

Babel

Version 3.62
2021/07/19

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	8
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	16
1.12	The base option	18
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	33
1.18	Dates	35
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Transforms	38
1.22	Selection based on BCP 47 tags	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	48
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	50
1.30	Tips, workarounds, known issues and notes	50
1.31	Current and future work	51
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	52
2.1	Format	52
3	The interface between the core of babel and the language definition files	53
3.1	Guidelines for contributed languages	54
3.2	Basic macros	55
3.3	Skeleton	56
3.4	Support for active characters	57
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	58
3.7	Macros common to a number of languages	58
3.8	Encoding-dependent strings	58
4	Changes	62
4.1	Changes in babel version 3.9	62

II	Source code	62
5	Identification and loading of required files	63
6	locale directory	63
7	Tools	63
7.1	Multiple languages	68
7.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	68
7.3	base	70
7.4	Conditional loading of shorthands	73
7.5	Cross referencing macros	74
7.6	Marks	77
7.7	Preventing clashes with other packages	78
7.7.1	ifthen	78
7.7.2	varioref	78
7.7.3	hhline	79
7.7.4	hyperref	79
7.7.5	fancyhdr	79
7.8	Encoding and fonts	80
7.9	Basic bidi support	81
7.10	Local Language Configuration	85
7.11	Language options	85
8	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	89
8.1	Tools	89
9	Multiple languages	90
9.1	Selecting the language	92
9.2	Errors	101
9.3	Hooks	104
9.4	Setting up language files	106
9.5	Shorthands	108
9.6	Language attributes	117
9.7	Support for saving macro definitions	119
9.8	Short tags	120
9.9	Hyphens	121
9.10	Multiencoding strings	122
9.11	Macros common to a number of languages	129
9.12	Making glyphs available	129
9.12.1	Quotation marks	129
9.12.2	Letters	131
9.12.3	Shorthands for quotation marks	132
9.12.4	Umlauts and tremas	132
9.13	Layout	134
9.14	Load engine specific macros	134
9.15	Creating and modifying languages	134
10	Adjusting the Babel behavior	156
11	Loading hyphenation patterns	158
12	Font handling with fontspec	162

13	Hooks for XeTeX and LuaTeX	167
13.1	XeTeX	167
13.2	Layout	169
13.3	LuaTeX	170
13.4	Southeast Asian scripts	176
13.5	CJK line breaking	177
13.6	Arabic justification	180
13.7	Common stuff	184
13.8	Automatic fonts and ids switching	184
13.9	Bidi	189
13.10	Layout	191
13.11	Lua: transforms	194
13.12	Lua: Auto bidi with basic and basic-r	202
14	Data for CJK	213
15	The ‘nil’ language	214
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	222

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 pdf \TeX , xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-Plain \TeX . Part II describes the code, and usually it can be ignored.

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

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

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

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

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

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

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

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

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

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

PDF_{TEX}

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

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

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

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

⁶Thanks to Enrico Gregorio

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

1.11 Package options

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

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

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

activegrave Same for `.

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

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

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

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

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

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

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

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

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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

1.12 The base option

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

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

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

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

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

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

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

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

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

1.13 ini files

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

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

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.

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

Or also:

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

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

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

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

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:

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.

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{\lq \u \o \j \k \j} % Random
```

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

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}	bas	Basaa
agg	Aghem	be	Belarusian ^{ul}
ak	Akan	bem	Bemba
am	Amharic ^{ul}	bez	Bena
ar	Arabic ^{ul}	bg	Bulgarian ^{ul}
ar-DZ	Arabic ^{ul}	bm	Bambara
ar-MA	Arabic ^{ul}	bn	Bangla ^{ul}
ar-SY	Arabic ^{ul}	bo	Tibetan ^u
as	Assamese	brx	Bodo
asa	Asu	bs-Cyrl	Bosnian
ast	Asturian ^{ul}	bs-Latn	Bosnian ^{ul}
az-Cyrl	Azerbaijani	bs	Bosnian ^{ul}
az-Latn	Azerbaijani	ca	Catalan ^{ul}
az	Azerbaijani ^{ul}	ce	Chechen

cgg	Chiga	gv	Manx
chr	Cherokee	ha-GH	Hausa
ckb	Central Kurdish	ha-NE	Hausa ¹
cop	Coptic	ha	Hausa
cs	Czech ^{ul}	haw	Hawaiian
cu	Church Slavic	he	Hebrew ^{ul}
cu-Cyrs	Church Slavic	hi	Hindi ^u
cu-Glag	Church Slavic	hr	Croatian ^{ul}
cy	Welsh ^{ul}	hsb	Upper Sorbian ^{ul}
da	Danish ^{ul}	hu	Hungarian ^{ul}
dav	Taita	hy	Armenian ^u
de-AT	German ^{ul}	ia	Interlingua ^{ul}
de-CH	German ^{ul}	id	Indonesian ^{ul}
de	German ^{ul}	ig	Igbo
dje	Zarma	ii	Sichuan Yi
dsb	Lower Sorbian ^{ul}	is	Icelandic ^{ul}
dua	Duala	it	Italian ^{ul}
dyo	Jola-Fonyi	ja	Japanese
dz	Dzongkha	jgo	Ngomba
ebu	Embu	jmc	Machame
ee	Ewe	ka	Georgian ^{ul}
el	Greek ^{ul}	kab	Kabyle
el-polyton	Polytonic Greek ^{ul}	kam	Kamba
en-AU	English ^{ul}	kde	Makonde
en-CA	English ^{ul}	kea	Kabuverdianu
en-GB	English ^{ul}	khq	Koyra Chiini
en-NZ	English ^{ul}	ki	Kikuyu
en-US	English ^{ul}	kk	Kazakh
en	English ^{ul}	kkj	Kako
eo	Esperanto ^{ul}	kl	Kalaallisut
es-MX	Spanish ^{ul}	kln	Kalenjin
es	Spanish ^{ul}	km	Khmer
et	Estonian ^{ul}	kn	Kannada ^{ul}
eu	Basque ^{ul}	ko	Korean
ewo	Ewondo	kok	Konkani
fa	Persian ^{ul}	ks	Kashmiri
ff	Fulah	ksb	Shambala
fi	Finnish ^{ul}	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French ^{ul}	ky	Kyrgyz
fr-BE	French ^{ul}	lag	Langi
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

mer	Meru	saq	Samburu
mfe	Morisyen	sbp	Sangu
mg	Malagasy	se	Northern Sami ^{ul}
mgh	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian ^{ul}	sg	Sango
ml	Malayalam ^{ul}	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi ^{ul}	shi	Tachelhit
ms-BN	Malay ^l	si	Sinhala
ms-SG	Malay ^l	sk	Slovak ^{ul}
ms	Malay ^{ul}	sl	Slovenian ^{ul}
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian ^{ul}
naq	Nama	sr-Cyrl-BA	Serbian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Cyrl-ME	Serbian ^{ul}
nd	North Ndebele	sr-Cyrl-XK	Serbian ^{ul}
ne	Nepali	sr-Cyrl	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Latn-BA	Serbian ^{ul}
nmg	Kwasio	sr-Latn-ME	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sr-Latn-XK	Serbian ^{ul}
nnh	Ngiemboon	sr-Latn	Serbian ^{ul}
nus	Nuer	sr	Serbian ^{ul}
nyn	Nyankole	sv	Swedish ^{ul}
om	Oromo	sw	Swahili
or	Odia	ta	Tamil ^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

yue	Cantonese	zh-Hans	Chinese
zgh	Standard Moroccan Tamazight	zh-Hant-HK	Chinese
		zh-Hant-MO	Chinese
zh-Hans-HK	Chinese	zh-Hant	Chinese
zh-Hans-MO	Chinese	zh	Chinese
zh-Hans-SG	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	brazilian
akan	breton
albanian	british
american	bulgarian
amharic	burmese
ancientgreek	canadian
arabic	cantonese
arabic-algeria	catalan
arabic-DZ	centralatlastamazight
arabic-morocco	centralkurdish
arabic-MA	chechen
arabic-syria	cherokee
arabic-SY	chiga
armenian	chinese-hans-hk
assamese	chinese-hans-mo
asturian	chinese-hans-sg
asu	chinese-hans
australian	chinese-hant-hk
austrian	chinese-hant-mo
azerbaijani-cyrillic	chinese-hant
azerbaijani-cyrl	chinese-simplified-hongkongsarchina
azerbaijani-latin	chinese-simplified-macausarchina
azerbaijani-latn	chinese-simplified-singapore
azerbaijani	chinese-simplified
bafia	chinese-traditional-hongkongsarchina
bambara	chinese-traditional-macausarchina
basaa	chinese-traditional
basque	chinese
belarusian	churchslavic
bemba	churchslavic-cyrs
bena	churchslavic-oldcyrillic ¹²
bengali	churchslavic-glag
bodo	churchslavic-glagolitic
bosnian-cyrillic	cognian
bosnian-cyrl	cornish
bosnian-latin	croatian
bosnian-latn	czech
bosnian	danish

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

duala
dutch
dzongkha
embu
english-au
english-australia
english-ca
english-canada
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
northernnsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali

sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq

telugu	uzbek-latin
teso	uzbek-latn
thai	uzbek
tibetan	vai-latin
tigrinya	vai-latn
tongan	vai-vai
turkish	vai-vaii
turkmen	vai
ukenglish	vietnam
ukrainian	vietnamese
uppertsorbian	vunjo
urdu	walser
usenglish	welsh
usorbian	westernfrisian
uyghur	yangben
uzbek-arab	yiddish
uzbek-arabic	yoruba
uzbek-cyrillic	zarma
uzbek-cyrl	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.

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

NOTE You may load fontspec explicitly. For example:

LUATEX/XETEX

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

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

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

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

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

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

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

This is *not* 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` $\{\langle language-name \rangle\}\{\langle caption-name \rangle\}\{\langle string \rangle\}$

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

Macros to be run when a language is selected can be add to \extras<lang>:

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

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

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

`import=` *⟨language-tag⟩*

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

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

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

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

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

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

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

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

captions= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

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

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

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

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

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

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

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

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


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

Remember there is an alternative syntax for the latter:

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

script= $\langle script-name \rangle$

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

language= $\langle language-name \rangle$

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

alph= $\langle counter-name \rangle$

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

Alph= $\langle counter-name \rangle$

Same for `\Alph`.

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

onchar= ids | fonts

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

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

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

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

intrapenalty= `<penalty>`

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

justification= `kashida | elongated | unhyphenated`

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

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

mapfont= `direction`

Assigns the font for the writing direction of this language (only with `bidi=basic`). Whenever possible, instead of this option use `onchar`, based on the script, which usually makes more sense. More precisely, what `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

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

1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

Telugu	Uyghur	Uzbek	Cantonese
Thai	Urdu	Vai	Chinese

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

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

New 4.41 Many ‘ini’ locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with `xetex` and `luatex` and are fully expendable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000. There are several ways to use them (for the available styles in each language, see the list below):

- `\localnumeral{<style>}{<number>}`, like `\localnumeral{abjad}{15}`
- `\localecounter{<style>}{<counter>}`, like `\localecounter{lower}{section}`
- In `\babelprovide`, as an argument to the keys `alph` and `Alph`, which redefine what `\alph` and `\Alph` print. For example:

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

The styles are:

Ancient Greek lower.ancient, upper.ancient
Amharic afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebena, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yemsa
Arabic abjad, maghrebi.abjad
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. *Çileyâ Pêşîn 2019*, but with variant=iza fa it prints 31'ê *Çileyâ Pêşînê 2019*.

1.19 Accessing language info

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

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

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

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

\localeinfo {*<field>*}

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

name.english as provided by the Unicode CLDR.

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

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

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

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

script.name, as provided by the Unicode CLDR.

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

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

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

`\getlocaleproperty` * $\{\langle macro \rangle\}\{\langle locale \rangle\}\{\langle property \rangle\}$

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

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

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

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

Babel remembers which ini files have been loaded. There is a loop named `\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that `\LocaleForEach{\message{ **#1** }}` just shows the loaded ini's.

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

`\babelhyphen` * $\{\langle type \rangle\}$

`\babelhyphen` * $\{\langle text \rangle\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

`\babelpatterns` [`\language`],`\language`},{`\patterns`}

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

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

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

New 3.31 (Only *luatex*.) With `\babelprovide` and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (**New 3.32** it is disabled in verbatim mode, or more precisely when the `hyphenrules` are set to `nohyphenation`). It can be activated alternatively by setting explicitly the `intraspace`.

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

1.21 Transforms

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

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

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

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

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

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for <code>dad</code> (simple and \TeX -friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .

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

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

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

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

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 ([\acute{u}]), the replacement could be {1| \acute{u} | \acute{u} }, which maps \acute{t} to \acute{u} , and \acute{u} to \acute{u} , 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{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}

```

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

`autoload.bcp47` with values on and off.

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

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

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

```

\babeladjust{ bcp47.toname = on }

```

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

1.23 Selecting scripts

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

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.

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

`\ensureascii` $\langle text \rangle$

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

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

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

1.24 Selecting directions

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

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

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

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

```
\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.

¹⁷But still defined for backwards compatibility.

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 الهليني (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` and `\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).

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr `{\lr-text}`

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

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

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

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

\BabelPatchSection `{\section-name}`

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

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

New 3.17 Something like:

```
\BabelFootnote{\parsfootnote}{\language}\{ }\{ }
```

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

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

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

```
\BabelFootnote{\footnote}{\language}\{ }\{ }%
\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.

New 3.62 This is not the only way to inject code at those points. The events listed below can be used as a hook name in `\AddToHook` in the form `babel/{<name>}`, but there is a limitation, because the parameters passed with the `babel` mechanism are not allowed. The `\AddToHook` mechanism does *not* replace the current one in ‘`babel`’. Its main advantage is you can reconfigure ‘`babel`’ even before loading it. See the example below.

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

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

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

1.27 Languages supported by babel with ldf files

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

Afrikaans	afrikaans
Azerbaijani	azerbaijani
Basque	basque
Breton	breton
Bulgarian	bulgarian
Catalan	catalan
Croatian	croatian
Czech	czech
Danish	danish
Dutch	dutch
English	english, USenglish, american, UKenglish, british, canadian, australian, newzealand
Esperanto	esperanto
Estonian	estonian
Finnish	finnish
French	french, francais, canadien, acadian
Galician	galician
German	austrian, german, germanb, ngerman, naustrian
Greek	greek, polutonikogreek
Hebrew	hebrew
Icelandic	icelandic
Indonesian	indonesian (bahasa, indon, bahasai)
Interlingua	interlingua
Irish Gaelic	irish
Italian	italian
Latin	latin
Lower Sorbian	lowersorbian
Malay	malay, melayu (bahasam)
North Sami	samin
Norwegian	norsk, nynorsk

Polish polish
Portuguese portuguese, brazilian (portuges, brazil)¹⁹
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

```

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

```

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

1.28 Unicode character properties in luatex

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

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

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

```

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

```

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

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

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

1.29 Tweaking some features

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

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

1.30 Tips, workarounds, known issues and notes

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

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

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

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is a similar issue with floats, too. There is no known workaround.

²⁰This explains why \TeX 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.

- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the ‘to do’ list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the ‘to do’ list, although there is a partial solution.)

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

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

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 \TeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study. Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ból”, in Spanish an item labelled “3.^o” may be referred to as either “ítem 3.^o” or “3.^{er} ítem”, and so on. An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to `\specials` remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (xe-bidi).

1.32 Tentative and experimental code

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

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

defines captions, date, numerals, etc., but ignores the code in the tex file (for example, extended numerals in Greek).

Labels

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

2 Loading languages with language.dat

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

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

2.1 Format

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

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

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

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

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

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

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

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 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.

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

²⁶But not removed, for backward compatibility.

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<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, \LaTeX 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 \LaTeX 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`
`\bbl@deactivate`

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

The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \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, `<cname>`, the control sequence for which the meaning has to be saved.

`\babel@savevariable`

A second macro is provided to save the current value of a variable. In this context,

²⁷This mechanism was introduced by Bernd Raichle.

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

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

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

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

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 a 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:

```
\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}
```

²⁸In future releases further categories may be added.

```

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

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

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

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

\EndBabelCommands

```

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

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

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

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

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

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

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

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

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

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

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

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

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 $\langle map-list \rangle$ 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 \LaTeX , 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 $\{\langle to-lower-macros \rangle\}$

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 though the given uppercase codes, using the step, and assigns them the `lccode`, which is also increased (MM stands for *many-to-many*).

- `\BabelLowerMO{⟨ucode-from⟩}{⟨ucode-to⟩}{⟨step⟩}{⟨lcode⟩}` loops through the given uppercase codes, using the step, and assigns them the lcode, which is fixed (MO stands for *many-to-one*).

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.62>>
2 <<date=2021/07/19>>
```

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}

```

\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

```

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

```

40 \expandafter\bb1@trim@b
41 \else
42 \expandafter\bb1@trim@b\expandafter#1%
43 \fi}%
44 \long\def\bb1@trim@b#1##1 \nil{\bb1@trim@i##1}}
45 \bb1@tempa{ }
46 \long\def\bb1@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bb1@trim@def#1{\bb1@trim{\def#1}}

```

\bb1@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\bb1@ifunset#1{%
50 \expandafter\ifx\csname#1\endcsname\relax
51 \expandafter\@firstoftwo
52 \else
53 \expandafter\@secondoftwo
54 \fi}
55 \bb1@ifunset{ifcsname}%
56 {}%
57 {\gdef\bb1@ifunset#1{%
58 \ifcsname#1\endcsname
59 \expandafter\ifx\csname#1\endcsname\relax
60 \bb1@afterelse\expandafter\@firstoftwo
61 \else
62 \bb1@afterfi\expandafter\@secondoftwo
63 \fi
64 \else
65 \expandafter\@firstoftwo
66 \fi}}
67 \endgroup

```

\bb1@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\bb1@ifblank#1{%
69 \bb1@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bb1@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bb1@ifset#1#2#3{%
72 \bb1@ifunset{#1}{#3}{\bb1@exp{\bb1@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\bb1@forkv#1#2{%
74 \def\bb1@kvcmd##1##2##3{#2}%
75 \bb1@kvnext#1,\@nil,}
76 \def\bb1@kvnext#1,{%
77 \ifx\@nil#1\relax\else
78 \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
79 \expandafter\bb1@kvnext
80 \fi}
81 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
82 \bb1@trim@def\bb1@forkv@a{#1}%
83 \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@forkv@a}}{#2}{#4}}

```

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

```

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

```

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     \edef\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_T_EX, 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@bsphack{%
155   \ifhmode
156     \hskip\z@skip
157     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158   \else
159     \let\bbl@esphack\@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}

```

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

```

173 \ifx\IfFormatAtLeastTF\@undefined
174   \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
175 \else
176   \let\bbl@ifformatlater\IfFormatAtLeastTF
177 \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.

