the\everyhbox}%
6462         \everyvbox{\the\everyvbox}}%
6463       \everyhbox{%
6464         \the\everyhbox
6465         \bodydir\the\bodydir
6466         \mathdir\the\mathdir
6467         \everyhbox{\the\everyhbox}%
6468         \everyvbox{\the\everyvbox}}%
6469       \<fi>}}%
6470   \def\@hangfrom#1{%
6471     \setbox\@tempboxa\hbox{#1}%
6472     \hangindent\wd\@tempboxa
6473     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6474       \shapemode\@ne
6475     \fi
6476     \noindent\box\@tempboxa}
6477 \fi
6478 \IfBabelLayout{tabular}
6479   {\let\bbl@OL@tabular\@tabular

```



```

6480 \bbl@replace\@tabular{$}\bbl@nextfake$}%
6481 \let\bbl@NL@@tabular\@tabular
6482 \AtBeginDocument{%
6483   \ifx\bbl@NL@@tabular\@tabular\else
6484     \bbl@replace\@tabular{$}\bbl@nextfake$}%
6485     \let\bbl@NL@@tabular\@tabular
6486   \fi}}
6487 {}
6488 \IfBabelLayout{lists}
6489 {\let\bbl@OL@list\list
6490  \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6491  \let\bbl@NL@list\list
6492  \def\bbl@listparshape#1#2#3{%
6493    \parshape #1 #2 #3 %
6494    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6495      \shapemode\tw@
6496    \fi}}
6497 {}
6498 \IfBabelLayout{graphics}
6499 {\let\bbl@pictresetdir\relax
6500  \def\bbl@pictsetdir#1{%
6501    \ifcase\bbl@thetextdir
6502      \let\bbl@pictresetdir\relax
6503    \else
6504      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6505        \or\textdir TLT
6506        \else\bodydir TLT \textdir TLT
6507      \fi
6508      % \(\text|par)dir required in pgf:
6509      \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6510    \fi}%
6511  \ifx\AddToHook\undefined\else
6512    \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6513    \directlua{
6514      Babel.get_picture_dir = true
6515      Babel.picture_has_bidi = 0
6516      function Babel.picture_dir (head)
6517        if not Babel.get_picture_dir then return head end
6518        for item in node.traverse(head) do
6519          if item.id == node.id'glyph' then
6520            local itemchar = item.char
6521            % TODO. Copypaste pattern from Babel.bidi (-r)
6522            local chardata = Babel.characters[itemchar]
6523            local dir = chardata and chardata.d or nil
6524            if not dir then
6525              for nn, et in ipairs(Babel.ranges) do
6526                if itemchar < et[1] then
6527                  break
6528                elseif itemchar <= et[2] then
6529                  dir = et[3]
6530                  break
6531                end
6532              end
6533            end
6534            if dir and (dir == 'al' or dir == 'r') then
6535              Babel.picture_has_bidi = 1
6536            end
6537          end
6538        end

```

```

6539         return head
6540     end
6541     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6542         "Babel.picture_dir")
6543 }%
6544 \AtBeginDocument{%
6545     \long\def\put(#1,#2)#3{%
6546         \@killglue
6547         % Try:
6548         \ifx\bbl@pictresetdir\relax
6549             \def\bbl@tempc{0}%
6550         \else
6551             \directlua{
6552                 Babel.get_picture_dir = true
6553                 Babel.picture_has_bidi = 0
6554             }%
6555             \setbox\z@\hb@xt@\z@{%
6556                 \@defaultunitsset\@tempdimc{#1}\unitlength
6557                 \kern\@tempdimc
6558                 #3\hss}%
6559             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6560         \fi
6561         % Do:
6562         \@defaultunitsset\@tempdimc{#2}\unitlength
6563         \raise\@tempdimc\hb@xt@\z@{%
6564             \@defaultunitsset\@tempdimc{#1}\unitlength
6565             \kern\@tempdimc
6566             {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6567         \ignorespaces}%
6568         \MakeRobust\put}%
6569 \fi
6570 \AtBeginDocument
6571 {\ifx\tikz@atbegin@node\@undefined\else
6572     \ifx\AddToHook\@undefined\else % TODO. Still tentative.
6573         \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6574         \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6575     \fi
6576     \let\bbl@OL@pgfpicture\pgfpicture
6577     \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
6578     {\bbl@pictsetdir\z@\pgfpicturetrue}%
6579     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6580     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6581     \bbl@sreplace\tikz{\begingroup}%
6582     {\begingroup\bbl@pictsetdir\tw@}%
6583 \fi
6584 \ifx\AddToHook\@undefined\else
6585     \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6586 \fi
6587 }}
6588 {}

```

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.

```

6589 \IfBabelLayout{counters}%
6590 {\let\bbl@OL@@textsuperscript\textsuperscript
6591  \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6592  \let\bbl@latinarabic=\@arabic
6593  \let\bbl@OL@@arabic\@arabic

```

```

6594 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6595 \@ifpackagewith{babel}{bidi=default}%
6596 {\let\bbl@asciroman=\@roman
6597 \let\bbl@OL@roman=\@roman
6598 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6599 \let\bbl@asciRoman=\@Roman
6600 \let\bbl@OL@roman=\@Roman
6601 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
6602 \let\bbl@OL@labelenumii\labelenumii
6603 \def\labelenumii{}\theenumii}%
6604 \let\bbl@OL@p@enumiii\p@enumiii
6605 \def\p@enumiii{\p@enumii)\theenumii{}}{}{}}
6606 <<Footnote changes>>
6607 \IfBabelLayout{footnotes}%
6608 {\let\bbl@OL@footnote\footnote
6609 \BabelFootnote\footnote\languagename{}}{}%
6610 \BabelFootnote\localfootnote\languagename{}}{}%
6611 \BabelFootnote\mainfootnote{}}{}{}}
6612 {}

```

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.

```

6613 \IfBabelLayout{extras}%
6614 {\let\bbl@OL@underline\underline
6615 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6616 \let\bbl@OL@LaTeX2e\LaTeX2e
6617 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6618 \if b\expandafter\car\@series\@nil\boldmath\fi
6619 \babelsublr{%
6620 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6621 {}
6622 </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):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```

6623 (*basic-r)
6624 Babel = Babel or {}
6625
6626 Babel.bidi_enabled = true
6627
6628 require('babel-data-bidi.lua')
6629
6630 local characters = Babel.characters
6631 local ranges = Babel.ranges
6632
6633 local DIR = node.id("dir")
6634
6635 local function dir_mark(head, from, to, outer)
6636   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6637   local d = node.new(DIR)
6638   d.dir = '+' .. dir
6639   node.insert_before(head, from, d)
6640   d = node.new(DIR)
6641   d.dir = '-' .. dir
6642   node.insert_after(head, to, d)
6643 end
6644
6645 function Babel.bidi(head, ispar)
6646   local first_n, last_n      -- first and last char with nums
6647   local last_es              -- an auxiliary 'last' used with nums
6648   local first_d, last_d      -- first and last char in L/R block
6649   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):

```

6650   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6651   local strong_lr = (strong == 'l') and 'l' or 'r'
6652   local outer = strong
6653
6654   local new_dir = false
6655   local first_dir = false
6656   local inmath = false
6657
6658   local last_lr
6659
6660   local type_n = ''
6661
6662   for item in node.traverse(head) do
6663
6664     -- three cases: glyph, dir, otherwise

```

```

6665   if item.id == node.id'glyph'
6666     or (item.id == 7 and item.subtype == 2) then
6667
6668     local itemchar
6669     if item.id == 7 and item.subtype == 2 then
6670       itemchar = item.replace.char
6671     else
6672       itemchar = item.char
6673     end
6674     local chardata = characters[itemchar]
6675     dir = chardata and chardata.d or nil
6676     if not dir then
6677       for nn, et in ipairs(ranges) do
6678         if itemchar < et[1] then
6679           break
6680         elseif itemchar <= et[2] then
6681           dir = et[3]
6682           break
6683         end
6684       end
6685     end
6686     dir = dir or 'l'
6687     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).

```

6688   if new_dir then
6689     attr_dir = 0
6690     for at in node.traverse(item.attr) do
6691       if at.number == luatexbase.registernumber'bbl@attr@dir' then
6692         attr_dir = at.value % 3
6693       end
6694     end
6695     if attr_dir == 1 then
6696       strong = 'r'
6697     elseif attr_dir == 2 then
6698       strong = 'al'
6699     else
6700       strong = 'l'
6701     end
6702     strong_lr = (strong == 'l') and 'l' or 'r'
6703     outer = strong_lr
6704     new_dir = false
6705   end
6706
6707   if dir == 'nsm' then dir = strong end -- W1

```

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

```

6708   dir_real = dir -- We need dir_real to set strong below
6709   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:

```

6710   if strong == 'al' then
6711     if dir == 'en' then dir = 'an' end -- W2
6712     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6

```

```

6713         strong_lr = 'r'                                -- W3
6714     end

```

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

```

6715     elseif item.id == node.id'dir' and not inmath then
6716         new_dir = true
6717         dir = nil
6718     elseif item.id == node.id'math' then
6719         inmath = (item.subtype == 0)
6720     else
6721         dir = nil          -- Not a char
6722     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>.