```

178 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after

```

```

179 \toks@ \expandafter \expandafter \expandafter {%
180 \csname extras \language name \endcsname}%
181 \bbl@exp{\in@{#1}{\the\toks@}}%
182 \ifin@ \else
183 \@temptokena{#2}%
184 \edef\bbl@tempc{\the\@temptokena\the\toks@}%
185 \toks@ \expandafter{\bbl@tempc#3}%
186 \expandafter \edef \csname extras \language name \endcsname{\the\toks@}%
187 \fi}
188 <</Basic macros>>

```

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

```

189 <<*Make sure ProvidesFile is defined>> ≡
190 \ifx\ProvidesFile\@undefined
191 \def\ProvidesFile#1[#2 #3 #4]{%
192 \wlog{File: #1 #4 #3 <#2>}%
193 \let\ProvidesFile\@undefined}
194 \fi
195 <</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.

```

196 <<*Define core switching macros>> ≡
197 \ifx\language\@undefined
198 \csname newcount \endcsname \language
199 \fi
200 <</Define core switching macros>>

```

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

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

```

201 <<*Define core switching macros>> ≡
202 \countdef\last@language=19
203 \def\addlanguage{\csname newlanguage \endcsname}
204 <</Define core switching macros>>

```

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

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

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

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.

```

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

```

```

262   Some functions for '#1' are tentative.\\%
263   They might not work as expected and their behavior\\%
264   may change in the future.\\%
265   Reported}}
266 \def\nolanerr#1{%
267   \bbl@error
268   {You haven't defined the language '#1' yet.\\%
269     Perhaps you misspelled it or your installation\\%
270     is not complete}%
271   {Your command will be ignored, type <return> to proceed}}
272 \def\nopatterns#1{%
273   \bbl@warning
274   {No hyphenation patterns were preloaded for\\%
275     the language '#1' into the format.\\%
276     Please, configure your TeX system to add them and\\%
277     rebuild the format. Now I will use the patterns\\%
278     preloaded for \bbl@nulllanguage\space instead}}
279   % End of errors
280 \@ifpackagewith{babel}{silent}
281   {\let\bbl@info@gobble
282    \let\bbl@infowarn@gobble
283    \let\bbl@warning@gobble}
284   {}
285 %
286 \def\AfterBabelLanguage#1{%
287   \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.

```

288 \ifx\bbl@languages\undefined\else
289   \begingroup
290     \catcode\^^I=12
291     \@ifpackagewith{babel}{showlanguages}{%
292       \begingroup
293         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
294         \wlog{<*languages>}%
295         \bbl@languages
296         \wlog{</languages>}%
297       \endgroup}{%
298     \endgroup
299     \def\bbl@elt#1#2#3#4{%
300       \ifnum#2=\z@
301         \gdef\bbl@nulllanguage{#1}%
302         \def\bbl@elt##1##2##3##4{}}%
303     \fi}%
304   \bbl@languages
305 \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`.

```

306 \bbl@trace{Defining option 'base'}
307 \@ifpackagewith{babel}{base}{%
308   \let\bbl@onlyswitch\@empty

```

```

309 \let\bbl@provide@locale\relax
310 \input babel.def
311 \let\bbl@onlyswitch\@undefined
312 \ifx\directlua\@undefined
313   \DeclareOption*{\bbl@patterns{\CurrentOption}}%
314   \else
315     \input luababel.def
316     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
317   \fi
318 \DeclareOption{base}{}%
319 \DeclareOption{showlanguages}{}%
320 \ProcessOptions
321 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
322 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
323 \global\let\@ifl@ter@@\@ifl@ter
324 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
325 \endinput}{}%
326% \end{macrocode}
327%
328% \subsection{\texttt{key=value} options and other general option}
329%
330%   The following macros extract language modifiers, and only real
331%   package options are kept in the option list. Modifiers are saved
332%   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
333%   no modifiers have been given, the former is |\relax|. How
334%   modifiers are handled are left to language styles; they can use
335%   |\in@|, loop them with |\@for| or load |keyval|, for example.
336%
337%   \begin{macrocode}
338 \bbl@trace{key=value and another general options}
339 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
340 \def\bbl@tempb#1.#2{% Remove trailing dot
341   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
342 \def\bbl@tempd#1.#2@nnil{% TODO. Refactor lists?
343   \ifx\@empty#2%
344     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
345   \else
346     \in@{,provide=}{{,#1}}%
347     \ifin@
348       \edef\bbl@tempc{%
349         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
350   \else
351     \in@{=}{#1}%
352     \ifin@
353       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
354   \else
355     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
356     \bbl@csarg\edef{mod#1}{\bbl@tempb#2}%
357   \fi
358   \fi
359 \fi}
360 \let\bbl@tempc\@empty
361 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
362 \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.

```

363 \DeclareOption{KeepShorthandsActive}{}

```



```

364 \DeclareOption{activeacute}{}
365 \DeclareOption{activegrave}{}
366 \DeclareOption{debug}{}
367 \DeclareOption{noconfigs}{}
368 \DeclareOption{showlanguages}{}
369 \DeclareOption{silent}{}
370 % \DeclareOption{mono}{}
371 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
372 \chardef\bbl@iniflag\z@
373 \DeclareOption{provide=*}{\chardef\bbl@iniflag@ne} % main -> +1
374 \DeclareOption{provide+=*}{\chardef\bbl@iniflag@tw@} % add = 2
375 \DeclareOption{provide*=*}{\chardef\bbl@iniflag@thr@@} % add + main
376 % A separate option
377 \let\bbl@autoload@options\@empty
378 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
379 % Don't use. Experimental. TODO.
380 \newif\ifbbl@single
381 \DeclareOption{selectors=off}{\bbl@singletrue}
382 <<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.

```

383 \let\bbl@opt@shorthands\@nnil
384 \let\bbl@opt@config\@nnil
385 \let\bbl@opt@main\@nnil
386 \let\bbl@opt@headfoot\@nnil
387 \let\bbl@opt@layout\@nnil
388 \let\bbl@opt@provide\@nnil

```

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

```

389 \def\bbl@tempa#1=#2\bbl@tempa{%
390   \bbl@csarg\ifx{opt@#1}\@nnil
391     \bbl@csarg\edef{opt@#1}{#2}%
392   \else
393     \bbl@error
394     {Bad option '#1=#2'. Either you have misspelled the\\%
395      key or there is a previous setting of '#1'. Valid\\%
396      keys are, among others, 'shorthands', 'main', 'bidi',\\%
397      'strings', 'config', 'headfoot', 'safe', 'math'.}%
398     {See the manual for further details.}
399   \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.

```

400 \let\bbl@language@opts\@empty
401 \DeclareOption*{%
402   \bbl@xin@{\string=}{\CurrentOption}%
403   \ifin@
404     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
405   \else
406     \bbl@add@list\bbl@language@opts{\CurrentOption}%
407   \fi}

```

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

```

408 \ProcessOptions*

```

```

409 \ifx\bbl@opt@provide\@nnil\else % Tests. Ignore.
410 \chardef\bbl@iniflag\@ne
411 \fi
412 %

```

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 `\bbl@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

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

```

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

```

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

```

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

```

434 \edef\bbl@opt@shorthands{%
435   \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

436 \bbl@ifshorthand{'}%
437   {\PassOptionsToPackage{activeacute}{babel}}{}
438 \bbl@ifshorthand{`}%
439   {\PassOptionsToPackage{activegrave}{babel}}{}
440 \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.

```

441 \ifx\bbl@opt@headfoot\@nnil\else
442   \g@addto@macro\@resetactivechars{%
443     \set@typeset@protect
444     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
445     \let\protect\noexpand}
446 \fi

```

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

```

447 \ifx\bbbl@opt@safe\undefined
448   \def\bbbl@opt@safe{BR}
449 \fi
450 \ifx\bbbl@opt@main\@nnil\else
451   \edef\bbbl@language@opts{%
452     \ifx\bbbl@language@opts\@empty\else\bbbl@language@opts,\fi
453     \bbbl@opt@main}
454 \fi

```

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

```

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

```

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

```
467 \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.

```

468 <<*More package options>> ≡
469 \DeclareOption{safe=none}{\let\bbbl@opt@safe\@empty}
470 \DeclareOption{safe=bib}{\def\bbbl@opt@safe{B}}
471 \DeclareOption{safe=ref}{\def\bbbl@opt@safe{R}}
472 <</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.

```

473 \bbbl@trace{Cross referencing macros}
474 \ifx\bbbl@opt@safe\@empty\else
475   \def\@newl@bel#1#2#3{%
476     {\@safe@activestrue
477       \bbbl@ifunset{#1@#2}%
478       \relax
479       {\gdef\@multiplelabels{%
480         \@latex@warning@no@line{There were multiply-defined labels}}%
481         \@latex@warning@no@line{Label `#2' multiply defined}}%
482       \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.

```

483 \CheckCommand*\@testdef[3]{%
484   \def\reserved@a{#3}%
485   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
486   \else
487     \@tempswatrue
488   \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.

```

489 \def\@testdef#1#2#3{% TODO. With @samestring?
490   \@safe@activetrue
491   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
492   \def\bbl@tempb{#3}%
493   \@safe@activetrue
494   \ifx\bbl@tempa\relax
495   \else
496     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
497   \fi
498   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
499   \ifx\bbl@tempa\bbl@tempb
500   \else
501     \@tempswatrue
502   \fi}
503 \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.

```

504 \bbl@xin@{R}\bbl@opt@safe
505 \ifin@
506   \bbl@redefineroobust\ref#1{%
507     \@safe@activetrue\org@ref{#1}\@safe@activetrue}
508   \bbl@redefineroobust\pageref#1{%
509     \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
510 \else
511   \let\org@ref\ref
512   \let\org@pageref\pageref
513 \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.

```

514 \bbl@xin@{B}\bbl@opt@safe
515 \ifin@
516   \bbl@redefine\@citex[#1]#2{%
517     \@safe@activetrue\edef\@tempa{#2}\@safe@activetrue}
518   \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.

```

519 \AtBeginDocument{%
520   \@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.)

```
521 \def\@citex[#1][#2]#3{%
522   \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
523   \org@@citex[#1][#2]{\@tempa}}%
524 }{}}
```

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

```
525 \AtBeginDocument{%
526   \@ifpackageloaded{cite}{%
527     \def\@citex[#1]#2{%
528       \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
529     }{}}
```

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

```
530 \bbl@redefine\nocite#1{%
531   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
532 \bbl@redefine\bibcite{%
533   \bbl@cite@choice
534   \bibcite}
```

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

```
535 \def\bbl@bibcite#1#2{%
536   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
537 \def\bbl@cite@choice{%
538   \global\let\bibcite\bbl@bibcite
539   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
540   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
541   \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.

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

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

```
543 \bbl@redefine\@bibitem#1{%
544   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
545 \else
546   \let\org@nocite\nocite
547   \let\org@@citex\@citex
548   \let\org@bibcite\bibcite
549   \let\org@@bibitem\@bibitem
550 \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.

```

551 \bbl@trace{Marks}
552 \IfBabelLayout{sectioning}
553   {\ifx\bbl@opt@headfoot\@nnil
554     \g@addto@macro\@resetactivechars{%
555       \set@typeset@protect
556       \expandafter\select@language@x\expandafter{\bbl@main@language}%
557       \let\protect\noexpand
558       \ifcase\bbl@bidimode\else % Only with bidi. See also above
559         \edef\thepage{%
560           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
561       \fi}%
562   \fi}
563 {\ifbbl@single\else
564   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
565   \markright#1{%
566     \bbl@ifblank{#1}%
567     {\org@markright{}}%
568     {\toks@{#1}%
569       \bbl@exp{%
570         \\org@markright{\protect\\foreignlanguage{\language}%
571           {\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`

```

572   \ifx\@mkboth\markboth
573     \def\bbl@tempc{\let\@mkboth\markboth}
574   \else
575     \def\bbl@tempc{}
576   \fi
577   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
578   \markboth#1#2{%
579     \protected@edef\bbl@tempb##1{%
580       \protect\foreignlanguage
581       {\language}{\protect\bbl@restore@actives##1}%
582     \bbl@ifblank{#1}%
583     {\toks@{}}%
584     {\toks@\expandafter{\bbl@tempb{#1}}}%
585     \bbl@ifblank{#2}%
586     {\@temptokena{}}%
587     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
588     \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
589     \bbl@tempc
590   \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.

```
591 \bbl@trace{Preventing clashes with other packages}
592 \bbl@xin@{R}\bbl@opt@safe
593 \ifin@
594 \AtBeginDocument{%
595   \@ifpackageloaded{ifthen}{%
596     \bbl@redefine@long\ifthenelse#1#2#3{%
597       \let\bbl@temp@pref\pageref
598       \let\pageref\org@pageref
599       \let\bbl@temp@ref\ref
600       \let\ref\org@ref
601       \@safe@activestrue
602       \org@ifthenelse{#1}%
603         {\let\pageref\bbl@temp@pref
604          \let\ref\bbl@temp@ref
605          \@safe@activesfalse
606          #2}%
607         {\let\pageref\bbl@temp@pref
608          \let\ref\bbl@temp@ref
609          \@safe@activesfalse
610          #3}%
611     }%
612   }{}%
613 }
```

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`.

```
614 \AtBeginDocument{%
615   \@ifpackageloaded{varioref}{%
616     \bbl@redefine\@@vpageref#1[#2]#3{%
617       \@safe@activestrue
618       \org@@@vpageref{#1}[#2]{#3}%
619       \@safe@activesfalse}%
620     \bbl@redefine\vrefpagemum#1#2{%
621       \@safe@activestrue
622       \org@vrefpagemum{#1}{#2}%
623       \@safe@activesfalse}%
624   }
```

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.

```
624 \expandafter\def\csname Ref \endcsname#1{%
625 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
626 }{}%
627 }
628 \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.

```
629 \AtEndOfPackage{%
630 \AtBeginDocument{%
631 \ifpackageloaded{hhline}%
632 {\expandafter\ifx\csname normal@char\string\endcsname\relax
633 \else
634 \makeatletter
635 \def\@currname{hhline}\input{hhline.sty}\makeatother
636 \fi}%
637 {}}}
```

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.

```
638 % \AtBeginDocument{%
639 % \ifx\pdfstringdefDisableCommands\@undefined\else
640 % \pdfstringdefDisableCommands{\languageshorthands{system}}%
641 % \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.

```
642 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
643 \lowercase{\foreignlanguage{#1}}}
```

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

```
644 \def\substitutefontfamily#1#2#3{%
645 \lowercase{\immediate\openout15=#1#2.fd\relax}%
646 \immediate\write15{%
647 \string\ProvidesFile{#1#2.fd}%
648 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
649 \space generated font description file]^^J
```



```

650 \string\DeclareFontFamily{#1}{#2}{}}^^J
651 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^^J
652 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^^J
653 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^^J
654 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^^J
655 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^^J
656 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^^J
657 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^^J
658 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^^J
659 }%
660 \closeout15
661 }
662 \@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. Requested encodings are currently stored in `\@fontenc@load@list`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

```

\ensureascii

663 \bbl@trace{Encoding and fonts}
664 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
665 \newcommand\BabelNonText{TS1,T3,TS3}
666 \let\org@TeX\TeX
667 \let\org@LaTeX\LaTeX
668 \let\ensureascii@firstofone
669 \AtBeginDocument{%
670   \def\@elt#1{, #1,}%
671   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
672   \let\@elt\relax
673   \let\bbl@tempb\@empty
674   \def\bbl@tempc{OT1}%
675   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
676     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
677   \bbl@foreach\bbl@tempa{%
678     \bbl@xin@{#1}{\BabelNonASCII}%
679     \ifin@
680       \def\bbl@tempb{#1}% Store last non-ascii
681     \else\bbl@xin@{#1}{\BabelNonText}% Pass
682       \ifin@
683         \def\bbl@tempc{#1}% Store last ascii
684       \fi
685     \fi}%
686   \ifx\bbl@tempb\@empty\else
687     \bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
688     \ifin@
689       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
690     \fi
691     \edef\ensureascii#1{%
692       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
693     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
694     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
695   \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.

```
696 \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.

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

```
717 \DeclareRobustCommand{\latintext}{%
718   \fontencoding{\latinencoding}\selectfont
719   \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.

```
720 \ifx\@undefined\DeclareTextFontCommand
721   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
722 \else
723   \DeclareTextFontCommand{\textlatin}{\latintext}
724 \fi
```

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

```
725 \bbl@ifformatlater{2021-06-01}%
726   {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
727   {\def\bbl@patchfont#1{%
728     \expandafter\bbl@add\csname selectfont \endcsname{#1}%
729     \expandafter\bbl@tglobal\csname selectfont \endcsname}}
```

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.

```

730 \bbl@trace{Loading basic (internal) bidi support}
731 \ifodd\bbl@engine
732 \else % TODO. Move to txtbabel
733   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
734     \bbl@error
735     {The bidi method 'basic' is available only in\\%
736       luatex. I'll continue with 'bidi=default', so\\%
737       expect wrong results}%
738     {See the manual for further details.}%
739     \let\bbl@beforeforeign\leavevmode
740     \AtEndOfPackage{%
741       \EnableBabelHook{babel-bidi}%
742       \bbl@xebidipar}
743   \fi\fi
744   \def\bbl@loadxebidi#1{%
745     \ifx\RTLfootnotetext\@undefined
746       \AtEndOfPackage{%
747         \EnableBabelHook{babel-bidi}%
748         \ifx\fontspec\@undefined
749           \bbl@loadfontspec % bidi needs fontspec
750         \fi
751         \usepackage#1{bidi}}%
752     \fi}
753   \ifnum\bbl@bidimode>200
754     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
755       \bbl@tentative{bidi=bidi}
756       \bbl@loadxebidi{}
757     \or
758       \bbl@loadxebidi{[rldocument]}
759     \or
760       \bbl@loadxebidi{}
761     \fi
762   \fi
763 \fi
764 % TODO? Separate:
765 \ifnum\bbl@bidimode=\@ne
766   \let\bbl@beforeforeign\leavevmode
767   \ifodd\bbl@engine
768     \newattribute\bbl@attr@dir
769     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
```

```

770 \bbl@exp{\output{\bodydir\pagedir\the\output}}
771 \fi
772 \AtEndOfPackage{%
773 \EnableBabelHook{babel-bidi}%
774 \ifodd\bbl@engine\else
775 \bbl@xebidipar
776 \fi}
777 \fi

```

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

```

778 \bbl@trace{Macros to switch the text direction}
779 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
780 \def\bbl@rscripts{% TODO. Base on codes ??
781 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
782 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
783 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
784 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
785 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
786 Old South Arabian,}%
787 \def\bbl@provide@dirs#1{%
788 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
789 \ifin@
790 \global\bbl@csarg\chardef{wdir@#1}\@ne
791 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
792 \ifin@
793 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
794 \fi
795 \else
796 \global\bbl@csarg\chardef{wdir@#1}\z@
797 \fi
798 \ifodd\bbl@engine
799 \bbl@csarg\ifcase{wdir@#1}%
800 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
801 \or
802 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
803 \or
804 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
805 \fi
806 \fi}
807 \def\bbl@switchdir{%
808 \bbl@ifunset{bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
809 \bbl@ifunset{bbl@wdir\languagename}{\bbl@provide@dirs{\languagename}}{}%
810 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
811 \def\bbl@setdirs#1{% TODO - math
812 \ifcase\bbl@select@type % TODO - strictly, not the right test
813 \bbl@bodydir{#1}%
814 \bbl@pdir{#1}%
815 \fi
816 \bbl@texmdir{#1}}
817 % TODO. Only if \bbl@bidimode > 0?:
818 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
819 \DisableBabelHook{babel-bidi}

```

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

```

820 \ifodd\bbl@engine % luatex=1
821 \else % pdftex=0, xetex=2
822 \newcount\bbl@dirlevel
823 \chardef\bbl@thetextdir\z@

```

```

824 \chardef\bbl@thepardir\z@
825 \def\bbl@textdir#1{%
826   \ifcase#1\relax
827     \chardef\bbl@thetextdir\z@
828     \bbl@textdir@i\beginL\endL
829   \else
830     \chardef\bbl@thetextdir\@ne
831     \bbl@textdir@i\beginR\endR
832   \fi}
833 \def\bbl@textdir@i#1#2{%
834   \ifhmode
835     \ifnum\currentgrouplevel>\z@
836       \ifnum\currentgrouplevel=\bbl@dirlevel
837         \bbl@error{Multiple bidi settings inside a group}%
838         {I'll insert a new group, but expect wrong results.}%
839         \bgroup\aftergroup#2\aftergroup\egroup
840       \else
841         \ifcase\currentgrouptype\or % 0 bottom
842           \aftergroup#2% 1 simple {}
843         \or
844           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
845         \or
846           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
847         \or\or\or % vbox vtop align
848         \or
849           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
850         \or\or\or\or\or\or\or % output math disc insert vcent mathchoice
851         \or
852           \aftergroup#2% 14 \begingroup
853         \else
854           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
855         \fi
856       \fi
857       \bbl@dirlevel\currentgrouplevel
858     \fi
859     #1%
860   \fi}
861 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
862 \let\bbl@bodydir\@gobble
863 \let\bbl@pagedir\@gobble
864 \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).

```

865 \def\bbl@xebidipar{%
866   \let\bbl@xebidipar\relax
867   \TeXeTstate\@ne
868   \def\bbl@xeeverypar{%
869     \ifcase\bbl@thepardir
870       \ifcase\bbl@thetextdir\else\beginR\fi
871     \else
872       {\setbox\z@\lastbox\beginR\box\z@}%
873     \fi}%
874   \let\bbl@severypar\everypar
875   \newtoks\everypar
876   \everypar=\bbl@severypar
877   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
878 \ifnum\bbl@bidimode>200

```

```

879 \let\bbl@textdir@i@gobbletwo
880 \let\bbl@xebidipar@empty
881 \AddBabelHook{bidi}{foreign}{%
882   \def\bbl@tempa{\def\BabelText####1}%
883   \ifcase\bbl@thetextdir
884     \expandafter\bbl@tempa\expandafter{\BabelText\LR{##1}}}%
885   \else
886     \expandafter\bbl@tempa\expandafter{\BabelText\RL{##1}}}%
887   \fi}
888 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
889 \fi
890 \fi

```

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

```

891 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}
892 \AtBeginDocument{%
893   \ifx\pdfstringdefDisableCommands\undefined\else
894     \ifx\pdfstringdefDisableCommands\relax\else
895       \pdfstringdefDisableCommands{\let\babelsublr \@firstofone}%
896     \fi
897   \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`.