```

6723     if dir == 'en' or dir == 'an' or dir == 'et' then
6724         if dir ~= 'et' then
6725             type_n = dir
6726         end
6727         first_n = first_n or item
6728         last_n = last_es or item
6729         last_es = nil
6730     elseif dir == 'es' and last_n then -- W3+W6
6731         last_es = item
6732     elseif dir == 'cs' then          -- it's right - do nothing
6733     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6734         if strong_lr == 'r' and type_n ~= '' then
6735             dir_mark(head, first_n, last_n, 'r')
6736         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6737             dir_mark(head, first_n, last_n, 'r')
6738             dir_mark(head, first_d, last_d, outer)
6739             first_d, last_d = nil, nil
6740         elseif strong_lr == 'l' and type_n ~= '' then
6741             last_d = last_n
6742         end
6743         type_n = ''
6744         first_n, last_n = nil, nil
6745     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:

```

6746     if dir == 'l' or dir == 'r' then
6747         if dir ~= outer then
6748             first_d = first_d or item
6749             last_d = item
6750         elseif first_d and dir ~= strong_lr then
6751             dir_mark(head, first_d, last_d, outer)
6752             first_d, last_d = nil, nil
6753         end
6754     end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resptly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

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

```
6755   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6756       item.char = characters[item.char] and
6757           characters[item.char].m or item.char
6758   elseif (dir or new_dir) and last_lr ~= item then
6759       local mir = outer .. strong_lr .. (dir or outer)
6760       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6761           for ch in node.traverse(node.next(last_lr)) do
6762               if ch == item then break end
6763               if ch.id == node.id'glyph' and characters[ch.char] then
6764                   ch.char = characters[ch.char].m or ch.char
6765               end
6766           end
6767       end
6768   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).

```
6769   if dir == 'l' or dir == 'r' then
6770       last_lr = item
6771       strong = dir_real           -- Don't search back - best save now
6772       strong_lr = (strong == 'l') and 'l' or 'r'
6773   elseif new_dir then
6774       last_lr = nil
6775   end
6776 end
```

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

```
6777   if last_lr and outer == 'r' then
6778       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6779           if characters[ch.char] then
6780               ch.char = characters[ch.char].m or ch.char
6781           end
6782       end
6783   end
6784   if first_n then
6785       dir_mark(head, first_n, last_n, outer)
6786   end
6787   if first_d then
6788       dir_mark(head, first_d, last_d, outer)
6789   end
```

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

```
6790   return node.prev(head) or head
6791 end
6792 </basic-r>
```

And here the Lua code for bidi=basic:

```
6793 (*basic)
6794 Babel = Babel or {}
6795
6796 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6797
6798 Babel.fontmap = Babel.fontmap or {}
6799 Babel.fontmap[0] = {}      -- l
6800 Babel.fontmap[1] = {}      -- r
6801 Babel.fontmap[2] = {}      -- al/an
6802
```

```

6803 Babel.bidi_enabled = true
6804 Babel.mirroring_enabled = true
6805
6806 require('babel-data-bidi.lua')
6807
6808 local characters = Babel.characters
6809 local ranges = Babel.ranges
6810
6811 local DIR = node.id('dir')
6812 local GLYPH = node.id('glyph')
6813
6814 local function insert_implicit(head, state, outer)
6815   local new_state = state
6816   if state.sim and state.eim and state.sim ~= state.eim then
6817     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6818     local d = node.new(DIR)
6819     d.dir = '+' .. dir
6820     node.insert_before(head, state.sim, d)
6821     local d = node.new(DIR)
6822     d.dir = '-' .. dir
6823     node.insert_after(head, state.eim, d)
6824   end
6825   new_state.sim, new_state.eim = nil, nil
6826   return head, new_state
6827 end
6828
6829 local function insert_numeric(head, state)
6830   local new
6831   local new_state = state
6832   if state.san and state.ean and state.san ~= state.ean then
6833     local d = node.new(DIR)
6834     d.dir = '+TLT'
6835     _, new = node.insert_before(head, state.san, d)
6836     if state.san == state.sim then state.sim = new end
6837     local d = node.new(DIR)
6838     d.dir = '-TLT'
6839     _, new = node.insert_after(head, state.ean, d)
6840     if state.ean == state.eim then state.eim = new end
6841   end
6842   new_state.san, new_state.ean = nil, nil
6843   return head, new_state
6844 end
6845
6846 -- TODO - \hbox with an explicit dir can lead to wrong results
6847 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6848 -- was s made to improve the situation, but the problem is the 3-dir
6849 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6850 -- well.
6851
6852 function Babel.bidi(head, ispar, hdir)
6853   local d -- d is used mainly for computations in a loop
6854   local prev_d = ''
6855   local new_d = false
6856
6857   local nodes = {}
6858   local outer_first = nil
6859   local inmath = false
6860
6861   local glue_d = nil

```



```

6862 local glue_i = nil
6863
6864 local has_en = false
6865 local first_et = nil
6866
6867 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6868
6869 local save_outer
6870 local temp = node.get_attribute(head, ATDIR)
6871 if temp then
6872     temp = temp % 3
6873     save_outer = (temp == 0 and 'l') or
6874                 (temp == 1 and 'r') or
6875                 (temp == 2 and 'al')
6876 elseif ispar then -- Or error? Shouldn't happen
6877     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6878 else -- Or error? Shouldn't happen
6879     save_outer = ('TRT' == hdir) and 'r' or 'l'
6880 end
6881 -- when the callback is called, we are just _after_ the box,
6882 -- and the textdir is that of the surrounding text
6883 -- if not ispar and hdir ~= tex.textdir then
6884 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6885 -- end
6886 local outer = save_outer
6887 local last = outer
6888 -- 'al' is only taken into account in the first, current loop
6889 if save_outer == 'al' then save_outer = 'r' end
6890
6891 local fontmap = Babel.fontmap
6892
6893 for item in node.traverse(head) do
6894
6895     -- In what follows, #node is the last (previous) node, because the
6896     -- current one is not added until we start processing the neutrals.
6897
6898     -- three cases: glyph, dir, otherwise
6899     if item.id == GLYPH
6900         or (item.id == 7 and item.subtype == 2) then
6901
6902         local d_font = nil
6903         local item_r
6904         if item.id == 7 and item.subtype == 2 then
6905             item_r = item.replace -- automatic discs have just 1 glyph
6906         else
6907             item_r = item
6908         end
6909         local chardata = characters[item_r.char]
6910         d = chardata and chardata.d or nil
6911         if not d or d == 'nsm' then
6912             for nn, et in ipairs(ranges) do
6913                 if item_r.char < et[1] then
6914                     break
6915                 elseif item_r.char <= et[2] then
6916                     if not d then d = et[3]
6917                     elseif d == 'nsm' then d_font = et[3]
6918                     end
6919                     break
6920                 end

```

```

6921         end
6922     end
6923     d = d or 'l'
6924
6925     -- A short 'pause' in bidi for mapfont
6926     d_font = d_font or d
6927     d_font = (d_font == 'l' and 0) or
6928             (d_font == 'nsm' and 0) or
6929             (d_font == 'r' and 1) or
6930             (d_font == 'al' and 2) or
6931             (d_font == 'an' and 2) or nil
6932     if d_font and fontmap and fontmap[d_font][item_r.font] then
6933         item_r.font = fontmap[d_font][item_r.font]
6934     end
6935
6936     if new_d then
6937         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6938         if inmath then
6939             attr_d = 0
6940         else
6941             attr_d = node.get_attribute(item, ATDIR)
6942             attr_d = attr_d % 3
6943         end
6944         if attr_d == 1 then
6945             outer_first = 'r'
6946             last = 'r'
6947         elseif attr_d == 2 then
6948             outer_first = 'r'
6949             last = 'al'
6950         else
6951             outer_first = 'l'
6952             last = 'l'
6953         end
6954         outer = last
6955         has_en = false
6956         first_et = nil
6957         new_d = false
6958     end
6959
6960     if glue_d then
6961         if (d == 'l' and 'l' or 'r') ~= glue_d then
6962             table.insert(nodes, {glue_i, 'on', nil})
6963         end
6964         glue_d = nil
6965         glue_i = nil
6966     end
6967
6968     elseif item.id == DIR then
6969         d = nil
6970         new_d = true
6971
6972     elseif item.id == node.id'glue' and item.subtype == 13 then
6973         glue_d = d
6974         glue_i = item
6975         d = nil
6976
6977     elseif item.id == node.id'math' then
6978         inmath = (item.subtype == 0)
6979

```

```

6980     else
6981         d = nil
6982     end
6983
6984     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6985     if last == 'al' and d == 'en' then
6986         d = 'an'          -- W3
6987     elseif last == 'al' and (d == 'et' or d == 'es') then
6988         d = 'on'          -- W6
6989     end
6990
6991     -- EN + CS/ES + EN      -- W4
6992     if d == 'en' and #nodes >= 2 then
6993         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6994             and nodes[#nodes-1][2] == 'en' then
6995             nodes[#nodes][2] = 'en'
6996         end
6997     end
6998
6999     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7000     if d == 'an' and #nodes >= 2 then
7001         if (nodes[#nodes][2] == 'cs')
7002             and nodes[#nodes-1][2] == 'an' then
7003             nodes[#nodes][2] = 'an'
7004         end
7005     end
7006
7007     -- ET/EN                  -- W5 + W7->l / W6->on
7008     if d == 'et' then
7009         first_et = first_et or (#nodes + 1)
7010     elseif d == 'en' then
7011         has_en = true
7012         first_et = first_et or (#nodes + 1)
7013     elseif first_et then      -- d may be nil here !
7014         if has_en then
7015             if last == 'l' then
7016                 temp = 'l'    -- W7
7017             else
7018                 temp = 'en'   -- W5
7019             end
7020         else
7021             temp = 'on'       -- W6
7022         end
7023         for e = first_et, #nodes do
7024             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7025         end
7026         first_et = nil
7027         has_en = false
7028     end
7029
7030     -- Force mathdir in math if ON (currently works as expected only
7031     -- with 'l')
7032     if inmath and d == 'on' then
7033         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7034     end
7035
7036     if d then
7037         if d == 'al' then
7038             d = 'r'