```

898 \bbl@trace{Local Language Configuration}
899 \ifx\loadlocalcfg\undefined
900   \@ifpackagewith{babel}{noconfigs}%
901   {\let\loadlocalcfg@gobble}%
902   {\def\loadlocalcfg#1{%
903     \InputIfFileExists{#1.cfg}%
904     {\typeout{*****^J%
905               * Local config file #1.cfg used^^J%
906               *}}}%
907   \@empty}}
908 \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).

```

909 \bbl@trace{Language options}
910 \let\bbl@afterlang\relax
911 \let\BabelModifiers\relax
912 \let\bbl@loaded@empty
913 \def\bbl@load@language#1{%
914   \InputIfFileExists{#1.ldf}%
915   {\edef\bbl@loaded{\CurrentOption
916     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
917     \expandafter\let\expandafter\bbl@afterlang
918       \csname\CurrentOption.ldf-h@@k\endcsname
919     \expandafter\let\expandafter\BabelModifiers
920       \csname bbl@mod@\CurrentOption\endcsname}%

```

```

921 {\bbl@error{%
922     Unknown option '\CurrentOption'. Either you misspelled it\\%
923     or the language definition file \CurrentOption.ldf was not found}%}
924     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
925     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
926     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.

```

927 \def\bbl@try@load@lang#1#2#3{%
928     \IfFileExists{\CurrentOption.ldf}%
929     {\bbl@load@language{\CurrentOption}}%
930     {#1\bbl@load@language{#2}#3}}
931 \DeclareOption{hebrew}{%
932     \input{rlbabel.def}%
933     \bbl@load@language{hebrew}}
934 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
935 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
936 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
937 \DeclareOption{polutonikogreek}{%
938     \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
939 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
940 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
941 \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.

```

942 \ifx\bbl@opt@config\@nnil
943     \@ifpackagewith{babel}{noconfigs}{}%
944     {\InputIfFileExists{bblopts.cfg}%
945         {\typeout{*****^^J%
946             * Local config file bblopts.cfg used^^J%
947             *}}%
948         {}}%
949 \else
950     \InputIfFileExists{\bbl@opt@config.cfg}%
951     {\typeout{*****^^J%
952         * Local config file \bbl@opt@config.cfg used^^J%
953         *}}%
954     {\bbl@error{%
955         Local config file '\bbl@opt@config.cfg' not found}%}
956     {Perhaps you misspelled it.}}%
957 \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.

```

958 \let\bbl@tempc\relax
959 \bbl@foreach\bbl@language@opts{%
960     \ifcase\bbl@iniflag % Default
961         \bbl@ifunset{ds@#1}%
962         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
963         {}%
964     \or % provide=
965         \@gobble % case 2 same as 1

```

```

966 \or % provide+=*
967 \bbl@ifunset{ds@#1}%
968 {\IfFileExists{#1.ldf}}}%
969 {\IfFileExists{babel-#1.tex}}{\@namedef{ds@#1}}}%
970 {}%
971 \bbl@ifunset{ds@#1}%
972 {\def\bbl@tempc{#1}%
973 \DeclareOption{#1}{%
974 \ifnum\bbl@iniflag>\@ne
975 \bbl@ldfinit
976 \babelprovide[import]{#1}%
977 \bbl@afterldf}}%
978 \else
979 \bbl@load@language{#1}%
980 \fi}}%
981 {}%
982 \or % provide*=*
983 \def\bbl@tempc{#1}%
984 \bbl@ifunset{ds@#1}%
985 {\DeclareOption{#1}{%
986 \bbl@ldfinit
987 \babelprovide[import]{#1}%
988 \bbl@afterldf}}}%
989 {}%
990 \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.

```

991 \let\bbl@tempb\@nnil
992 \bbl@foreach\@classoptionslist{%
993 \bbl@ifunset{ds@#1}%
994 {\IfFileExists{#1.ldf}%
995 {\def\bbl@tempb{#1}%
996 \DeclareOption{#1}{%
997 \ifnum\bbl@iniflag>\@ne
998 \bbl@ldfinit
999 \babelprovide[import]{#1}%
1000 \bbl@afterldf}}%
1001 \else
1002 \bbl@load@language{#1}%
1003 \fi}}%
1004 {\IfFileExists{babel-#1.tex}% TODO. Copypaste pattern
1005 {\def\bbl@tempb{#1}%
1006 \DeclareOption{#1}{%
1007 \ifnum\bbl@iniflag>\@ne
1008 \bbl@ldfinit
1009 \babelprovide[import]{#1}%
1010 \bbl@afterldf}}%
1011 \else
1012 \bbl@load@language{#1}%
1013 \fi}}%
1014 {}}}%
1015 {}

```

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

```

1016 \ifnum\bbl@iniflag=\z@\else
1017 \ifx\bbl@opt@main\@nnil
1018 \ifx\bbl@tempc\relax

```



```

1019     \let\bbl@opt@main\bbl@tempb
1020   \else
1021     \let\bbl@opt@main\bbl@tempc
1022   \fi
1023 \fi
1024 \fi
1025 \ifx\bbl@opt@main\@nnil\else
1026   \expandafter
1027   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1028   \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1029 \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):

```

1030 \def\AfterBabelLanguage#1{%
1031   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1032 \DeclareOption*{}
1033 \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.

```

1034 \bbl@trace{Option 'main'}
1035 \ifx\bbl@opt@main\@nnil
1036   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1037   \let\bbl@tempc\@empty
1038   \bbl@for\bbl@tempb\bbl@tempa{%
1039     \bbl@xin{,\bbl@tempb,}{,\bbl@loaded,}%
1040     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
1041   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1042   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1043   \ifx\bbl@tempb\bbl@tempc\else
1044     \bbl@warning{%
1045       Last declared language option is '\bbl@tempc',\%
1046       but the last processed one was '\bbl@tempb'.\%
1047       The main language can't be set as both a global\%
1048       and a package option. Use 'main=\bbl@tempc' as\%
1049       option. Reported}%
1050   \fi
1051 \else
1052   \ifodd\bbl@iniflag % case 1,3
1053     \bbl@ldfinit
1054     \let\CurrentOption\bbl@opt@main
1055     \ifx\bbl@opt@provide\@nnil
1056       \bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}%
1057     \else
1058       \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}{}%
1059         \bbl@xin{,provide,}{, #1,}%
1060         \ifin@
1061           \def\bbl@opt@provide{#2}%
1062           \bbl@replace\bbl@opt@provide{;}{,}%
1063         \fi}%
1064       \bbl@exp{%
1065         \bbl@babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
1066       \fi
1067     \bbl@afterldf{}%

```

```

1068 \else % case 0,2
1069 \chardef\bbl@iniflag\z@ % Force ldf
1070 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1071 \ExecuteOptions{\bbl@opt@main}
1072 \DeclareOption*{%
1073 \ProcessOptions*
1074 \fi
1075 \fi
1076 \def\AfterBabelLanguage{%
1077 \bbl@error
1078 {Too late for \string\AfterBabelLanguage}%
1079 {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.

1080 \ifx\bbl@main@language\@undefined
1081 \bbl@info{%
1082 You haven't specified a language. I'll use 'nil'\%
1083 as the main language. Reported}
1084 \bbl@load@language{nil}
1085 \fi
1086 \</package>
1087 \<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 \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

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

8.1 Tools

```

1088 \ifx\ldf@quit\@undefined\else
1089 \endinput\fi % Same line!
1090 \<<Make sure ProvidesFile is defined>>
1091 \ProvidesFile{babel.def}[\<<date>> \<<version>>] Babel common definitions]

```

The file babel.def expects some definitions made in the $\LaTeX 2_{\epsilon}$ style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore an alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```

1092 \ifx\AtBeginDocument\@undefined % TODO. change test.
1093 \<<Emulate LaTeX>>
1094 \def\language{english}%
1095 \let\bbl@opt@shorthands\@nnil
1096 \def\bbl@ifshorthand#1#2#3{#2}%
1097 \let\bbl@language@opts\@empty
1098 \ifx\babeloptionstrings\@undefined
1099 \let\bbl@opt@strings\@nnil
1100 \else

```

```

1101 \let\bbl@opt@strings\babeloptionstrings
1102 \fi
1103 \def\BabelStringsDefault{generic}
1104 \def\bbl@tempa{normal}
1105 \ifx\babeloptionmath\bbl@tempa
1106 \def\bbl@mathnormal{\noexpand\textormath}
1107 \fi
1108 \def\AfterBabelLanguage#1#2{}
1109 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1110 \let\bbl@afterlang\relax
1111 \def\bbl@opt@safe{BR}
1112 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1113 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1114 \expandafter\newif\csname ifbbl@single\endcsname
1115 \chardef\bbl@bidimode\z@
1116 \fi

```

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.

```

1117 <<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.

```

1118 \def\bbl@version{<<version>>}
1119 \def\bbl@date{<<date>>}
1120 \def\adddialect#1#2{%
1121 \global\chardef#1#2\relax
1122 \bbl@usehooks{adddialect}{\{#1\}{#2\}}%
1123 \begingroup
1124 \count@#1\relax
1125 \def\bbl@elt##1##2##3##4{%
1126 \ifnum\count@=##2\relax
1127 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
1128 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
1129 set to \expandafter\string\csname l@##1\endcsname\\%
1130 (\string\language\the\count@). Reported}%
1131 \def\bbl@elt####1####2####3####4}%
1132 \fi}%
1133 \bbl@cs{languages}%
1134 \endgroup}

```

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

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

```

1135 \def\bbl@fixname#1{%
1136 \begingroup
1137 \def\bbl@tempe{l@}%
1138 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1139 \bbl@tempd
1140 {\lowercase\expandafter{\bbl@tempd}%
1141 {\uppercase\expandafter{\bbl@tempd}%

```

```

1142         \@empty
1143         {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1144         \uppercase\expandafter{\bbl@tempd}}}%
1145         {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1146         \lowercase\expandafter{\bbl@tempd}}}%
1147     \@empty
1148     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1149 \bbl@tempd
1150 \bbl@exp{\@bbl@usehooks{language}{\language}{#1}}}%
1151 \def\bbl@iflanguage#1{%
1152 \@ifundefined{1@#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.

```

1153 \def\bbl@bcpcase#1#2#3#4\@#5{%
1154 \ifx\@empty#3%
1155     \uppercase{\def#5{#1#2}}%
1156 \else
1157     \uppercase{\def#5{#1}}%
1158     \lowercase{\edef#5{#5#2#3#4}}%
1159 \fi}
1160 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1161 \let\bbl@bcp\relax
1162 \lowercase{\def\bbl@tempa{#1}}%
1163 \ifx\@empty#2%
1164     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1165 \else\ifx\@empty#3%
1166     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1167     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1168     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1169     {}%
1170 \ifx\bbl@bcp\relax
1171     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1172 \fi
1173 \else
1174     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1175     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1176     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1177     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1178     {}%
1179 \ifx\bbl@bcp\relax
1180     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1181     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1182     {}%
1183 \fi
1184 \ifx\bbl@bcp\relax
1185     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1186     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1187     {}%
1188 \fi
1189 \ifx\bbl@bcp\relax
1190     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1191 \fi
1192 \fi\fi}
1193 \let\bbl@initoload\relax
1194 \def\bbl@provide@locale{%

```

```

1195 \ifx\babelprovide\@undefined
1196   \bbl@error{For a language to be defined on the fly 'base'\\%
1197             is not enough, and the whole package must be\\%
1198             loaded. Either delete the 'base' option or\\%
1199             request the languages explicitly}%
1200             {See the manual for further details.}%
1201 \fi
1202 % TODO. Option to search if loaded, with \LocaleForEach
1203 \let\bbl@auxname\language % Still necessary. TODO
1204 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1205   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1206 \ifbbl@bcp@allowed
1207   \expandafter\ifx\csname date\language\endcsname\relax
1208     \expandafter
1209     \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@
1210     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
1211       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1212       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1213       \expandafter\ifx\csname date\language\endcsname\relax
1214         \let\bbl@initoload\bbl@bcp
1215         \bbl@exp{\@babelprovide[\bbl@autoload@bcpoptions]{\language}}%
1216         \let\bbl@initoload\relax
1217       \fi
1218       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1219     \fi
1220   \fi
1221 \fi
1222 \expandafter\ifx\csname date\language\endcsname\relax
1223   \IfFileExists{babel-\language.tex}%
1224   {\bbl@exp{\@babelprovide[\bbl@autoload@options]{\language}}}%
1225   {}%
1226 \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.

```

1227 \def\iflanguage#1{%
1228   \bbl@iflanguage{#1}{%
1229     \ifnum\csname l@#1\endcsname=\language
1230       \expandafter\@firstoftwo
1231     \else
1232       \expandafter\@secondoftwo
1233     \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.

```

1234 \let\bbl@select@type\z@
1235 \edef\selectlanguage{%
1236   \noexpand\protect
1237   \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`.

```

1238 \ifx\@undefined\protect\let\protect\relax\fi

```

The following definition is preserved for backwards compatibility (eg, arabi, koma). It is related to a trick for 2.09, now discarded.

```
1239 \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.

```
1240 \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`

```
1241 \def\bbl@push@language{%
1242   \ifx\language\undefined\else
1243     \ifx\currentgrouplevel\undefined
1244       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1245     \else
1246       \ifnum\currentgrouplevel=\z@
1247         \xdef\bbl@language@stack{\language+}%
1248       \else
1249         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1250       \fi
1251     \fi
1252 \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`.

```
1253 \def\bbl@pop@lang#1+#2\@@{%
1254   \edef\language{#1}%
1255   \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).

```
1256 \let\bbl@ifrestoring\@secondoftwo
1257 \def\bbl@pop@language{%
1258   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1259   \let\bbl@ifrestoring\@firstoftwo
1260   \expandafter\bbl@set@language\expandafter{\language}%
1261   \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).

```

1262 \chardef\localeid\z@
1263 \def\bbl@id@last{0} % No real need for a new counter
1264 \def\bbl@id@assign{%
1265   \bbl@ifunset{bbl@id@@\language}%
1266   {\count@bbl@id@last\relax
1267    \advance\count@one
1268    \bbl@csarg\chardef{id@@\language}\count@
1269    \edef\bbl@id@last{\the\count@}%
1270    \ifcase\bbl@engine\or
1271      \directlua{
1272        Babel = Babel or {}
1273        Babel.locale_props = Babel.locale_props or {}
1274        Babel.locale_props[bbl@id@last] = {}
1275        Babel.locale_props[bbl@id@last].name = '\language'
1276      }%
1277    \fi}%
1278  }%
1279  \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

1280 \expandafter\def\csname selectlanguage \endcsname#1{%
1281   \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw@fi
1282   \bbl@push@language
1283   \aftergroup\bbl@pop@language
1284   \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).