```

```

7039         last = 'al'
7040     elseif d == 'l' or d == 'r' then
7041         last = d
7042     end
7043     prev_d = d
7044     table.insert(nodes, {item, d, outer_first})
7045 end
7046
7047 outer_first = nil
7048
7049 end
7050
7051 -- TODO -- repeated here in case EN/ET is the last node. Find a
7052 -- better way of doing things:
7053 if first_et then      -- dir may be nil here !
7054     if has_en then
7055         if last == 'l' then
7056             temp = 'l'    -- W7
7057         else
7058             temp = 'en'   -- W5
7059         end
7060     else
7061         temp = 'on'      -- W6
7062     end
7063     for e = first_et, #nodes do
7064         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7065     end
7066 end
7067
7068 -- dummy node, to close things
7069 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7070
7071 ----- NEUTRAL -----
7072
7073 outer = save_outer
7074 last = outer
7075
7076 local first_on = nil
7077
7078 for q = 1, #nodes do
7079     local item
7080
7081     local outer_first = nodes[q][3]
7082     outer = outer_first or outer
7083     last = outer_first or last
7084
7085     local d = nodes[q][2]
7086     if d == 'an' or d == 'en' then d = 'r' end
7087     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7088
7089     if d == 'on' then
7090         first_on = first_on or q
7091     elseif first_on then
7092         if last == d then
7093             temp = d
7094         else
7095             temp = outer
7096         end
7097         for r = first_on, q - 1 do

```

```

7098     nodes[r][2] = temp
7099     item = nodes[r][1]    -- MIRRORING
7100     if Babel.mirroring_enabled and item.id == GLYPH
7101         and temp == 'r' and characters[item.char] then
7102         local font_mode = font.fonts[item.font].properties.mode
7103         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7104             item.char = characters[item.char].m or item.char
7105         end
7106     end
7107 end
7108 first_on = nil
7109 end
7110
7111 if d == 'r' or d == 'l' then last = d end
7112 end
7113
7114 ----- IMPLICIT, REORDER -----
7115
7116 outer = save_outer
7117 last = outer
7118
7119 local state = {}
7120 state.has_r = false
7121
7122 for q = 1, #nodes do
7123     local item = nodes[q][1]
7124
7125     outer = nodes[q][3] or outer
7126
7127     local d = nodes[q][2]
7128
7129     if d == 'nsm' then d = last end          -- W1
7130     if d == 'en' then d = 'an' end
7131     local isdir = (d == 'r' or d == 'l')
7132
7133     if outer == 'l' and d == 'an' then
7134         state.san = state.san or item
7135         state.ean = item
7136     elseif state.san then
7137         head, state = insert_numeric(head, state)
7138     end
7139
7140     if outer == 'l' then
7141         if d == 'an' or d == 'r' then      -- im -> implicit
7142             if d == 'r' then state.has_r = true end
7143             state.sim = state.sim or item
7144             state.eim = item
7145         elseif d == 'l' and state.sim and state.has_r then
7146             head, state = insert_implicit(head, state, outer)
7147         elseif d == 'l' then
7148             state.sim, state.eim, state.has_r = nil, nil, false
7149         end
7150     else
7151         if d == 'an' or d == 'l' then
7152             if nodes[q][3] then -- nil except after an explicit dir
7153                 state.sim = item -- so we move sim 'inside' the group
7154             else
7155                 state.sim = state.sim or item
7156             end
7157         end
7158     end
7159 end

```

```

7157         end
7158         state.eim = item
7159     elseif d == 'r' and state.sim then
7160         head, state = insert_implicit(head, state, outer)
7161     elseif d == 'r' then
7162         state.sim, state.eim = nil, nil
7163     end
7164 end
7165
7166 if isdir then
7167     last = d          -- Don't search back - best save now
7168 elseif d == 'on' and state.san then
7169     state.san = state.san or item
7170     state.ean = item
7171 end
7172
7173 end
7174
7175 return node.prev(head) or head
7176 end
7177 </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.