```

1285 \def\BabelContentsFiles{toc,lof,lot}
1286 \def\bbl@set@language#1{% from selectlanguage, pop@
1287   % The old buggy way. Preserved for compatibility.
1288   \edef\language{%
1289     \ifnum\escapechar=\expandafter\string#1\@empty
1290     \else\string#1\@empty\fi}%
1291   \ifcat\relax\noexpand#1%
1292     \expandafter\ifx\csname date\language\endcsname\relax
1293       \edef\language{#1}%
1294       \let\localename\language
1295     \else
1296       \bbl@info{Using '\string\language' instead of 'language' is\\%
1297         deprecated. If what you want is to use a\\%
1298         macro containing the actual locale, make\\%
1299         sure it does not not match any language.\\%
1300         Reported}%

```

```

1301 \ifx\scantokens\@undefined
1302 \def\localename{??}%
1303 \else
1304 \scantokens\expandafter{\expandafter
1305 \def\expandafter\localename\expandafter{\language}%
1306 \fi
1307 \fi
1308 \else
1309 \def\localename{#1}% This one has the correct catcodes
1310 \fi
1311 \select@language{\language}%
1312 % write to aux
1313 \expandafter\ifx\csize date\language\endcsname\relax\else
1314 \if@files
1315 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1316 \bbl@savelastskip
1317 \protected@write\@auxout{\string\babel@aux{\bbl@auxname}}}%
1318 \bbl@restorelastskip
1319 \fi
1320 \bbl@usehooks{write}}}%
1321 \fi
1322 \fi}
1323 %
1324 \let\bbl@restorelastskip\relax
1325 \def\bbl@savelastskip{%
1326 \let\bbl@restorelastskip\relax
1327 \ifvmode
1328 \ifdim\lastskip=\z@
1329 \let\bbl@restorelastskip\nobreak
1330 \else
1331 \bbl@exp{%
1332 \def\\bbl@restorelastskip{%
1333 \skip@=\the\lastskip
1334 \\nobreak \vskip-\skip@ \vskip\skip@}}%
1335 \fi
1336 \fi}
1337 %
1338 \newif\ifbbl@bcpallowed
1339 \bbl@bcpallowedfalse
1340 \def\select@language#1{% from set@, babel@aux
1341 % set hymap
1342 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
1343 % set name
1344 \edef\language{#1}%
1345 \bbl@fixname\language
1346 % TODO. name@map must be here?
1347 \bbl@provide@locale
1348 \bbl@iflanguage\language{%
1349 \expandafter\ifx\csize date\language\endcsname\relax
1350 \bbl@error
1351 {Unknown language '\language'. Either you have\\%
1352 misspelled its name, it has not been installed,\\%
1353 or you requested it in a previous run. Fix its name,\\%
1354 install it or just rerun the file, respectively. In\\%
1355 some cases, you may need to remove the aux file}%
1356 {You may proceed, but expect wrong results}%
1357 \else
1358 % set type
1359 \let\bbl@select@type\z@

```



```

1360     \expandafter\babel@switch\expandafter{\language}%
1361     \fi}}
1362 \def\babel@aux#1#2{%
1363   \select@language{#1}%
1364   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
1365     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
1366 \def\babel@toc#1#2{%
1367   \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.

```

1368 \newif\ifbbl@usedategroup
1369 \def\bbl@switch#1{% from select@, foreign@
1370   % make sure there is info for the language if so requested
1371   \bbl@ensureinfo{#1}%
1372   % restore
1373   \originalTeX
1374   \expandafter\def\expandafter\originalTeX\expandafter{%
1375     \csname noextras#1\endcsname
1376     \let\originalTeX\empty
1377     \babel@beginsave}%
1378   \bbl@usehooks{afterreset}{}%
1379   \languageshorthands{none}%
1380   % set the locale id
1381   \bbl@id@assign
1382   % switch captions, date
1383   % No text is supposed to be added here, so we remove any
1384   % spurious spaces.
1385   \bbl@bsphack
1386   \ifcase\bbl@select@type
1387     \csname captions#1\endcsname\relax
1388     \csname date#1\endcsname\relax
1389   \else
1390     \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1391     \ifin@
1392       \csname captions#1\endcsname\relax
1393     \fi
1394     \bbl@xin@{,date,}{,\bbl@select@opts,}%
1395     \ifin@ % if \foreign... within \<lang>date
1396       \csname date#1\endcsname\relax
1397     \fi
1398   \fi
1399   \bbl@esphack
1400   % switch extras
1401   \bbl@usehooks{beforeextras}{}%
1402   \csname extras#1\endcsname\relax
1403   \bbl@usehooks{afterextras}{}%
1404   % > babel-ensure
1405   % > babel-sh-<short>

```

```

1406 % > babel-bidi
1407 % > babel-fontspec
1408 % hyphenation - case mapping
1409 \ifcase\bbbl@opt@hyphenmap\or
1410   \def\BabelLower##1##2{\lccode##1=##2\relax}%
1411   \ifnum\bbbl@hymapsel>4\else
1412     \csname\languagename @bbbl@hyphenmap\endcsname
1413   \fi
1414   \chardef\bbbl@opt@hyphenmap\z@
1415 \else
1416   \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
1417     \csname\languagename @bbbl@hyphenmap\endcsname
1418   \fi
1419 \fi
1420 \let\bbbl@hymapsel\@cclv
1421 % hyphenation - select rules
1422 \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
1423   \edef\bbbl@tempa{u}%
1424 \else
1425   \edef\bbbl@tempa{\bbbl@cl{\lnbrk}}%
1426 \fi
1427 % linebreaking - handle u, e, k (v in the future)
1428 \bbbl@xin@{/u}{/\bbbl@tempa}%
1429 \ifin@else\bbbl@xin@{/e}{/\bbbl@tempa}\fi % elongated forms
1430 \ifin@else\bbbl@xin@{/k}{/\bbbl@tempa}\fi % only kashida
1431 \ifin@else\bbbl@xin@{/v}{/\bbbl@tempa}\fi % variable font
1432 \ifin@
1433   % unhyphenated/kashida/elongated = allow stretching
1434   \language\l@unhyphenated
1435   \babel@savevariable\emergencystretch
1436   \emergencystretch\maxdimen
1437   \babel@savevariable\hbadness
1438   \hbadness\@M
1439 \else
1440   % other = select patterns
1441   \bbbl@patterns{#1}%
1442 \fi
1443 % hyphenation - mins
1444 \babel@savevariable\lefthyphenmin
1445 \babel@savevariable\righthyphenmin
1446 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1447   \set@hyphenmins\tw@\thr@\relax
1448 \else
1449   \expandafter\expandafter\expandafter\set@hyphenmins
1450     \csname #1hyphenmins\endcsname\relax
1451 \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.

```

1452 \long\def\otherlanguage#1{%
1453   \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@\fi
1454   \csname selectlanguage \endcsname{#1}%
1455   \ignorespaces}

```

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

mode.

```
1456 \long\def\endotherlanguage{%
1457   \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`.

```
1458 \expandafter\def\csname otherlanguage*\endcsname{%
1459   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1460 \def\bbl@otherlanguage@s[#1]#2{%
1461   \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
1462   \def\bbl@select@opts{#1}%
1463   \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”.

```
1464 \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`.

```
1465 \providecommand\bbl@beforeforeign{}
1466 \edef\foreignlanguage{%
1467   \noexpand\protect
1468   \expandafter\noexpand\csname foreignlanguage \endcsname}
1469 \expandafter\def\csname foreignlanguage \endcsname{%
1470   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1471 \providecommand\bbl@foreign@x[3][{}]{%
1472   \begingroup
1473     \def\bbl@select@opts{#1}%
1474     \let\BabelText\@firstofone
1475     \bbl@beforeforeign
1476     \foreign@language{#2}%
1477     \bbl@usehooks{foreign}{}%
1478     \BabelText{#3}% Now in horizontal mode!
1479   \endgroup}
1480 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@par
1481   \begingroup
1482     {\par}%
1483     \let\bbl@select@opts\@empty
```

```

1484 \let\BabelText\@firstofone
1485 \foreign@language{#1}%
1486 \bbl@usehooks{foreign*}{}%
1487 \bbl@dirparastext
1488 \BabelText{#2}% Still in vertical mode!
1489 {\par}%
1490 \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`.

```

1491 \def\foreign@language#1{%
1492 % set name
1493 \edef\language{#1}%
1494 \ifbbl@usedategroup
1495 \bbl@add\bbl@select@opts{,date,}%
1496 \bbl@usedategroupfalse
1497 \fi
1498 \bbl@fixname\language
1499 % TODO. name@map here?
1500 \bbl@provide@locale
1501 \bbl@iflanguage\language{%
1502 \expandafter\ifx\csname date\language\endcsname\relax
1503 \bbl@warning % TODO - why a warning, not an error?
1504 {Unknown language '#1'. Either you have\\%
1505 misspelled its name, it has not been installed,\\%
1506 or you requested it in a previous run. Fix its name,\\%
1507 install it or just rerun the file, respectively. In\\%
1508 some cases, you may need to remove the aux file.\\%
1509 I'll proceed, but expect wrong results.\\%
1510 Reported}%
1511 \fi
1512 % set type
1513 \let\bbl@select@type\@ne
1514 \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.

```

1515 \let\bbl@hyphlist\@empty
1516 \let\bbl@hyphenation\relax
1517 \let\bbl@pttnlist\@empty
1518 \let\bbl@patterns\relax
1519 \let\bbl@hymapset\@cclv
1520 \def\bbl@patterns#1{%
1521 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1522 \csname l@#1\endcsname
1523 \edef\bbl@tempa{#1}%
1524 \else
1525 \csname l@#1:\f@encoding\endcsname
1526 \edef\bbl@tempa{#1:\f@encoding}%
1527 \fi
1528 \@expandtwoargs\bbl@usehooks{patterns}{\bbl@tempa}}

```

```

1529 % > luatex
1530 \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
1531   \begingroup
1532     \bbl@xin@{\number\language,}{\bbl@hyphlist}%
1533     \ifin\else
1534       \@expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
1535     \hyphenation{%
1536       \bbl@hyphenation@
1537       \@ifundefined{bbl@hyphenation@#1}%
1538       \@empty
1539       {\space\csname bbl@hyphenation@#1\endcsname}}%
1540     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1541   \fi
1542 \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*`.

```

1543 \def\hyphenrules#1{%
1544   \edef\bbl@tempf{#1}%
1545   \bbl@fixname\bbl@tempf
1546   \bbl@iflanguage\bbl@tempf{%
1547     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1548     \ifx\languageshorthands\@undefined\else
1549       \languageshorthands{none}%
1550     \fi
1551     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1552       \set@hyphenmins\tw@\thr@\relax
1553     \else
1554       \expandafter\expandafter\expandafter\set@hyphenmins
1555       \csname\bbl@tempf hyphenmins\endcsname\relax
1556     \fi}}
1557 \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.

```

1558 \def\providehyphenmins#1#2{%
1559   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1560     \@namedef{#1hyphenmins}{#2}%
1561   \fi}

```

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

```

1562 \def\set@hyphenmins#1#2{%
1563   \lefthyphenmin#1\relax
1564   \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.

```

1565 \ifx\ProvidesFile\@undefined
1566   \def\ProvidesLanguage#1[#2 #3 #4]{%
1567     \wlog{Language: #1 #4 #3 <#2>}%
1568   }
1569 \else

```

```

1570 \def\ProvidesLanguage#1{%
1571     \begingroup
1572     \catcode`\ 10 %
1573     \@makeother\/%
1574     \@ifnextchar[%]
1575         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1576 \def\@provideslanguage#1[#2]{%
1577     \wlog{Language: #1 #2}%
1578     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1579     \endgroup}
1580 \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`.

```
1581 \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.

```
1582 \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’:

```

1583 \providecommand\setlocale{%
1584     \bbl@error
1585     {Not yet available}%
1586     {Find an armchair, sit down and wait}}
1587 \let\uselocale\setlocale
1588 \let\locale\setlocale
1589 \let\selectlocale\setlocale
1590 \let\localename\setlocale
1591 \let\textlocale\setlocale
1592 \let\textlanguage\setlocale
1593 \let\languagetext\setlocale

```

9.2 Errors

`\@nolanerr` The babel package will signal an error when a documents tries to select a language that hasn’t been defined earlier. When a user selects a language for which no hyphenation patterns were loaded into the format he will be given a warning about that fact. We revert to the patterns for `\language=0` in that case. In most formats that will be (US)english, but it might also be empty.

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

```

1594 \edef\bbl@nulllanguage{\string\language=0}
1595 \ifx\PackageError\undefined % TODO. Move to Plain
1596     \def\bbl@error#1#2{%
1597         \begingroup
1598         \newlinechar=`^^J
1599         \def\{^^J(babel) }%
1600         \errhelp{#2}\errmessage{\{#1}%
1601         \endgroup}
1602 \def\bbl@warning#1{%
1603     \begingroup
1604     \newlinechar=`^^J
1605     \def\{^^J(babel) }%

```

```

1606     \message{\#1}%
1607   \endgroup}
1608   \let\bbl@infowarn\bbl@warning
1609   \def\bbl@info#1{%
1610     \begingroup
1611       \newlinechar=`^^J
1612       \def\{^^J}%
1613       \wlog{#1}%
1614     \endgroup}
1615 \fi
1616 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1617 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1618   \global\@namedef{#2}{\textbf{?#1?}}%
1619   \@nameuse{#2}%
1620   \edef\bbl@tempa{#1}%
1621   \bbl@sreplace\bbl@tempa{name}{}%
1622   \bbl@warning{% TODO.
1623     \@backslashchar#1 not set for '\language'. Please,\%
1624     define it after the language has been loaded\%
1625     (typically in the preamble) with:\%
1626     \string\setlocalecaption{\language}{\bbl@tempa}{..\%
1627     Reported}}
1628 \def\bbl@tentative{\protect\bbl@tentative@i}
1629 \def\bbl@tentative@i#1{%
1630   \bbl@warning{%
1631     Some functions for '#1' are tentative.\%
1632     They might not work as expected and their behavior\%
1633     could change in the future.\%
1634     Reported}}
1635 \def\@nolanerr#1{%
1636   \bbl@error
1637   {You haven't defined the language '#1' yet.\%
1638     Perhaps you misspelled it or your installation\%
1639     is not complete}%
1640   {Your command will be ignored, type <return> to proceed}}
1641 \def\@nopatterns#1{%
1642   \bbl@warning
1643   {No hyphenation patterns were preloaded for\%
1644     the language '#1' into the format.\%
1645     Please, configure your TeX system to add them and\%
1646     rebuild the format. Now I will use the patterns\%
1647     preloaded for \bbl@nulllanguage\space instead}}
1648 \let\bbl@usehooks\@gobbletwo
1649 \ifx\bbl@onlyswitch\@empty\endinput\fi
1650 % Here ended switch.def

Here ended switch.def.

1651 \ifx\directlua\@undefined\else
1652   \ifx\bbl@luapatterns\@undefined
1653     \input luababel.def
1654   \fi
1655 \fi
1656 <<Basic macros>>
1657 \bbl@trace{Compatibility with language.def}
1658 \ifx\bbl@languages\@undefined
1659   \ifx\directlua\@undefined
1660     \openin1 = language.def % TODO. Remove hardcoded number
1661     \ifeof1
1662       \closein1

```

```

1663     \message{I couldn't find the file language.def}
1664   \else
1665     \closein1
1666     \begingroup
1667       \def\addlanguage#1#2#3#4#5{%
1668         \expandafter\ifx\csname lang@#1\endcsname\relax\else
1669           \global\expandafter\let\csname l@#1\expandafter\endcsname
1670             \csname lang@#1\endcsname
1671         \fi}%
1672       \def\uselanguage#1{%
1673         \input language.def
1674       \endgroup
1675     \fi
1676   \fi
1677   \chardef\l@english\z@
1678 \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.

```

1679 \def\addto#1#2{%
1680   \ifx#1\@undefined
1681     \def#1{#2}%
1682   \else
1683     \ifx#1\relax
1684       \def#1{#2}%
1685     \else
1686       {\toks@\expandafter{#1#2}%
1687        \xdef#1{\the\toks@}}%
1688     \fi
1689   \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?

```

1690 \def\bbl@withactive#1#2{%
1691   \begingroup
1692   \lccode`~=#2\relax
1693   \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`.

```

1694 \def\bbl@redefine#1{%
1695   \edef\bbl@tempa{\bbl@stripslash#1}%
1696   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1697   \expandafter\def\csname\bbl@tempa\endcsname{
1698     \@onlypreamble\bbl@redefine

```

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

```

1699 \def\bbl@redefine@long#1{%
1700   \edef\bbl@tempa{\bbl@stripslash#1}%
1701   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1702   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1703     \@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`.

```
1704 \def\bbl@redefineroobust#1{%
1705   \edef\bbl@tempa{\bbl@stripslash#1}%
1706   \bbl@ifunset{\bbl@tempa\space}%
1707   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1708    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1709   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1710   \@namedef{\bbl@tempa\space}}
1711 \@onlypreamble\bbl@redefineroobust
```

9.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by babel to execute hooks defined for an event.

```
1712 \bbl@trace{Hooks}
1713 \newcommand\AddBabelHook[3][]{%
1714   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}}%
1715   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1716   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1717   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1718   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1719   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1720   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1721 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1722 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1723 \def\bbl@usehooks#1#2{%
1724   \ifx\UseHook\@undefined\else\UseHook{babel/#1}\fi
1725   \def\bbl@elth##1{%
1726     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1727     \bbl@cs{ev@#1@}%
1728     \ifx\language\@undefined\else % Test required for Plain (?)
1729       \ifx\UseHook\@undefined\else\UseHook{babel/#1/\language}\fi
1730       \def\bbl@elth##1{%
1731         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1732         \bbl@cl{ev@#1}%
1733       \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).

```
1734 \def\bbl@evargs{,% <- don't delete this comma
1735   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1736   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1737   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1738   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1739   beforestart=0,language=2}
1740 \ifx\NewHook\@undefined\else
1741   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1742   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1743 \fi
```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro `\bbl@e@<language>` contains `\bbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1744 \bbl@trace{Defining babelensure}
1745 \newcommand\babelensure[2][{}]{% TODO - revise test files
1746   \AddBabelHook{babel-ensure}{afterextras}{%
1747     \ifcase\bbl@select@type
1748       \bbl@cl{e}%
1749     \fi}%
1750   \begingroup
1751     \let\bbl@ens@include\@empty
1752     \let\bbl@ens@exclude\@empty
1753     \def\bbl@ens@fontenc{\relax}%
1754     \def\bbl@tempb##1{%
1755       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1756     \edef\bbl@tempa{\bbl@tempb##1\@empty}%
1757     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1758     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1759     \def\bbl@tempc{\bbl@ensure}%
1760     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1761       \expandafter{\bbl@ens@include}}%
1762     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1763       \expandafter{\bbl@ens@exclude}}%
1764     \toks@\expandafter{\bbl@tempc}%
1765     \bbl@exp{%
1766   \endgroup
1767   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1768 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1769   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1770     \ifx##1\undefined % 3.32 - Don't assume the macro exists
1771       \edef##1{\noexpand\bbl@nocaption
1772         {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1773     \fi
1774     \ifx##1\@empty\else
1775       \in@{##1}{#2}%
1776       \ifin@ \else
1777         \bbl@ifunset{\bbl@ensure@\language\name}%
1778         {\bbl@exp{%
1779           \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1780             \\\foreignlanguage{\language\name}%
1781             {\ifx\relax#3\else
1782               \\\fontencoding{#3}\selectfont
1783             \fi
1784             #####1}}}%
1785         }%
1786         \toks@\expandafter{##1}%
1787         \edef##1{%
1788           \bbl@csarg\noexpand{ensure@\language\name}%
1789           {\the\toks@}}%
1790       \fi
1791       \expandafter\bbl@tempb
1792     \fi}%
1793   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1794   \def\bbl@tempa##1{% elt for include list
1795     \ifx##1\@empty\else
1796       \bbl@csarg\in@{ensure@\language\name\expandafter}\expandafter{##1}%

```

```

1797 \ifin@else
1798 \bbl@tempb##1\@empty
1799 \fi
1800 \expandafter\bbl@tempa
1801 \fi}%
1802 \bbl@tempa#1\@empty}
1803 \def\bbl@captionslist{%
1804 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1805 \contentsname\listfigurename\listtablename\indexname\figurename
1806 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1807 \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.

```

1808 \bbl@trace{Macros for setting language files up}
1809 \def\bbl@ldfinit{%
1810 \let\bbl@screset\@empty
1811 \let\BabelStrings\bbl@opt@string
1812 \let\BabelOptions\@empty
1813 \let\BabelLanguages\relax
1814 \ifx\originalTeX\@undefined
1815 \let\originalTeX\@empty
1816 \else
1817 \originalTeX
1818 \fi}
1819 \def\LdfInit#1#2{%
1820 \chardef\atcatcode=\catcode`\@
1821 \catcode`\@=11\relax
1822 \chardef\eqcatcode=\catcode`\=
1823 \catcode`\==12\relax
1824 \expandafter\if\expandafter\@backslashchar
1825 \expandafter\@car\string#2\@nil
1826 \ifx#2\@undefined\else
1827 \ldf@quit{#1}%
1828 \fi
1829 \else
1830 \expandafter\ifx\csname#2\endcsname\relax\else
1831 \ldf@quit{#1}%
1832 \fi
1833 \fi

```

```

1834 \bbl@ldfinit}

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

1835 \def\ldf@quit#1{%
1836 \expandafter\main@language\expandafter{#1}%
1837 \catcode`\@=\atcatcode \let\atcatcode\relax
1838 \catcode`\==\eqcatcode \let\eqcatcode\relax
1839 \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.