```

7178 <*nil>
7179 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7180 \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.

```

7181 \ifx\l@nil\undefined
7182   \newlanguage\l@nil
7183   \@namedef{bbl@hyphendata@the\l@nil}{}{}% Remove warning
7184   \let\bbl@elt\relax
7185   \edef\bbl@languages{% Add it to the list of languages
7186     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
7187 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
7188 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7189 \let\captionnil\@empty
7190 \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.

```
7191 \ldf@finish{nil}
7192 </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`.

```
7193 <(*bplain | blplain)
7194 \catcode`\{=1 % left brace is begin-group character
7195 \catcode`\}=2 % right brace is end-group character
7196 \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).

```
7197 \openin 0 hyphen.cfg
7198 \ifeof0
7199 \else
7200 \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.

```
7201 \def\input #1 {%
7202 \let\input\input
7203 \a hyphen.cfg
7204 \let\input\input
7205 }
7206 \fi
7207 </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`.

```
7208 <bplain>\a plain.tex
7209 <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.

```
7210 <bplain>\def\fmtname{babel-plain}
7211 <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 $\text{\LaTeX} 2_{\epsilon}$ that are needed for `babel`.

```
7212 <<*Emulate LaTeX>> \equiv
7213 % == Code for plain ==
7214 \def\@empty{}
7215 \def\loadlocalcfg#1{%
7216   \openin0#1.cfg
7217   \ifeof0
7218     \closein0
7219   \else
7220     \closein0
7221     {\immediate\write16{*****}%
7222      \immediate\write16{* Local config file #1.cfg used}%
7223      \immediate\write16{*}%
7224     }
7225   \input #1.cfg\relax
7226   \fi
7227   \@endofldf}
```

16.3 General tools

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

```
7228 \long\def\@firstofone#1{#1}
7229 \long\def\@firstoftwo#1#2{#1}
7230 \long\def\@secondoftwo#1#2{#2}
7231 \def\@nnil{\@nil}
7232 \def\@gobbletwo#1#2{}
7233 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7234 \def\@star@or@long#1{%
7235   \@ifstar
7236   {\let\l@ngrel@x\relax#1}%
7237   {\let\l@ngrel@x\long#1}}
7238 \let\l@ngrel@x\relax
7239 \def\@car#1#2\@nil{#1}
7240 \def\@cdr#1#2\@nil{#2}
7241 \let\@typeset@protect\relax
7242 \let\protected@edef\edef
7243 \long\def\@gobble#1{}
7244 \edef\@backslashchar{\expandafter\@gobble\string\}
7245 \def\strip@prefix#1>{}
7246 \def\g@addto@macro#1#2{%
7247   \toks@\expandafter{#1#2}%
7248   \xdef#1{\the\toks@}}
7249 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
```



```

7250 \def\@nameuse#1{\csname #1\endcsname}
7251 \def\@ifundefined#1{%
7252   \expandafter\ifx\csname#1\endcsname\relax
7253     \expandafter\@firstoftwo
7254   \else
7255     \expandafter\@secondoftwo
7256   \fi}
7257 \def\@expandtwoargs#1#2#3{%
7258   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7259 \def\zap@space#1 #2{%
7260   #1%
7261   \ifx#2\@empty\else\expandafter\zap@space\fi
7262   #2}
7263 \let\bbl@trace\@gobble

```

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

```

7264 \ifx\@preamblecmds\undefined
7265   \def\@preamblecmds{}
7266 \fi
7267 \def\@onlypreamble#1{%
7268   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7269     \@preamblecmds\do#1}}
7270 \@onlypreamble\@onlypreamble

```

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

```

7271 \def\begindocument{%
7272   \@begindocumenthook
7273   \global\let\@begindocumenthook\undefined
7274   \def\do##1{\global\let##1\@undefined}%
7275   \@preamblecmds
7276   \global\let\do\noexpand}
7277 \ifx\@begindocumenthook\undefined
7278   \def\@begindocumenthook{}
7279 \fi
7280 \@onlypreamble\@begindocumenthook
7281 \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`.

```

7282 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7283 \@onlypreamble\AtEndOfPackage
7284 \def\@endofldf{}
7285 \@onlypreamble\@endofldf
7286 \let\bbl@afterlang\@empty
7287 \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.

```

7288 \catcode`\&=\z@
7289 \ifx&\if@files\@undefined
7290   \expandafter\let\csname if@files\endcsname\expandafter\endcsname
7291   \csname iffalse\endcsname
7292 \fi
7293 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7294 \def\newcommand{\@star@or@long\new@command}

```

```

7295 \def\new@command#1{%
7296   \@testopt{\@newcommand#1}0}
7297 \def\@newcommand#1[#2]{%
7298   \@ifnextchar [{\@xargdef#1[#2]}%
7299               {\@argdef#1[#2]}}
7300 \long\def\@argdef#1[#2]#3{%
7301   \@yargdef#1\@ne{#2}{#3}}
7302 \long\def\@xargdef#1[#2][#3]#4{%
7303   \expandafter\def\expandafter#1\expandafter{%
7304     \expandafter\@protected@testopt\expandafter #1%
7305     \csname\string#1\expandafter\endcsname{#3}}%
7306   \expandafter\@yargdef \csname\string#1\endcsname
7307   \tw@{#2}{#4}}
7308 \long\def\@yargdef#1#2#3{%
7309   \@tempcnta#3\relax
7310   \advance \@tempcnta \@ne
7311   \let\@hash@\relax
7312   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7313   \@tempcntb #2%
7314   \@whilenum \@tempcntb < \@tempcnta
7315   \do{%
7316     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
7317     \advance\@tempcntb \@ne}%
7318   \let\@hash@###
7319   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7320 \def\providecommand{\@star@or@long\provide@command}
7321 \def\provide@command#1{%
7322   \begingroup
7323     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7324   \endgroup
7325   \expandafter\@ifundefined\@gtempa
7326     {\def\reserved@a{\new@command#1}}%
7327     {\let\reserved@a\relax
7328     \def\reserved@a{\new@command\reserved@a}}%
7329   \reserved@a}%

7330 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7331 \def\declare@robustcommand#1{%
7332   \edef\reserved@a{\string#1}%
7333   \def\reserved@b{#1}%
7334   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7335   \edef#1{%
7336     \ifx\reserved@a\reserved@b
7337       \noexpand\x@protect
7338       \noexpand#1%
7339     \fi
7340     \noexpand\protect
7341     \expandafter\noexpand\csname
7342       \expandafter\@gobble\string#1 \endcsname
7343   }%
7344   \expandafter\new@command\csname
7345     \expandafter\@gobble\string#1 \endcsname
7346 }
7347 \def\x@protect#1{%
7348   \ifx\protect\@typeset@protect\else
7349     \@x@protect#1%
7350   \fi
7351 }
7352 \catcode`\&=\z@ % Trick to hide conditionals

```

```
7353 \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`.

```
7354 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7355 \catcode`\&=4
7356 \ifx\in@\@undefined
7357 \def\in@#1#2{%
7358   \def\in@@##1#1##2##3\in@{%
7359     \ifx\in@@##2\in@false\else\in@true\fi}%
7360   \in@@#2#1\in@\in@@}
7361 \else
7362   \let\bbl@tempa\@empty
7363 \fi
7364 \bbl@tempa
```

\TeX 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).

```
7365 \def\ifpackagewith#1#2#3#4{#3}
```

The \TeX 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.

```
7366 \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 \TeX 2_ε versions; just enough to make things work in plain \TeX environments.

```
7367 \ifx\@tempcnta\@undefined
7368   \csname newcount\endcsname\@tempcnta\relax
7369 \fi
7370 \ifx\@tempcntb\@undefined
7371   \csname newcount\endcsname\@tempcntb\relax
7372 \fi
```

To prevent wasting two counters in \TeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
7373 \ifx\bye\@undefined
7374   \advance\count10 by -2\relax
7375 \fi
7376 \ifx\@ifnextchar\@undefined
7377   \def\@ifnextchar#1#2#3{%
7378     \let\reserved@d=#1%
7379     \def\reserved@a{#2}\def\reserved@b{#3}%
7380     \futurelet\@let@token\@ifnch}
7381 \def\@ifnch{%
7382   \ifx\@let@token\@sptoken
7383     \let\reserved@c\@xifnch
7384   \else
7385     \ifx\@let@token\reserved@d
7386       \let\reserved@c\reserved@a
7387     \else
7388       \let\reserved@c\reserved@b
7389     \fi
7390   \fi
7391   \reserved@c}
7392 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
```

```

7393 \def\:{\@ifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7394 \fi
7395 \def\@testopt#1#2{%
7396   \@ifnextchar[{\#1}{\#1[\#2]}}
7397 \def\@protected@testopt#1{%
7398   \ifx\protect\@typeset@protect
7399     \expandafter\@testopt
7400   \else
7401     \@x@protect#1%
7402   \fi}
7403 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7404   #2\relax}\fi}
7405 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7406   \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

7407 \def\DeclareTextCommand{%
7408   \@dec@text@cmd\providecommand
7409 }
7410 \def\ProvideTextCommand{%
7411   \@dec@text@cmd\providecommand
7412 }
7413 \def\DeclareTextSymbol#1#2#3{%
7414   \@dec@text@cmd\chardef#1{#2}#3\relax
7415 }
7416 \def\@dec@text@cmd#1#2#3{%
7417   \expandafter\def\expandafter#2%
7418     \expandafter{%
7419       \csname#3-cmd\expandafter\endcsname
7420       \expandafter#2%
7421       \csname#3\string#2\endcsname
7422     }%
7423 % \let\@ifdefinable\@rc@ifdefinable
7424 \expandafter#1\csname#3\string#2\endcsname
7425 }
7426 \def\@current@cmd#1{%
7427   \ifx\protect\@typeset@protect\else
7428     \noexpand#1\expandafter\@gobble
7429   \fi
7430 }
7431 \def\@changed@cmd#1#2{%
7432   \ifx\protect\@typeset@protect
7433     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7434       \expandafter\ifx\csname ?\string#1\endcsname\relax
7435         \expandafter\def\csname ?\string#1\endcsname{%
7436           \@changed@x@err{#1}%
7437         }%
7438       \fi
7439       \global\expandafter\let
7440         \csname\cf@encoding\string#1\expandafter\endcsname
7441         \csname ?\string#1\endcsname
7442     \fi
7443     \csname\cf@encoding\string#1%
7444       \expandafter\endcsname
7445   \else
7446     \noexpand#1%