1840 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1841 \bbl@afterlang
1842 \let\bbl@afterlang\relax
1843 \let\BabelModifiers\relax
1844 \let\bbl@screaset\relax}%
1845 \def\ldf@finish#1{%
1846 \loadlocalcfg{#1}%
1847 \bbl@afterldf{#1}%
1848 \expandafter\main@language\expandafter{#1}%
1849 \catcode`\@=\atcatcode \let\atcatcode\relax
1850 \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  $\LaTeX$ .

1851 \@onlypreamble\LdfInit
1852 \@onlypreamble\ldf@quit
1853 \@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.

1854 \def\main@language#1{%
1855 \def\bbl@main@language{#1}%
1856 \let\language\bbl@main@language % TODO. Set localname
1857 \bbl@id@assign
1858 \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.

1859 \def\bbl@beforestart{%
1860 \def\@nolanerr##1{%
1861 \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1862 \bbl@usehooks{beforestart}}}%
1863 \global\let\bbl@beforestart\relax}
1864 \AtBeginDocument{%
1865 {\@nameuse{bbl@beforestart}}% Group!
1866 \if@filesw
1867 \providecommand\babel@aux[2]{}%
1868 \immediate\write\@mainaux{%
1869 \string\providecommand\string\babel@aux[2]{}%
1870 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1871 \fi
1872 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1873 \ifbbl@single % must go after the line above.

```

```

1874 \renewcommand\selectlanguage[1]{}%
1875 \renewcommand\foreignlanguage[2]{#2}%
1876 \global\let\babel@aux\@gobbles % Also as flag
1877 \fi
1878 \ifcase\bbbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1879 \def\select@language@x#1{%
1880 \ifcase\bbbl@select@type
1881 \bbbl@ifsamestring\language{#1}{\select@language{#1}}%
1882 \else
1883 \select@language{#1}%
1884 \fi}

```

9.5 Shorthands

\bbbl@add@special The macro `\bbbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if \LaTeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1885 \bbbl@trace{Shorthands}
1886 \def\bbbl@add@special#1{% 1:a macro like \", \?, etc.
1887 \bbbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1888 \bbbl@ifunset{\@sanitize}{\bbbl@add\@sanitize{\@makeother#1}}%
1889 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1890 \begingroup
1891 \catcode`#1\active
1892 \nfss@catcodes
1893 \ifnum\catcode`#1=\active
1894 \endgroup
1895 \bbbl@add\nfss@catcodes{\@makeother#1}%
1896 \else
1897 \endgroup
1898 \fi
1899 \fi}

```

\bbbl@remove@special The companion of the former macro is `\bbbl@remove@special`. It removes a character from the set macros `\dospecials` and `\@sanitize`, but it is not used at all in the babel core.

```

1900 \def\bbbl@remove@special#1{%
1901 \begingroup
1902 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1903 \else\noexpand##1\noexpand##2\fi}%
1904 \def\do{\x\do}%
1905 \def\@makeother{\x\@makeother}%
1906 \edef\x{\endgroup
1907 \def\noexpand\dospecials{\dospecials}%
1908 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1909 \def\noexpand\@sanitize{\@sanitize}%
1910 \fi}%
1911 \x}

```

\initiate@active@char A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char` (*char*) to expand to the character in its 'normal state' and it defines the active character to expand to

`\normal@char⟨char⟩` by default (`⟨char⟩` being the character to be made active). Later its definition can be changed to expand to `\active@char⟨char⟩` by calling `\bbl@activate{⟨char⟩}`. For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines `"` as `\active@prefix "\active@char"` (where the first `"` is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original `"`); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char"`. The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, `<level>@group`, `<level>@active` and `<next-level>@active` (except in system).

```

1912 \def\bbl@active@def#1#2#3#4{%
1913   \@namedef{#3#1}{%
1914     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1915       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1916     \else
1917       \bbl@afterfi\csname#2@sh@#1\endcsname
1918     \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.

```

1919   \long\@namedef{#3@arg#1}##1{%
1920     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1921       \bbl@afterelse\csname#4#1\endcsname##1%
1922     \else
1923       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1924     \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.

```

1925 \def\initiate@active@char#1{%
1926   \bbl@ifunset{active@char\string#1}%
1927   {\bbl@withactive
1928     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1929   {}}

```

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).

```

1930 \def\@initiate@active@char#1#2#3{%
1931   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1932   \ifx#1\@undefined
1933     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1934   \else
1935     \bbl@csarg\let{oridef@#2}#1%
1936     \bbl@csarg\edef{oridef@#2}{%
1937       \let\noexpand#1%
1938       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1939   \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`).

```

1940 \ifx#1#3\relax
1941 \expandafter\let\csname normal@char#2\endcsname#3%
1942 \else
1943 \bbl@info{Making #2 an active character}%
1944 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1945 \@namedef{normal@char#2}{%
1946 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1947 \else
1948 \@namedef{normal@char#2}{#3}%
1949 \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).

```

1950 \bbl@restoreactive{#2}%
1951 \AtBeginDocument{%
1952 \catcode`#2\active
1953 \if@filesw
1954 \immediate\write\@mainaux{\catcode`\string#2\active}%
1955 \fi}%
1956 \expandafter\bbl@add@special\csname#2\endcsname
1957 \catcode`#2\active
1958 \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⟩`).

```

1959 \let\bbl@tempa\@firstoftwo
1960 \if\string^#2%
1961 \def\bbl@tempa{\noexpand\textormath}%
1962 \else
1963 \ifx\bbl@mathnormal\@undefined\else
1964 \let\bbl@tempa\bbl@mathnormal
1965 \fi
1966 \fi
1967 \expandafter\edef\csname active@char#2\endcsname{%
1968 \bbl@tempa
1969 {\noexpand\if@safe@actives
1970 \noexpand\expandafter
1971 \expandafter\noexpand\csname normal@char#2\endcsname
1972 \noexpand\else
1973 \noexpand\expandafter
1974 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1975 \noexpand\fi}%
1976 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1977 \bbl@csarg\edef{doactive#2}{%
1978 \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!).

```

1979 \bbl@csarg\edef{active@#2}{%

```

```

1980 \noexpand\active@prefix\noexpand#1%
1981 \expandafter\noexpand\csname active@char#2\endcsname}%
1982 \bbl@csarg\edef{normal@#2}{%
1983 \noexpand\active@prefix\noexpand#1%
1984 \expandafter\noexpand\csname normal@char#2\endcsname}%
1985 \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.

```

1986 \bbl@active@def#2\user@group{user@active}{language@active}%
1987 \bbl@active@def#2\language@group{language@active}{system@active}%
1988 \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.

```

1989 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1990 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1991 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1992 {\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@ms` 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.

```

1993 \if\string'#2%
1994 \let\prim@s\bbl@prim@s
1995 \let\active@math@prime#1%
1996 \fi
1997 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}

```

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

```

1998 <<(*More package options)>> ≡
1999 \DeclareOption{math=active}{%
2000 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2001 <</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.

```

2002 \@ifpackagewith{babel}{KeepShorthandsActive}%
2003 {\let\bbl@restoreactive\@gobble}%
2004 {\def\bbl@restoreactive#1{%
2005 \bbl@exp{%
2006 \\\AfterBabelLanguage\\CurrentOption
2007 {\catcode`#1=\the\catcode`#1\relax}%
2008 \\\AtEndOfPackage
2009 {\catcode`#1=\the\catcode`#1\relax}}}%
2010 \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.


```

2011 \def\bbl@sh@select#1#2{%
2012   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2013     \bbl@afterelse\bbl@scndcs
2014   \else
2015     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2016   \fi}

```

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

```

2017 \begingroup
2018 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2019 {\gdef\active@prefix#1{%
2020   \ifx\protect\@typeset@protect
2021   \else
2022     \ifx\protect\@unexpandable@protect
2023       \noexpand#1%
2024     \else
2025       \protect#1%
2026     \fi
2027     \expandafter\@gobble
2028   \fi}}
2029 {\gdef\active@prefix#1{%
2030   \ifincsname
2031     \string#1%
2032     \expandafter\@gobble
2033   \else
2034     \ifx\protect\@typeset@protect
2035     \else
2036       \ifx\protect\@unexpandable@protect
2037         \noexpand#1%
2038       \else
2039         \protect#1%
2040       \fi
2041       \expandafter\expandafter\expandafter\@gobble
2042     \fi
2043   \fi}}
2044 \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⟩`.

```

2045 \newif\if@safe@actives
2046 \@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.

```

2047 \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`.

```

2048 \chardef\bbl@activated\z@
2049 \def\bbl@activate#1{%
2050   \chardef\bbl@activated\@ne

```

```

2051 \bbl@withactive{\expandafter\let\expandafter}#1%
2052 \csname bbl@active@\string#1\endcsname}
2053 \def\bbl@deactivate#1{%
2054 \chardef\bbl@activated\tw@
2055 \bbl@withactive{\expandafter\let\expandafter}#1%
2056 \csname bbl@normal@\string#1\endcsname}

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
2057 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2058 \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.

2059 \def\babel@texpdf#1#2#3#4{%
2060 \ifx\texorpdfstring\undefined
2061 \textormath{#1}{#3}%
2062 \else
2063 \texorpdfstring{\textormath{#1}{#3}}{#2}%
2064 % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2065 \fi}
2066 %
2067 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2068 \def\@decl@short#1#2#3\@nil#4{%
2069 \def\bbl@tempa{#3}%
2070 \ifx\bbl@tempa\@empty
2071 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2072 \bbl@ifunset{#1@sh@\string#2@}{}%
2073 {\def\bbl@tempa{#4}%
2074 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2075 \else
2076 \bbl@info
2077 {Redefining #1 shorthand \string#2\\
2078 in language \CurrentOption}%
2079 \fi}%
2080 \@namedef{#1@sh@\string#2@}{#4}%
2081 \else
2082 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2083 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2084 {\def\bbl@tempa{#4}%
2085 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2086 \else
2087 \bbl@info
2088 {Redefining #1 shorthand \string#2\string#3\\
2089 in language \CurrentOption}%
2090 \fi}%
2091 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2092 \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.

```

```

2093 \def\textormath{%
2094   \ifmmode
2095     \expandafter\@secondoftwo
2096   \else
2097     \expandafter\@firstoftwo
2098   \fi}

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

2099 \def\user@group{user}
2100 \def\language@group{english} % TODO. I don't like defaults
2101 \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.

2102 \def\useshorthands{%
2103   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
2104 \def\bb1@usesh@s#1{%
2105   \bb1@usesh@x
2106   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
2107   {#1}}
2108 \def\bb1@usesh@x#1#2{%
2109   \bb1@ifshorthand{#2}%
2110   {\def\user@group{user}%
2111     \initiate@active@char{#2}%
2112     #1%
2113     \bb1@activate{#2}}%
2114   {\bb1@error
2115     {I can't declare a shorthand turned off (\string#2)}
2116     {Sorry, but you can't use shorthands which have been\\
2117       turned off in the package options}}}

\defineshorthand Currently we only support two groups of user level shorthands, named internally user and
                  user@<lang> (language-dependent user shorthands). By default, only the first one is taken into
                  account, but if the former is also used (in the optional argument of \defineshorthand) a new level is
                  inserted for it (user@generic, done by \bb1@set@user@generic); we make also sure {} and
                  \protect are taken into account in this new top level.

2118 \def\user@language@group{user@\language@group}
2119 \def\bb1@set@user@generic#1#2{%
2120   \bb1@ifunset{user@generic@active#1}%
2121   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
2122     \bb1@active@def#1\user@group{user@generic@active}{language@active}%
2123     \expandafter\edef\csname#2@sh@#1@\endcsname{%
2124       \expandafter\noexpand\csname normal@char#1\endcsname}%
2125     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2126       \expandafter\noexpand\csname user@active#1\endcsname}}%
2127   \@empty}
2128 \newcommand\defineshorthand[3][user]{%
2129   \edef\bb1@tempa{\zap@space#1 \@empty}%
2130   \bb1@for\bb1@tempb\bb1@tempa{%
2131     \if*\expandafter\@car\bb1@tempb\@nil
2132       \edef\bb1@tempb{user@\expandafter@gobble\bb1@tempb}%
2133       \@expandtwoargs
2134       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
2135     \fi
2136     \declare@shorthand{\bb1@tempb}{#2}{#3}}}

```

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

```
2137 \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`.

```
2138 \def\aliasshorthand#1#2{%
2139   \bbl@ifshorthand{#2}%
2140   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2141     \ifx\document\@notprerr
2142       \@notshorthand{#2}%
2143     \else
2144       \initiate@active@char{#2}%
2145       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2146       \csname active@char\string#1\endcsname
2147       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2148       \csname normal@char\string#1\endcsname
2149       \bbl@activate{#2}%
2150     \fi
2151   \fi}%
2152   {\bbl@error
2153     {Cannot declare a shorthand turned off (\string#2)}
2154     {Sorry, but you cannot use shorthands which have been\\%
2155       turned off in the package options}}}
```

`\@notshorthand`

```
2156 \def\@notshorthand#1{%
2157   \bbl@error{%
2158     The character '\string #1' should be made a shorthand character;\\%
2159     add the command \string\usesshorthands\string{#1\string} to
2160     the preamble.\\%
2161     I will ignore your instruction}%
2162   {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.

```
2163 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2164 \DeclareRobustCommand*\shorthandoff{%
2165   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2166 \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.

```
2167 \def\bbl@switch@sh#1#2{%
2168   \ifx#2\@nnil\else
2169     \bbl@ifunset{\bbl@active@\string#2}%
2170     {\bbl@error
2171       {I can't switch '\string#2' on or off--not a shorthand}%
2172       {This character is not a shorthand. Maybe you made\\%
2173         a typing mistake? I will ignore your instruction.}}%
2174     {\ifcase#1%   off, on, off*
```

```

2175 \catcode`#212\relax
2176 \or
2177 \catcode`#2\active
2178 \bbl@ifunset{bbl@shdef@\string#2}%
2179 {}%
2180 {\bbl@withactive{\expandafter\let\expandafter}#2%
2181 \csname bbl@shdef@\string#2\endcsname
2182 \bbl@csarg\let{shdef@\string#2}\relax}%
2183 \ifcase\bbl@activated\or
2184 \bbl@activate{#2}%
2185 \else
2186 \bbl@deactivate{#2}%
2187 \fi
2188 \or
2189 \bbl@ifunset{bbl@shdef@\string#2}%
2190 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
2191 {}%
2192 \csname bbl@oricat@\string#2\endcsname
2193 \csname bbl@oridef@\string#2\endcsname
2194 \fi}%
2195 \bbl@afterfi\bbl@switch@sh#1%
2196 \fi}

```

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

```

2197 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2198 \def\bbl@putsh#1{%
2199 \bbl@ifunset{bbl@active@\string#1}%
2200 {\bbl@putsh@i#1\@empty\@nnil}%
2201 {\csname bbl@active@\string#1\endcsname}}
2202 \def\bbl@putsh@i#1#2\@nnil{%
2203 \csname\language@group @sh@\string#1@%
2204 \ifx\@empty#2\else\string#2@\fi\endcsname}
2205 \ifx\bbl@opt@shorthands\@nnil\else
2206 \let\bbl@s@initiate@active@char\initiate@active@char
2207 \def\initiate@active@char#1{%
2208 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2209 \let\bbl@s@switch@sh\bbl@switch@sh
2210 \def\bbl@switch@sh#1#2{%
2211 \ifx#2\@nnil\else
2212 \bbl@afterfi
2213 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2214 \fi}
2215 \let\bbl@s@activate\bbl@activate
2216 \def\bbl@activate#1{%
2217 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2218 \let\bbl@s@deactivate\bbl@deactivate
2219 \def\bbl@deactivate#1{%
2220 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2221 \fi

```

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

```

2222 \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.

```

2223 \def\bbl@prim@s{%

```

```

2224 \prime\futurelet\@let@token\bbl@pr@m@s}
2225 \def\bbl@if@primes#1#2{%
2226 \ifx#1\@let@token
2227 \expandafter\@firstoftwo
2228 \else\ifx#2\@let@token
2229 \bbl@afterelse\expandafter\@firstoftwo
2230 \else
2231 \bbl@afterfi\expandafter\@secondoftwo
2232 \fi\fi}
2233 \begingroup
2234 \catcode`\^=7 \catcode`\*= \active \lccode`\*=`^^
2235 \catcode`\'=12 \catcode`\`= \active \lccode`\`=``'
2236 \lowercase{%
2237 \gdef\bbl@pr@m@s{%
2238 \bbl@if@primes"%
2239 \pr@@@s
2240 {\bbl@if@primes*\pr@@@t\egroup}}
2241 \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).

```

2242 \initiate@active@char{~}
2243 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2244 \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.

```

2245 \expandafter\def\csname OT1dqpos\endcsname{127}
2246 \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

```

2247 \ifx\f@encoding\@undefined
2248 \def\f@encoding{OT1}
2249 \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.

```

2250 \bbl@trace{Language attributes}
2251 \newcommand\languageattribute[2]{%
2252 \def\bbl@tempc{#1}%
2253 \bbl@fixname\bbl@tempc
2254 \bbl@iflanguage\bbl@tempc{%
2255 \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.

```

2256 \ifx\babelknown@attribs\@undefined
2257 \in@false
2258 \else
2259 \babelxin@{,\babeltempc-##1,}{,\babelknown@attribs,}%
2260 \fi
2261 \ifin@
2262 \babelwarning{%
2263 You have more than once selected the attribute '##1'\%
2264 for language #1. Reported}%
2265 \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.

```

2266 \babelexp{%
2267 \\\babeladd@list\\babelknown@attribs{\babeltempc-##1}}%
2268 \edef\babeltempa{\babeltempc-##1}%
2269 \expandafter\babelifknown@trib\expandafter{\babeltempa}\babelattributes%
2270 {\csname\babeltempc @attr##1\endcsname}%
2271 {\@attrerr{\babeltempc}{##1}}%
2272 \fi}}
2273 \@onlypreamble\languageattribute

```

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

```

2274 \newcommand*{\@attrerr}[2]{%
2275 \babelerror
2276 {The attribute #2 is unknown for language #1.}%
2277 {Your command will be ignored, type <return> to proceed}}

```

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

```

2278 \def\babeldeclare@ttribute#1#2#3{%
2279 \babelxin@{,#2,}{,\BabelModifiers,}%
2280 \ifin@
2281 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2282 \fi
2283 \babeladd@list\babelattributes{#1-#2}%
2284 \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

\babelifattributeset 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.

```

2285 \def\babelifattributeset#1#2#3#4{%
2286 \ifx\babelknown@attribs\@undefined
2287 \in@false
2288 \else
2289 \babelxin@{,#1-#2,}{,\babelknown@attribs,}%
2290 \fi
2291 \ifin@
2292 \babelafterelse#3%
2293 \else
2294 \babelafterfi#4%
2295 \fi}

```

\babelifknown@trib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise.

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

```

2296 \def\bbl@ifknown@ttrib#1#2{%
2297   \let\bbl@tempa\@secondoftwo
2298   \bbl@loopx\bbl@tempb{#2}{%
2299     \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
2300     \ifin@
2301     \let\bbl@tempa\@firstoftwo
2302     \else
2303     \fi}%
2304   \bbl@tempa}

```

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

```

2305 \def\bbl@clear@ttribs{%
2306   \ifx\bbl@attributes\undefined\else
2307     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2308       \expandafter\bbl@clear@ttrib\bbl@tempa.
2309     }%
2310     \let\bbl@attributes\undefined
2311   \fi}
2312 \def\bbl@clear@ttrib#1-#2.{%
2313   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2314 \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` 2315 `\bbl@trace{Macros for saving definitions}`
 2316 `\def\babel@beginsave{\babel@savecnt\z@}`

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

```

2317 \newcount\babel@savecnt
2318 \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.

```

2319 \def\babel@save#1{%
2320   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2321   \toks@\expandafter{\originalTeX\let#1=}%
2322   \bbl@exp{%
2323     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2324   \advance\babel@savecnt\@ne}
2325 \def\babel@savevariable#1{%
2326   \toks@\expandafter{\originalTeX #1=}%
2327   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

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

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

```

2328 \def\bbl@frenchspacing{%
2329   \ifnum\the\sfcode`\.=\@m
2330     \let\bbl@nonfrenchspacing\relax
2331   \else
2332     \frenchspacing
2333     \let\bbl@nonfrenchspacing\nonfrenchspacing
2334   \fi}
2335 \let\bbl@nonfrenchspacing\nonfrenchspacing
2336 \let\bbl@elt\relax
2337 \edef\bbl@fs@chars{%
2338   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2339   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2340   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
2341 \def\bbl@pre@fs{%
2342   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
2343   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
2344 \def\bbl@post@fs{%
2345   \bbl@save@sfcodes
2346   \edef\bbl@tempa{\bbl@cl{frspc}}%
2347   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\nil}%
2348   \if u\bbl@tempa      % do nothing
2349   \else\if n\bbl@tempa % non french
2350     \def\bbl@elt##1##2##3{%
2351       \ifnum\sfcode`##1=##2\relax
2352         \babel@savevariable{\sfcode`##1}%
2353         \sfcode`##1=##3\relax
2354       \fi}%
2355     \bbl@fs@chars
2356   \else\if y\bbl@tempa % french
2357     \def\bbl@elt##1##2##3{%
2358       \ifnum\sfcode`##1=##3\relax
2359         \babel@savevariable{\sfcode`##1}%
2360         \sfcode`##1=##2\relax
2361       \fi}%
2362     \bbl@fs@chars
2363   \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.

```

2364 \bbl@trace{Short tags}
2365 \def\babeltags#1{%
2366   \edef\bbl@tempa{\zap@space#1 \@empty}%
2367   \def\bbl@tempb##1=##2\@{#1}%
2368   \edef\bbl@tempc{%
2369     \noexpand\newcommand
2370     \expandafter\noexpand\csname #1\endcsname{%
2371       \noexpand\protect
2372       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2373     \noexpand\newcommand
2374     \expandafter\noexpand\csname text##1\endcsname{%

```

```

2375 \noexpand\foreignlanguage{##2}}
2376 \bbl@tempc}%
2377 \bbl@for\bbl@tempa\bbl@tempa{%
2378 \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.

```

2379 \bbl@trace{Hyphens}
2380 \@onlypreamble\babelhyphenation
2381 \AtEndOfPackage{%
2382 \newcommand\babelhyphenation[2][\@empty]{%
2383 \ifx\bbl@hyphenation@ \relax
2384 \let\bbl@hyphenation@\@empty
2385 \fi
2386 \ifx\bbl@hyphlist\@empty\else
2387 \bbl@warning{%
2388 You must not intermingle \string\selectlanguage\space and\\
2389 \string\babelhyphenation\space or some exceptions will not\\
2390 be taken into account. Reported}%
2391 \fi
2392 \ifx\@empty#1%
2393 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2394 \else
2395 \bbl@vforeach{#1}{%
2396 \def\bbl@tempa{##1}%
2397 \bbl@fixname\bbl@tempa
2398 \bbl@iflanguage\bbl@tempa{%
2399 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2400 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2401 {}%
2402 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2403 #2}}}%
2404 \fi}}

```

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

```

2405 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2406 \def\bbl@t@one{T1}
2407 \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`.

```

2408 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2409 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2410 \def\bbl@hyphen{%
2411 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2412 \def\bbl@hyphen@i#1#2{%
2413 \bbl@ifunset{bbl@hy@#1#2\@empty}%
2414 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2415 {\csname bbl@hy@#1#2\@empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word – the version with a single `@` is used when further hyphenation is allowed, while that with `@@` if

³² $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

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

```
2416 \def\bbl@usehyphen#1{%
2417   \leavevmode
2418   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2419   \nobreak\hskip\z@skip}
2420 \def\bbl@usehyphen#1{%
2421   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
2422 \def\bbl@hyphenchar{%
2423   \ifnum\hyphenchar\font=\m@ne
2424     \babeinullhyphen
2425   \else
2426     \char\hyphenchar\font
2427   \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.

```
2428 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2429 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2430 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2431 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2432 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
2433 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2434 \def\bbl@hy@repeat{%
2435   \bbl@usehyphen%
2436   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2437 \def\bbl@hy@repeat{%
2438   \bbl@usehyphen%
2439   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2440 \def\bbl@hy@empty{\hskip\z@skip}
2441 \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.

```
2442 \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.

```
2443 \bbl@trace{Multiencoding strings}
2444 \def\bbl@tglobal#1{\global\let#1#1}
2445 \def\bbl@recatcode#1{% TODO. Used only once?
2446   \@tempcnta="7F
2447   \def\bbl@tempa{%
2448     \ifnum\@tempcnta>"FF\else
2449       \catcode\@tempcnta=#1\relax
2450       \advance\@tempcnta\@ne
2451       \expandafter\bbl@tempa
```

```

2452   \fi}%
2453   \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@ucllc`. The parser is restarted inside `\<lang>\bbl@ucllc` 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).

```

2454 \@ifpackagewith{babel}{nocase}%
2455   {\let\bbl@patchucllc\relax}%
2456   {\def\bbl@patchucllc{%
2457     \global\let\bbl@patchucllc\relax
2458     \g@addto\macro\@uclclist{\reserved@b{\reserved@b\bbl@ucllc}}%
2459     \gdef\bbl@ucllc##1{%
2460       \let\bbl@encoded\bbl@encoded@ucllc
2461       \bbl@ifunset{\language @bbl@ucllc}% and resumes it
2462       {##1}%
2463       {\let\bbl@tempa##1\relax % Used by LANG@bbl@ucllc
2464         \csname\language @bbl@ucllc\endcsname}%
2465       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2466     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2467     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}
2468 <<(*More package options)>> ≡
2469 \DeclareOption{nocase}{}
2470 <</More package options>>

```

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

```

2471 <<(*More package options)>> ≡
2472 \let\bbl@opt@strings\@nnil % accept strings=value
2473 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2474 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2475 \def\BabelStringsDefault{generic}
2476 <</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.

```

2477 \@onlypreamble\StartBabelCommands
2478 \def\StartBabelCommands{%
2479   \begingroup
2480   \bbl@recatcode{11}%
2481   <<Macros local to BabelCommands>>
2482   \def\bbl@provstring##1##2{%
2483     \providecommand##1{##2}%
2484     \bbl@tglobal##1}%
2485   \global\let\bbl@scafter\@empty
2486   \let\StartBabelCommands\bbl@startcmds
2487   \ifx\BabelLanguages\relax
2488     \let\BabelLanguages\CurrentOption
2489   \fi
2490   \begingroup

```

```

2491 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2492 \StartBabelCommands}
2493 \def\bbl@startcmds{%
2494   \ifx\bbl@screset\@nnil\else
2495     \bbl@usehooks{stopcommands}{}%
2496   \fi
2497 \endgroup
2498 \begingroup
2499 \@ifstar
2500   {\ifx\bbl@opt@strings\@nnil
2501     \let\bbl@opt@strings\BabelStringsDefault
2502   \fi
2503   \bbl@startcmds@i}%
2504   \bbl@startcmds@i}
2505 \def\bbl@startcmds@i#1#2{%
2506   \edef\bbl@L{\zap@space#1 \@empty}%
2507   \edef\bbl@G{\zap@space#2 \@empty}%
2508   \bbl@startcmds@ii}
2509 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

2510 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2511   \let\SetString\@gobbletwo
2512   \let\bbl@stringdef\@gobbletwo
2513   \let\AfterBabelCommands\@gobble
2514   \ifx\@empty#1%
2515     \def\bbl@sc@label{generic}%
2516     \def\bbl@encstring##1##2{%
2517       \ProvideTextCommandDefault##1{##2}%
2518       \bbl@toglobal##1%
2519       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2520     \let\bbl@sctest\in@true
2521   \else
2522     \let\bbl@sc@charset\space % <- zapped below
2523     \let\bbl@sc@fontenc\space % <- " "
2524     \def\bbl@tempa##1=##2\@nil{%
2525       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2526     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2527     \def\bbl@tempa##1 ##2{% space -> comma
2528       ##1%
2529       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2530     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2531     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2532     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2533     \def\bbl@encstring##1##2{%
2534       \bbl@foreach\bbl@sc@fontenc{%
2535         \bbl@ifunset{T#####1}%
2536         {}%
2537         {\ProvideTextCommand##1{#####1}{##2}%
2538         \bbl@toglobal##1%
2539         \expandafter

```

```

2540         \bbl@toglobal\csname####1\string##1\endcsname}}}%
2541     \def\bbl@sctest{%
2542         \bbl@xin@{\, \bbl@opt@strings,}{\, \bbl@sc@label, \bbl@sc@fontenc,}}}%
2543     \fi
2544     \ifx\bbl@opt@strings\@nnil           % ie, no strings key -> defaults
2545     \else\ifx\bbl@opt@strings\relax      % ie, strings=encoded
2546         \let\AfterBabelCommands\bbl@aftercmds
2547         \let\SetString\bbl@setstring
2548         \let\bbl@stringdef\bbl@encstring
2549     \else                                % ie, strings=value
2550     \bbl@sctest
2551     \ifin@
2552         \let\AfterBabelCommands\bbl@aftercmds
2553         \let\SetString\bbl@setstring
2554         \let\bbl@stringdef\bbl@provstring
2555     \fi\fi\fi
2556     \bbl@scswitch
2557     \ifx\bbl@G\@empty
2558         \def\SetString##1##2{%
2559             \bbl@error{Missing group for string \string##1}%
2560             {You must assign strings to some category, typically\\%
2561             captions or extras, but you set none}}}%
2562     \fi
2563     \ifx\@empty#1%
2564         \bbl@usehooks{defaultcommands}{}}}%
2565     \else
2566         \@expandtwoargs
2567         \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
2568     \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).

```

2569 \def\bbl@forlang#1#2{%
2570     \bbl@for#1\bbl@L{%
2571         \bbl@xin@{\, #1,}{\, \BabelLanguages,}%
2572         \ifin@#2\relax\fi}}
2573 \def\bbl@scswitch{%
2574     \bbl@forlang\bbl@tempa{%
2575         \ifx\bbl@G\@empty\else
2576             \ifx\SetString\@gobbletwo\else
2577                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2578                 \bbl@xin@{\, \bbl@GL,}{\, \bbl@screset,}%
2579             \ifin@\else
2580                 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2581                 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
2582             \fi
2583         \fi
2584     \fi}}
2585 \AtEndOfPackage{%
2586     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\{#2\}}}%
2587     \let\bbl@scswitch\relax}
2588 \onlypreamble\EndBabelCommands
2589 \def\EndBabelCommands{%

```

```

2590 \bbl@usehooks{stopcommands}{}%
2591 \endgroup
2592 \endgroup
2593 \bbl@scafter}
2594 \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.

```

2595 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2596 \bbl@forlang\bbl@tempa{%
2597 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2598 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2599 {\bbl@exp{%
2600 \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%
2601 }%
2602 \def\BabelString{#2}%
2603 \bbl@usehooks{stringprocess}{}%
2604 \expandafter\bbl@stringdef
2605 \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.

```

2606 \ifx\bbl@opt@strings\relax
2607 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2608 \bbl@patchuclc
2609 \let\bbl@encoded\relax
2610 \def\bbl@encoded@uclc#1{%
2611 \@inmathwarn#1%
2612 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2613 \expandafter\ifx\csname ?\string#1\endcsname\relax
2614 \TextSymbolUnavailable#1%
2615 \else
2616 \csname ?\string#1\endcsname
2617 \fi
2618 \else
2619 \csname\cf@encoding\string#1\endcsname
2620 \fi}
2621 \else
2622 \def\bbl@scset#1#2{\def#1{#2}}
2623 \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.

```

2624 <<*Macros local to BabelCommands>> ≡
2625 \def\SetStringLoop##1##2{%
2626 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2627 \count@\z@
2628 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2629 \advance\count@\@ne
2630 \toks@\expandafter{\bbl@tempa}%
2631 \bbl@exp{%
2632 \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%

```

```

2633         \count@=\the\count@\relax}}}%
2634 <</Macros local to BabelCommands>>

```

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

```

2635 \def\bbl@aftercmds#1{%
2636   \toks@\expandafter{\bbl@scafter#1}%
2637   \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.

```

2638 <<*Macros local to BabelCommands>> ≡
2639   \newcommand\SetCase[3][]{%
2640     \bbl@patchuclc
2641     \bbl@forlang\bbl@tempa{%
2642       \expandafter\bbl@encstring
2643       \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2644       \expandafter\bbl@encstring
2645       \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2646       \expandafter\bbl@encstring
2647       \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2648 <</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.

```

2649 <<*Macros local to BabelCommands>> ≡
2650   \newcommand\SetHyphenMap[1]{%
2651     \bbl@forlang\bbl@tempa{%
2652       \expandafter\bbl@stringdef
2653       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2654 <</Macros local to BabelCommands>>

```

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

```

2655 \newcommand\BabelLower[2]{% one to one.
2656   \ifnum\lccode#1=#2\else
2657     \babel@savevariable{\lccode#1}%
2658     \lccode#1=#2\relax
2659   \fi}
2660 \newcommand\BabelLowerMM[4]{% many-to-many
2661   \@tempcnta=#1\relax
2662   \@tempcntb=#4\relax
2663   \def\bbl@tempa{%
2664     \ifnum\@tempcnta>#2\else
2665       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2666       \advance\@tempcnta#3\relax
2667       \advance\@tempcntb#3\relax
2668       \expandafter\bbl@tempa
2669     \fi}%
2670   \bbl@tempa}
2671 \newcommand\BabelLowerM0[4]{% many-to-one
2672   \@tempcnta=#1\relax
2673   \def\bbl@tempa{%
2674     \ifnum\@tempcnta>#2\else
2675       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2676       \advance\@tempcnta#3
2677       \expandafter\bbl@tempa
2678     \fi}%

```


2679 \bbl@tempa}

The following package options control the behavior of hyphenation mapping.

```
2680 <<*More package options>> ≡
2681 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2682 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap@ne}
2683 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2684 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2685 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2686 <</More package options>>
```

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

```
2687 \AtEndOfPackage{%
2688   \ifx\bbl@opt@hyphenmap\undefined
2689     \bbl@xin@{,}{\bbl@language@opts}%
2690     \chardef\bbl@opt@hyphenmap\ifin4\else\ne\fi
2691   \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.

```
2692 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2693   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2694 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2695   \bbl@trim@def\bbl@tempa{#2}%
2696   \bbl@xin@{.template}{\bbl@tempa}%
2697   \ifin@
2698     \bbl@ini@captions@template{#3}{#1}%
2699   \else
2700     \edef\bbl@tempd{%
2701       \expandafter\expandafter\expandafter
2702       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2703     \bbl@xin@
2704       {\expandafter\string\csname #2name\endcsname}%
2705     {\bbl@tempd}%
2706     \ifin@ % Renew caption
2707       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2708     \ifin@
2709       \bbl@exp{%
2710         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2711         {\\bbl@scset\<#2name>\<#1#2name>}}%
2712       {}}%
2713     \else % Old way converts to new way
2714       \bbl@ifunset{#1#2name}%
2715       {\bbl@exp{%
2716         \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2717         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2718         {\def\<#2name>{\<#1#2name>}}%
2719         {}}}%
2720       {}}%
2721     \fi
2722   \else
2723     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2724     \ifin@ % New way
2725     \bbl@exp{%
2726       \\bbl@add\<captions#1>{\\bbl@scset\<#2name>\<#1#2name>}}%
2727     \\bbl@ifsamestring{\bbl@tempa}{\language}%
2728     {\\bbl@scset\<#2name>\<#1#2name>}}%
2729     {}}}
```

```

2730 \else % Old way, but defined in the new way
2731 \bbl@exp{%
2732 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2733 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2734 {\def\<#2name>{\<#1#2name>}}%
2735 }%
2736 \fi%
2737 \fi
2738 \@namedef{#1#2name}{#3}%
2739 \toks@\expandafter{\bbl@captionslist}%
2740 \bbl@exp{\in{\<#2name>}{\the\toks@}}%
2741 \ifin@else
2742 \bbl@exp{\bbl@add\\\bbl@captionslist{\<#2name>}}%
2743 \bbl@tglobal\bbl@captionslist
2744 \fi
2745 \fi}
2746 % \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.

```

2747 \bbl@trace{Macros related to glyphs}
2748 \def\set@low@box#1{\setbox\tw@{\hbox{,}}\setbox\z@{\hbox{#1}}%
2749 \dimen\z@{\ht\z@ \advance\dimen\z@ -\ht\tw@}%
2750 \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.

```

2751 \def\save@sfc@q#1{\leavevmode
2752 \begingroup
2753 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2754 \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.

```

2755 \ProvideTextCommand{\quotedblbase}{OT1}{%
2756 \save@sfc@q{\set@low@box{\textquotedblright\}}%
2757 \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.

```

2758 \ProvideTextCommandDefault{\quotedblbase}{%
2759 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2760 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2761 \save@sfc@q{\set@low@box{\textquoteright\}}%
2762 \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.