```

```

7447 \fi
7448 }
7449 \def\@changed@x@err#1{%
7450 \errhelp{Your command will be ignored, type <return> to proceed}%
7451 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7452 \def\DeclareTextCommandDefault#1{%
7453 \DeclareTextCommand#1?%
7454 }
7455 \def\ProvideTextCommandDefault#1{%
7456 \ProvideTextCommand#1?%
7457 }
7458 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7459 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7460 \def\DeclareTextAccent#1#2#3{%
7461 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7462 }
7463 \def\DeclareTextCompositeCommand#1#2#3#4{%
7464 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7465 \edef\reserved@b{\string##1}%
7466 \edef\reserved@c{%
7467 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7468 \ifx\reserved@b\reserved@c
7469 \expandafter\expandafter\expandafter\ifx
7470 \expandafter\@car\reserved@a\relax\relax\@nil
7471 \@text@composite
7472 \else
7473 \edef\reserved@b##1{%
7474 \def\expandafter\noexpand
7475 \csname#2\string#1\endcsname####1{%
7476 \noexpand\@text@composite
7477 \expandafter\noexpand\csname#2\string#1\endcsname
7478 ####1\noexpand\@empty\noexpand\@text@composite
7479 {##1}%
7480 }%
7481 }%
7482 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7483 \fi
7484 \expandafter\def\csname\expandafter\string\csname
7485 #2\endcsname\string#1-\string#3\endcsname{#4}
7486 \else
7487 \errhelp{Your command will be ignored, type <return> to proceed}%
7488 \errmessage{\string\DeclareTextCompositeCommand\space used on
7489 inappropriate command \protect#1}
7490 \fi
7491 }
7492 \def\@text@composite#1#2#3\@text@composite{%
7493 \expandafter\@text@composite@x
7494 \csname\string#1-\string#2\endcsname
7495 }
7496 \def\@text@composite@x#1#2{%
7497 \ifx#1\relax
7498 #2%
7499 \else
7500 #1%
7501 \fi
7502 }
7503 %
7504 \def\@strip@args#1:#2-#3\@strip@args{#2}
7505 \def\DeclareTextComposite#1#2#3#4{%

```

```

7506 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7507 \bgroup
7508 \lccode` \@=#4%
7509 \lowercase{%
7510 \egroup
7511 \reserved@a @%
7512 }%
7513 }
7514 %
7515 \def\UseTextSymbol#1#2{#2}
7516 \def\UseTextAccent#1#2#3{}
7517 \def\@use@text@encoding#1{}
7518 \def\DeclareTextSymbolDefault#1#2{%
7519 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}{#1}}%
7520 }
7521 \def\DeclareTextAccentDefault#1#2{%
7522 \DeclareTextCommandDefault#1{\UseTextAccent{#2}{#1}}%
7523 }
7524 \def\cf@encoding{OT1}

```

Currently we only use the $\LaTeX 2_{\epsilon}$ method for accents for those that are known to be made active in *some* language definition file.

```

7525 \DeclareTextAccent{"}{OT1}{127}
7526 \DeclareTextAccent{'}{OT1}{19}
7527 \DeclareTextAccent{^}{OT1}{94}
7528 \DeclareTextAccent{`}{OT1}{18}
7529 \DeclareTextAccent{~}{OT1}{126}

```

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

```

7530 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7531 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
7532 \DeclareTextSymbol{\textquoteleft}{OT1}{`'}
7533 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
7534 \DeclareTextSymbol{\i}{OT1}{16}
7535 \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`.

```

7536 \ifx\scriptsize\@undefined
7537 \let\scriptsize\sevenrm
7538 \fi
7539 % End of code for plain
7540 <</Emulate LaTeX>>

```

A proxy file:

```

7541 < *plain>
7542 \input babel.def
7543 </plain>

```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.

During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).