```

2763 \ProvideTextCommandDefault{\quotesinglbase}{%
2764 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2765 \ProvideTextCommand{\guillemetleft}{OT1}{%
2766   \ifmmode
2767     \ll
2768   \else
2769     \save@sf@q{\nobreak
2770       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2771   \fi}
2772 \ProvideTextCommand{\guillemetright}{OT1}{%
2773   \ifmmode
2774     \gg
2775   \else
2776     \save@sf@q{\nobreak
2777       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2778   \fi}
2779 \ProvideTextCommand{\guillemotleft}{OT1}{%
2780   \ifmmode
2781     \ll
2782   \else
2783     \save@sf@q{\nobreak
2784       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2785   \fi}
2786 \ProvideTextCommand{\guillemotright}{OT1}{%
2787   \ifmmode
2788     \gg
2789   \else
2790     \save@sf@q{\nobreak
2791       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2792   \fi}

```

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

```

2793 \ProvideTextCommandDefault{\guillemetleft}{%
2794   \UseTextSymbol{OT1}{\guillemetleft}}
2795 \ProvideTextCommandDefault{\guillemetright}{%
2796   \UseTextSymbol{OT1}{\guillemetright}}
2797 \ProvideTextCommandDefault{\guillemotleft}{%
2798   \UseTextSymbol{OT1}{\guillemotleft}}
2799 \ProvideTextCommandDefault{\guillemotright}{%
2800   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2801 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2802   \ifmmode
2803     <%
2804   \else
2805     \save@sf@q{\nobreak
2806       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2807   \fi}
2808 \ProvideTextCommand{\guilsinglright}{OT1}{%
2809   \ifmmode
2810     >%
2811   \else
2812     \save@sf@q{\nobreak
2813       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2814   \fi}

```

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

```

2815 \ProvideTextCommandDefault{\guilsinglleft}{%
2816 \UseTextSymbol{OT1}{\guilsinglleft}}
2817 \ProvideTextCommandDefault{\guilsinglright}{%
2818 \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.

```

2819 \DeclareTextCommand{\ij}{OT1}{%
2820 i\kern-0.02em\bbl@allowhyphens j}
2821 \DeclareTextCommand{\IJ}{OT1}{%
2822 I\kern-0.02em\bbl@allowhyphens J}
2823 \DeclareTextCommand{\ij}{T1}{\char188}
2824 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2825 \ProvideTextCommandDefault{\ij}{%
2826 \UseTextSymbol{OT1}{\ij}}
2827 \ProvideTextCommandDefault{\IJ}{%
2828 \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).

```

2829 \def\crrtic@{\hrule height0.1ex width0.3em}
2830 \def\crttic@{\hrule height0.1ex width0.33em}
2831 \def\ddj@{%
2832 \setbox0\hbox{d}\dimen@=\ht0
2833 \advance\dimen@1ex
2834 \dimen@.45\dimen@
2835 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2836 \advance\dimen@ii.5ex
2837 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2838 \def\DDJ@{%
2839 \setbox0\hbox{D}\dimen@=.55\ht0
2840 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2841 \advance\dimen@ii.15ex % correction for the dash position
2842 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2843 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2844 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2845 %
2846 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2847 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2848 \ProvideTextCommandDefault{\dj}{%
2849 \UseTextSymbol{OT1}{\dj}}
2850 \ProvideTextCommandDefault{\DJ}{%
2851 \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.

```

2852 \DeclareTextCommand{\SS}{OT1}{\SS}
2853 \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 2854 \ProvideTextCommandDefault{\glq}{%
2855 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2856 \ProvideTextCommand{\grq}{T1}{%
2857 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2858 \ProvideTextCommand{\grq}{TU}{%
2859 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2860 \ProvideTextCommand{\grq}{OT1}{%
2861 \save@sf@q{\kern-.0125em
2862 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2863 \kern.07em\relax}}
2864 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2865 \ProvideTextCommandDefault{\glqq}{%
2866 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2867 \ProvideTextCommand{\grqq}{T1}{%
2868 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2869 \ProvideTextCommand{\grqq}{TU}{%
2870 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2871 \ProvideTextCommand{\grqq}{OT1}{%
2872 \save@sf@q{\kern-.07em
2873 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2874 \kern.07em\relax}}
2875 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2876 \ProvideTextCommandDefault{\flq}{%
2877 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2878 \ProvideTextCommandDefault{\frq}{%
2879 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2880 \ProvideTextCommandDefault{\flqq}{%
2881 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2882 \ProvideTextCommandDefault{\frqq}{%
2883 \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).

```
2884 \def\umlauthigh{%
```

```

2885 \def\bbl@umlauta##1{\leavevmode\bgroup%
2886 \expandafter\accent\csname\fontencoding dqpos\endcsname
2887 ##1\bbl@allowhyphens\egroup}%
2888 \let\bbl@umlaute\bbl@umlauta}
2889 \def\umlautlow{%
2890 \def\bbl@umlauta{\protect\lower@umlaut}}
2891 \def\umlautelow{%
2892 \def\bbl@umlaute{\protect\lower@umlaut}}
2893 \umlauthigh

```

`\lower@umlaut` The command `\lower@umlaut` is used to position the `\` closer to the letter. We want the umlaut character lowered, nearer to the letter. To do this we need an extra *⟨dimen⟩* register.

```

2894 \expandafter\ifx\csname U@D\endcsname\relax
2895 \csname newdimen\endcsname\U@D
2896 \fi

```

The following code fools \TeX 's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the `METAFONT` parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

2897 \def\lower@umlaut#1{%
2898 \leavevmode\bgroup
2899 \U@D 1ex%
2900 {\setbox\z@\hbox{%
2901 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2902 \dimen@ -.45ex\advance\dimen@ht\z@
2903 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2904 \expandafter\accent\csname\fontencoding dqpos\endcsname
2905 \fontdimen5\font\U@D #1%
2906 \egroup}

```

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

```

2907 \AtBeginDocument{%
2908 \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2909 \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2910 \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2911 \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2912 \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2913 \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2914 \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2915 \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2916 \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2917 \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2918 \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty `\language` is defined. Currently used in `Amharic`.

```

2919 \ifx\l@english\undefined
2920 \chardef\l@english\z@
2921 \fi

```

```

2922 % The following is used to cancel rules in ini files (see Amharic).
2923 \ifx\l@unhyphenated\undefined
2924   \newlanguage\l@unhyphenated
2925 \fi

```

9.13 Layout

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

```

2926 \bbl@trace{Bidi layout}
2927 \providecommand\IfBabelLayout[3]{#3}%
2928 \newcommand\BabelPatchSection[1]{%
2929   \@ifundefined{#1}{}{%
2930     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2931     \@namedef{#1}{%
2932       \@ifstar{\bbl@presec@s{#1}}{%
2933         {\@dblarg{\bbl@presec@x{#1}}}}}%
2934 \def\bbl@presec@x#1[#2]#3{%
2935   \bbl@exp{%
2936     \\select@language@x{\bbl@main@language}%
2937     \\bbl@cs{sspre@#1}%
2938     \\bbl@cs{ss@#1}%
2939     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2940     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2941     \\select@language@x{\language}}}%
2942 \def\bbl@presec@s#1#2{%
2943   \bbl@exp{%
2944     \\select@language@x{\bbl@main@language}%
2945     \\bbl@cs{sspre@#1}%
2946     \\bbl@cs{ss@#1}*%
2947     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2948     \\select@language@x{\language}}}%
2949 \IfBabelLayout{sectioning}%
2950   {\BabelPatchSection{part}%
2951    \BabelPatchSection{chapter}%
2952    \BabelPatchSection{section}%
2953    \BabelPatchSection{subsection}%
2954    \BabelPatchSection{subsubsection}%
2955    \BabelPatchSection{paragraph}%
2956    \BabelPatchSection{subparagraph}%
2957    \def\babel@toc#1{%
2958      \select@language@x{\bbl@main@language}}}%
2959 \IfBabelLayout{captions}%
2960   {\BabelPatchSection{caption}}}%

```

9.14 Load engine specific macros

```

2961 \bbl@trace{Input engine specific macros}
2962 \ifcase\bbl@engine
2963   \input txtbabel.def
2964 \or
2965   \input luababel.def
2966 \or
2967   \input xebabel.def
2968 \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.

```
2969 \bbl@trace{Creating languages and reading ini files}
2970 \let\bbl@extend@ini@gobble
2971 \newcommand\babelprovide[2][{}%
2972   \let\bbl@savelangname\language
2973   \edef\bbl@savelocaleid{\the\localeid}%
2974   % Set name and locale id
2975   \edef\language{#2}%
2976   \bbl@id@assign
2977   % Initialize keys
2978   \let\bbl@KVP@captions\@nil
2979   \let\bbl@KVP@date\@nil
2980   \let\bbl@KVP@import\@nil
2981   \let\bbl@KVP@main\@nil
2982   \let\bbl@KVP@script\@nil
2983   \let\bbl@KVP@language\@nil
2984   \let\bbl@KVP@hyphenrules\@nil
2985   \let\bbl@KVP@linebreaking\@nil
2986   \let\bbl@KVP@justification\@nil
2987   \let\bbl@KVP@mapfont\@nil
2988   \let\bbl@KVP@maparabic\@nil
2989   \let\bbl@KVP@mapdigits\@nil
2990   \let\bbl@KVP@intraspace\@nil
2991   \let\bbl@KVP@intrapenalty\@nil
2992   \let\bbl@KVP@onchar\@nil
2993   \let\bbl@KVP@transforms\@nil
2994   \global\let\bbl@release@transforms\@empty
2995   \let\bbl@KVP@alph\@nil
2996   \let\bbl@KVP@Alph\@nil
2997   \let\bbl@KVP@labels\@nil
2998   \bbl@csarg\let{KVP@labels*}\@nil
2999   \global\let\bbl@inidata\@empty
3000   \global\let\bbl@extend@ini@gobble
3001   \gdef\bbl@key@list{}%
3002   \bbl@forkv{#1}{% TODO - error handling
3003     \in@{/{}}{##1}%
3004     \ifin@
3005       \global\let\bbl@extend@ini\bbl@extend@ini@aux
3006       \bbl@renewinikey##1\@{##2}%
3007     \else
3008       \bbl@csarg\def{KVP@##1}{##2}%
3009     \fi}%
3010   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
3011   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
3012   % == init ==
3013   \ifx\bbl@screset\@undefined
3014     \bbl@ldfinit
3015   \fi
3016   % ==
3017   \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3018   \ifcase\bbl@howloaded
3019     \let\bbl@lbkflag\@empty % new
3020   \else
3021     \ifx\bbl@KVP@hyphenrules\@nil\else
3022       \let\bbl@lbkflag\@empty
3023     \fi
3024     \ifx\bbl@KVP@import\@nil\else
3025       \let\bbl@lbkflag\@empty
```



```

3026 \fi
3027 \fi
3028 % == import, captions ==
3029 \ifx\bb1@KVP@import\@nil\else
3030 \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
3031 {\ifx\bb1@initoload\relax
3032 \begingroup
3033 \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3034 \bb1@input@texini{##2}%
3035 \endgroup
3036 \else
3037 \xdef\bb1@KVP@import{\bb1@initoload}%
3038 \fi}%
3039 {}%
3040 \fi
3041 \ifx\bb1@KVP@captions\@nil
3042 \let\bb1@KVP@captions\bb1@KVP@import
3043 \fi
3044 % ==
3045 \ifx\bb1@KVP@transforms\@nil\else
3046 \bb1@replace\bb1@KVP@transforms{ },}%
3047 \fi
3048 % == Load ini ==
3049 \ifcase\bb1@howloaded
3050 \bb1@provide@new{##2}%
3051 \else
3052 \bb1@ifblank{##1}%
3053 {}% With \bb1@load@basic below
3054 {\bb1@provide@renew{##2}}%
3055 \fi
3056 % Post tasks
3057 % -----
3058 % == subsequent calls after the first provide for a locale ==
3059 \ifx\bb1@inidata\@empty\else
3060 \bb1@extend@ini{##2}%
3061 \fi
3062 % == ensure captions ==
3063 \ifx\bb1@KVP@captions\@nil\else
3064 \bb1@ifunset{\bb1@extracaps@##2}%
3065 {\bb1@exp{\bb1@babelensure[exclude=\\today]{##2}}}%
3066 {\toks@ \expandafter \expandafter \expandafter
3067 {\csname \bb1@extracaps@##2\endcsname}%
3068 \bb1@exp{\bb1@babelensure[exclude=\\today,include=\\the\toks@]{##2}}}%
3069 \bb1@ifunset{\bb1@ensure@\\language}%
3070 {\bb1@exp{%
3071 \\\DeclareRobustCommand\<\bb1@ensure@\\language>[1]{%
3072 \\\foreignlanguage{\\language}%
3073 {####1}}}%
3074 {}%
3075 \bb1@exp{%
3076 \\\bb1@tglobal\<\bb1@ensure@\\language>%
3077 \\\bb1@tglobal\<\bb1@ensure@\\language\space>}%
3078 \fi
3079 % ==
3080 % At this point all parameters are defined if 'import'. Now we
3081 % execute some code depending on them. But what about if nothing was
3082 % imported? We just set the basic parameters, but still loading the
3083 % whole ini file.
3084 \bb1@load@basic{##2}%

```

```

3085 % == script, language ==
3086 % Override the values from ini or defines them
3087 \ifx\bb1@KVP@script\@nil\else
3088   \bb1@csarg\edef\sname@#2{\bb1@KVP@script}%
3089 \fi
3090 \ifx\bb1@KVP@language\@nil\else
3091   \bb1@csarg\edef\lname@#2{\bb1@KVP@language}%
3092 \fi
3093 % == onchar ==
3094 \ifx\bb1@KVP@onchar\@nil\else
3095   \bb1@luahyphenate
3096   \directlua{
3097     if Babel.locale_mapped == nil then
3098       Babel.locale_mapped = true
3099       Babel.linebreaking.add_before(Babel.locale_map)
3100       Babel.loc_to_scr = {}
3101       Babel.chr_to_loc = Babel.chr_to_loc or {}
3102     end}%
3103   \bb1@xin@{ ids }{ \bb1@KVP@onchar\space}%
3104   \ifin@
3105     \ifx\bb1@starthyphens\@undefined % Needed if no explicit selection
3106       \AddBabelHook{babel-onchar}{beforestart}{\bb1@starthyphens}%
3107     \fi
3108     \bb1@exp{\bb1@add\bb1@starthyphens
3109       {\bb1@patterns@lua{\language}}}%
3110     % TODO - error/warning if no script
3111     \directlua{
3112       if Babel.script_blocks['\bb1@cl{sbc}'] then
3113         Babel.loc_to_scr[\the\localeid] =
3114           Babel.script_blocks['\bb1@cl{sbc}']
3115         Babel.locale_props[\the\localeid].lc = \the\localeid\space
3116         Babel.locale_props[\the\localeid].lg = \the\@nameuse{1\language}\space
3117       end
3118     }%
3119   \fi
3120   \bb1@xin@{ fonts }{ \bb1@KVP@onchar\space}%
3121   \ifin@
3122     \bb1@ifunset{bb1@lsys\language}{\bb1@provide@lsys\language}{}%
3123     \bb1@ifunset{bb1@wdir\language}{\bb1@provide@dirs\language}{}%
3124     \directlua{
3125       if Babel.script_blocks['\bb1@cl{sbc}'] then
3126         Babel.loc_to_scr[\the\localeid] =
3127           Babel.script_blocks['\bb1@cl{sbc}']
3128       end}%
3129   \ifx\bb1@mapselect\@undefined % TODO. almost the same as mapfont
3130     \AtBeginDocument{%
3131       \bb1@patchfont{\bb1@mapselect}%
3132       {\selectfont}%
3133     \def\bb1@mapselect{%
3134       \let\bb1@mapselect\relax
3135       \edef\bb1@prefontid{\fontid\font}}%
3136     \def\bb1@mapdir##1{%
3137       {\def\language{##1}%
3138       \let\bb1@ifrestoring\@firstoftwo % To avoid font warning
3139       \bb1@switchfont
3140       \directlua{
3141         Babel.locale_props[\the\csname bb1@id@##1\endcsname]
3142           ['\bb1@prefontid'] = \fontid\font\space}}}%
3143   \fi

```

```

3144 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3145 \fi
3146 % TODO - catch non-valid values
3147 \fi
3148 % == mapfont ==
3149 % For bidi texts, to switch the font based on direction
3150 \ifx\bbl@KVP@mapfont\@nil\else
3151 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
3152 {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
3153 mapfont. Use 'direction'.%
3154 {See the manual for details.}}}%
3155 \bbl@ifunset{\bbl@lsys{\language}}{\bbl@provide@lsys{\language}}}%
3156 \bbl@ifunset{\bbl@wdir{\language}}{\bbl@provide@dirs{\language}}}%
3157 \ifx\bbl@mapselect\@undefined % TODO. See onchar.
3158 \AtBeginDocument{%
3159 \bbl@patchfont{\bbl@mapselect}}%
3160 {\selectfont}}%
3161 \def\bbl@mapselect{%
3162 \let\bbl@mapselect\relax
3163 \edef\bbl@prefontid{\fontid\font}}%
3164 \def\bbl@mapdir##1{%
3165 {\def\language{##1}%
3166 \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3167 \bbl@switchfont
3168 \directlua{Babel.fontmap
3169 [\the\csname bbl@wdir@##1\endcsname]%
3170 [\bbl@prefontid]=\fontid\font}}}%
3171 \fi
3172 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3173 \fi
3174 % == Line breaking: intraspace, intrapenalty ==
3175 % For CJK, East Asian, Southeast Asian, if interspace in ini
3176 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3177 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3178 \fi
3179 \bbl@provide@intraspace
3180 % == Line breaking: CJK quotes ==
3181 \ifcase\bbl@engine\or
3182 \bbl@xin@{/c}{\bbl@c1{lnbrk}}}%
3183 \ifin@
3184 \bbl@ifunset{\bbl@quote@\language}}}%
3185 {\directlua{
3186 Babel.locale_props[\the\localeid].cjk_quotes = {}
3187 local cs = 'op'
3188 for c in string.utfvalues(%
3189 [[\csname bbl@quote@\language\endcsname]]) do
3190 if Babel.cjk_characters[c].c == 'qu' then
3191 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
3192 end
3193 cs = ( cs == 'op') and 'cl' or 'op'
3194 end
3195 }}%
3196 \fi
3197 \fi
3198 % == Line breaking: justification ==
3199 \ifx\bbl@KVP@justification\@nil\else
3200 \let\bbl@KVP@linebreaking\bbl@KVP@justification
3201 \fi
3202 \ifx\bbl@KVP@linebreaking\@nil\else

```

```

3203 \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3204 \ifin@
3205 \bbl@csarg\xdef
3206 {lnbrk@\language\name}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
3207 \fi
3208 \fi
3209 \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
3210 \ifin@else\bbl@xin@{/k}{/\bbl@c1{lnbrk}}\fi
3211 \ifin@\bbl@arabicjust\fi
3212 % == Line breaking: hyphenate.other.(locale|script) ==
3213 \ifx\bbl@lbkflag\empty
3214 \bbl@ifunset{\bbl@hyotl@\language\name}{}%
3215 {\bbl@csarg\bbl@replace{hyotl@\language\name}{ }{,}}%
3216 \bbl@startcommands*\language\name}%
3217 \bbl@csarg\bbl@foreach{hyotl@\language\name}{%
3218 \ifcase\bbl@engine
3219 \ifnum##1<257
3220 \SetHyphenMap{\BabelLower{##1}{##1}}%
3221 \fi
3222 \else
3223 \SetHyphenMap{\BabelLower{##1}{##1}}%
3224 \fi}%
3225 \bbl@endcommands}%
3226 \bbl@ifunset{\bbl@hyots@\language\name}{}%
3227 {\bbl@csarg\bbl@replace{hyots@\language\name}{ }{,}}%
3228 \bbl@csarg\bbl@foreach{hyots@\language\name}{%
3229 \ifcase\bbl@engine
3230 \ifnum##1<257
3231 \global\lccode##1=##1\relax
3232 \fi
3233 \else
3234 \global\lccode##1=##1\relax
3235 \fi}}%
3236 \fi
3237 % == Counters: maparabic ==
3238 % Native digits, if provided in ini (TeX level, xe and lua)
3239 \ifcase\bbl@engine\else
3240 \bbl@ifunset{\bbl@dgnat@\language\name}{}%
3241 {\expandafter\ifx\csname\bbl@dgnat@\language\name\endcsname\@empty\else
3242 \expandafter\expandafter\expandafter
3243 \bbl@setdigits\csname\bbl@dgnat@\language\name\endcsname
3244 \ifx\bbl@KVP@maparabic\@nil\else
3245 \ifx\bbl@latinarabic\@undefined
3246 \expandafter\let\expandafter\@arabic
3247 \csname\bbl@counter@\language\name\endcsname
3248 \else % ie, if layout=counters, which redefines \@arabic
3249 \expandafter\let\expandafter\bbl@latinarabic
3250 \csname\bbl@counter@\language\name\endcsname
3251 \fi
3252 \fi
3253 \fi}%
3254 \fi
3255 % == Counters: mapdigits ==
3256 % Native digits (lua level).
3257 \ifodd\bbl@engine
3258 \ifx\bbl@KVP@mapdigits\@nil\else
3259 \bbl@ifunset{\bbl@dgnat@\language\name}{}%
3260 {\RequirePackage{luatexbase}}%
3261 \bbl@activate@preotf

```

```

3262 \directlua{
3263   Babel = Babel or {}  %% -> presets in luababel
3264   Babel.digits_mapped = true
3265   Babel.digits = Babel.digits or {}
3266   Babel.digits[\the\localeid] =
3267     table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3268   if not Babel.numbers then
3269     function Babel.numbers(head)
3270       local LOCALE = Babel.attr_locale
3271       local GLYPH = node.id'glyph'
3272       local inmath = false
3273       for item in node.traverse(head) do
3274         if not inmath and item.id == GLYPH then
3275           local temp = node.get_attribute(item, LOCALE)
3276           if Babel.digits[temp] then
3277             local chr = item.char
3278             if chr > 47 and chr < 58 then
3279               item.char = Babel.digits[temp][chr-47]
3280             end
3281           end
3282           elseif item.id == node.id'math' then
3283             inmath = (item.subtype == 0)
3284           end
3285         end
3286       return head
3287     end
3288   end
3289   }}%
3290 \fi
3291 \fi
3292 % == Counters: alph, Alph ==
3293 % What if extras<lang> contains a \babel@save\@alph? It won't be
3294 % restored correctly when exiting the language, so we ignore
3295 % this change with the \bbl@alph@saved trick.
3296 \ifx\bbl@KVP@alph\@nil\else
3297   \bbl@extras@wrap{\bbl@alph@saved}%
3298   {\let\bbl@alph@saved\@alph}%
3299   {\let\@alph\bbl@alph@saved
3300    \babel@save\@alph}%
3301   \bbl@exp{%
3302     \bbl@add\<extras\language\>%
3303     \let\@alph\bbl@cntr@\bbl@KVP@alph @\language\>}}%
3304 \fi
3305 \ifx\bbl@KVP@Alph\@nil\else
3306   \bbl@extras@wrap{\bbl@Alph@saved}%
3307   {\let\bbl@Alph@saved\@Alph}%
3308   {\let\@Alph\bbl@Alph@saved
3309    \babel@save\@Alph}%
3310   \bbl@exp{%
3311     \bbl@add\<extras\language\>%
3312     \let\@Alph\bbl@cntr@\bbl@KVP@Alph @\language\>}}%
3313 \fi
3314 % == require.babel in ini ==
3315 % To load or reload the babel-*.tex, if require.babel in ini
3316 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3317   \bbl@ifunset\bbl@rtex@\language\>{}%
3318   {\expandafter\ifx\csname bbl@rtex@\language\endcsname\@empty\else
3319     \let\BabelBeforeIni@gobbletwo
3320     \chardef\atcatcode=\catcode`\@

```

```

3321      \catcode`\@=11\relax
3322      \bbl@input@texini{\bbl@cs{rqtex@\language}}%
3323      \catcode`\@=\atcatcode
3324      \let\atcatcode\relax
3325      \global\bbl@csarg\let{rqtex@\language}\relax
3326      \fi}%
3327  \fi
3328  % == frenchspacing ==
3329  \ifcase\bbl@howloaded\in@true\else\in@false\fi
3330  \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
3331  \ifin@
3332    \bbl@extras@wrap{\bbl@pre@fs}%
3333    {\bbl@pre@fs}%
3334    {\bbl@post@fs}%
3335  \fi
3336  % == Release saved transforms ==
3337  \bbl@release@transforms\relax % \relax closes the last item.
3338  % == main ==
3339  \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3340    \let\language\bbl@savelangname
3341    \chardef\localeid\bbl@savelocaleid\relax
3342  \fi}

```

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

```

3343 \def\bbl@provide@new#1{%
3344   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3345   \@namedef{extras#1}{}%
3346   \@namedef{noextras#1}{}%
3347   \bbl@startcommands*{#1}{captions}%
3348   \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3349     \def\bbl@tempb##1{% elt for \bbl@captionslist
3350       \ifx##1\@empty\else
3351         \bbl@exp{%
3352           \SetString\##1{%
3353             \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3354         \expandafter\bbl@tempb
3355       \fi}%
3356   \expandafter\bbl@tempb\bbl@captionslist\@empty
3357   \else
3358     \ifx\bbl@initoload\relax
3359       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
3360     \else
3361       \bbl@read@ini{\bbl@initoload}2% % Same
3362     \fi
3363   \fi
3364   \StartBabelCommands*{#1}{date}%
3365   \ifx\bbl@KVP@import\@nil
3366     \bbl@exp{%
3367       \SetString\today{\bbl@nocaption{today}{#1today}}}%
3368   \else
3369     \bbl@savetoday
3370     \bbl@savedate
3371   \fi
3372   \bbl@endcommands
3373   \bbl@load@basic{#1}%
3374   % == hyphenmins == (only if new)
3375   \bbl@exp{%
3376     \gdef\<#1hyphenmins>{%

```

```

3377     {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3378     {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3379 % == hyphenrules (also in renew) ==
3380 \bbl@provide@hyphens{#1}%
3381 \ifx\bbl@KVP@main\@nil\else
3382     \expandafter\main@language\expandafter{#1}%
3383 \fi}
3384 %
3385 \def\bbl@provide@renew#1{%
3386     \ifx\bbl@KVP@captions\@nil\else
3387         \StartBabelCommands*{#1}{captions}%
3388         \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
3389         \EndBabelCommands
3390     \fi
3391     \ifx\bbl@KVP@import\@nil\else
3392         \StartBabelCommands*{#1}{date}%
3393         \bbl@savetoday
3394         \bbl@savedate
3395         \EndBabelCommands
3396     \fi
3397 % == hyphenrules (also in new) ==
3398 \ifx\bbl@lbkflag\@empty
3399     \bbl@provide@hyphens{#1}%
3400 \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.)

```

3401 \def\bbl@load@basic#1{%
3402     \ifcase\bbl@howloaded\or\or
3403         \ifcase\csname bbl@llevel@\language\endcsname
3404             \bbl@csarg\let{lname@\language}\relax
3405         \fi
3406     \fi
3407     \bbl@ifunset{bbl@lname@#1}%
3408     {\def\BabelBeforeIni##1##2{%
3409         \begingroup
3410             \let\bbl@ini@captions@aux\@gobbletwo
3411             \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
3412             \bbl@read@ini{##1}1%
3413             \ifx\bbl@initoload\relax\endinput\fi
3414         \endgroup}%
3415         \begingroup    % boxed, to avoid extra spaces:
3416             \ifx\bbl@initoload\relax
3417                 \bbl@input@texini{##1}%
3418             \else
3419                 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3420             \fi
3421         \endgroup}%
3422     }}

```

The hyphenrules option is handled with an auxiliary macro.

```

3423 \def\bbl@provide@hyphens#1{%
3424     \let\bbl@tempa\relax
3425     \ifx\bbl@KVP@hyphenrules\@nil\else
3426         \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3427         \bbl@foreach\bbl@KVP@hyphenrules{%
3428             \ifx\bbl@tempa\relax    % if not yet found
3429                 \bbl@ifsamestring{##1}{+}%

```

```

3430      {\bbl@exp{\addlanguage\<l@##1>}}}%
3431      {}%
3432      \bbl@ifunset{l@##1}%
3433      {}%
3434      {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
3435      \fi}%
3436  \fi
3437  \ifx\bbl@tempa\relax %           if no opt or no language in opt found
3438      \ifx\bbl@KVP@import\@nil
3439          \ifx\bbl@initoload\relax\else
3440              \bbl@exp{%           and hyphenrules is not empty
3441                  \\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3442                  {}%
3443                  {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}}%
3444      \fi
3445      \else % if importing
3446          \bbl@exp{%           and hyphenrules is not empty
3447              \\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3448              {}%
3449              {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}}%
3450      \fi
3451  \fi
3452  \bbl@ifunset{bbl@tempa}%         ie, relax or undefined
3453  {\bbl@ifunset{l@#1}%           no hyphenrules found - fallback
3454      {\bbl@exp{\adddialect\<l@#1>\language}}}%
3455      {}}%                       so, l@<lang> is ok - nothing to do
3456  {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini

```

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

```

3457 \def\bbl@input@texini#1{%
3458     \bbl@bsphack
3459     \bbl@exp{%
3460         \catcode`\\%=14 \catcode`\\=0
3461         \catcode`\\{=1 \catcode`\\}=2
3462         \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
3463         \catcode`\\%=the\catcode`\%relax
3464         \catcode`\\=the\catcode`\relax
3465         \catcode`\\{=the\catcode`\{relax
3466         \catcode`\\}=the\catcode`\}relax}%
3467     \bbl@esphack}

```

The following macros read and store ini files (but don't process them). For each line, there are 3 possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbl@read@ini.

```

3468 \def\bbl@inline#1\bbl@inline{%
3469     \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@}% ]
3470 \def\bbl@iniset[#1]#2\@{\def\bbl@section{#1}}
3471 \def\bbl@iniskip#1\@{\%           if starts with ;
3472 \def\bbl@inistore#1=#2\@{\%       full (default)
3473     \bbl@trim@def\bbl@tempa{#1}%
3474     \bbl@trim\toks@{#2}%
3475     \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
3476     \ifin\else
3477         \bbl@exp{%
3478             \\\g@addto@macro\\bbl@inidata{%
3479                 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3480     \fi}
3481 \def\bbl@inistore@min#1=#2\@{\%   minimal (maybe set in \bbl@read@ini)
3482     \bbl@trim@def\bbl@tempa{#1}%

```



```

3483 \bbl@trim\toks@{#2}%
3484 \bbl@xin@{.identification.}{.\bbl@section.}%
3485 \ifin@
3486 \bbl@exp{\g@addto@macro\bbl@inidata{%
3487 \bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3488 \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.

```

3489 \ifx\bbl@readstream\undefined
3490 \csname newread\endcsname\bbl@readstream
3491 \fi
3492 \def\bbl@read@ini#1#2{%
3493 \global\let\bbl@extend@ini@gobble
3494 \openin\bbl@readstream=babel-#1.ini
3495 \ifeof\bbl@readstream
3496 \bbl@error
3497 {There is no ini file for the requested language\%
3498 (#1). Perhaps you misspelled it or your installation\%
3499 is not complete.}%
3500 {Fix the name or reinstall babel.}%
3501 \else
3502 % == Store ini data in \bbl@inidata ==
3503 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3504 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3505 \bbl@info{Importing
3506 \ifcase#2font and identification \or basic \fi
3507 data for \language\%
3508 from babel-#1.ini. Reported}%
3509 \ifnum#2=\z@
3510 \global\let\bbl@inidata\empty
3511 \let\bbl@inistore\bbl@inistore@min % Remember it's local
3512 \fi
3513 \def\bbl@section{identification}%
3514 \bbl@exp{\bbl@inistore tag.ini=#1\@@}%
3515 \bbl@inistore load.level=#2\@@
3516 \loop
3517 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3518 \endlinechar\m@ne
3519 \read\bbl@readstream to \bbl@line
3520 \endlinechar\^^M
3521 \ifx\bbl@line\empty\else
3522 \expandafter\bbl@inline\bbl@line\bbl@inline
3523 \fi
3524 \repeat
3525 % == Process stored data ==
3526 \bbl@csarg\xdef{lini\language}{#1}%
3527 \bbl@read@ini@aux
3528 % == 'Export' data ==
3529 \bbl@ini@exports{#2}%
3530 \global\bbl@csarg\let{inidata\language}\bbl@inidata
3531 \global\let\bbl@inidata\empty
3532 \bbl@exp{\bbl@add@list\bbl@ini@loaded\language}%
3533 \bbl@tglobal\bbl@ini@loaded
3534 \fi}

```

```

3535 \def\bbl@read@ini@aux{%
3536   \let\bbl@savestrings\@empty
3537   \let\bbl@savetoday\@empty
3538   \let\bbl@savestate\@empty
3539   \def\bbl@elt##1##2##3{%
3540     \def\bbl@section{##1}%
3541     \in@{=date.}{=##1}% Find a better place
3542     \ifin@
3543       \bbl@ini@calendar{##1}%
3544     \fi
3545     \bbl@ifunset{bbl@inikv@##1}{}%
3546     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
3547   \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.

```

3548 \def\bbl@extend@ini@aux#1{%
3549   \bbl@startcommands*{#1}{captions}%
3550   % Activate captions/... and modify exports
3551   \bbl@csarg\def{inikv@captions.licr}##1##2{%
3552     \setlocalecaption{#1}{##1}{##2}}%
3553   \def\bbl@inikv@captions##1##2{%
3554     \bbl@ini@captions@aux{##1}{##2}}%
3555   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3556   \def\bbl@exportkey##1##2##3{%
3557     \bbl@ifunset{bbl@kv@##2}{}%
3558     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
3559       \bbl@exp{\global\let<bbl@##1@<language>\<bbl@kv@##2>}%
3560       \fi}}%
3561   % As with \bbl@read@ini, but with some changes
3562   \bbl@read@ini@aux
3563   \bbl@ini@exports\tw@
3564   % Update inidata@lang by pretending the ini is read.
3565   \def\bbl@elt##1##2##3{%
3566     \def\bbl@section{##1}%
3567     \bbl@iniline##2=##3\bbl@iniline}%
3568     \csname bbl@inidata@#1\endcsname
3569     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
3570   \StartBabelCommands*{#1}{date}% And from the import stuff
3571   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3572   \bbl@savetoday
3573   \bbl@savestate
3574   \bbl@endcommands}

```

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

```

3575 \def\bbl@ini@calendar#1{%
3576   \lowercase{\def\bbl@tempa{=##1=}}%
3577   \bbl@replace\bbl@tempa{=date.gregorian}{}%
3578   \bbl@replace\bbl@tempa{=date.}{}%
3579   \in@{.licr}{#1}%
3580   \ifin@
3581     \ifcase\bbl@engine
3582       \bbl@replace\bbl@tempa{.licr=}{}%
3583     \else
3584       \let\bbl@tempa\relax
3585     \fi
3586   \fi
3587   \ifx\bbl@tempa\relax\else
3588     \bbl@replace\bbl@tempa{=}{}%

```

```

3589 \bbl@exp{%
3590 \def\<bbl@inikv@#1>###1###2{%
3591 \\\bbl@inidate###1...\relax{###2}{\bbl@tempa}}}%
3592 \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).

```

3593 \def\bbl@renewinikey#1/#2\@#3{%
3594 \edef\bbl@tempa{\zap@space #1 \@empty}% section
3595 \edef\bbl@tempb{\zap@space #2 \@empty}% key
3596 \bbl@trim\toks@{#3}% value
3597 \bbl@exp{%
3598 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
3599 \\\g@addto@macro\\bbl@inidata{%
3600 \\\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.

```

3601 \def\bbl@exportkey#1#2#3{%
3602 \bbl@ifunset{bbl@kv@#2}%
3603 {\bbl@csarg\gdef{#1@\language}\@empty}%
3604 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3605 \bbl@csarg\gdef{#1@\language}\@empty}%
3606 \else
3607 \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3608 \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.

```

3609 \def\bbl@iniwarning#1{%
3610 \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3611 {\bbl@warning{%
3612 From babel-\bbl@cs{lini@\language}.ini:\%
3613 \bbl@cs{kv@identification.warning#1}\%
3614 Reported }}}
3615 %
3616 \let\bbl@release@transforms\@empty
3617 %
3618 \def\bbl@ini@exports#1{%
3619 % Identification always exported
3620 \bbl@iniwarning}%
3621 \ifcase\bbl@engine
3622 \bbl@iniwarning{.pdflatex}%
3623 \or
3624 \bbl@iniwarning{.lualatex}%
3625 \or
3626 \bbl@iniwarning{.xelatex}%
3627 \fi%
3628 \bbl@exportkey{llevel}{identification.load.level}}}%
3629 \bbl@exportkey{elname}{identification.name.english}}}%
3630 \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3631 {\csname bbl@elname@\language\endcsname}}}%
3632 \bbl@exportkey{tbc}{identification.tag.bcp47}}}%
3633 \bbl@exportkey{lbc}{identification.language.tag.bcp47}}}%
3634 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}}}%
3635 \bbl@exportkey{esname}{identification.script.name}}}%

```

```

3636 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3637 {\csname bbl@esname@language\endcsname}}%
3638 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3639 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3640 % Also maps bcp47 -> language
3641 \ifbbl@bcptoname
3642 \bbl@csarg\edef{bcp@map@\bbl@cl{tbcpr}}{\language}%
3643 \fi
3644 % Conditional
3645 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
3646 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3647 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3648 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3649 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3650 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3651 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3652 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3653 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3654 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3655 \bbl@exportkey{chrng}{characters.ranges}{}%
3656 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3657 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3658 \ifnum#1=\tw@ % only (re)new
3659 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3660 \bbl@tglobal\bbl@savetoday
3661 \bbl@tglobal\bbl@savestate
3662 \bbl@savestrings
3663 \fi
3664 \fi}

```

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

```

3665 \def\bbl@inikv#1#2{% key=value
3666 \toks@{#2}% This hides #'s from ini values
3667 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3668 \let\bbl@inikv@identification\bbl@inikv
3669 \let\bbl@inikv@typography\bbl@inikv
3670 \let\bbl@inikv@characters\bbl@inikv
3671 \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’.

```

3672 \def\bbl@inikv@counters#1#2{%
3673 \bbl@ifsamestring{#1}{digits}%
3674 {\bbl@error{The counter name 'digits' is reserved for mapping\%
3675 decimal digits}%
3676 {Use another name.}}%
3677 }%
3678 \def\bbl@tempc{#1}%
3679 \bbl@trim@def{\bbl@tempb*}{#2}%
3680 \in@{.1$}{#1$}%
3681 \ifin@
3682 \bbl@replace\bbl@tempc{.1}{}%
3683 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
3684 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3685 \fi
3686 \in@{.F.}{#1}%

```

```

3687 \ifin@else\in@{.S.}{#1}\fi
3688 \ifin@
3689 \bbl@csarg\protected@xdef{cntr@#1@\language}\bbl@tempb*}%
3690 \else
3691 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3692 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3693 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3694 \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.

```

3695 \ifcase\bbl@engine
3696 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3697 \bbl@ini@captions@aux{#1}{#2}}
3698 \else
3699 \def\bbl@inikv@captions#1#2{%
3700 \bbl@ini@captions@aux{#1}{#2}}
3701 \fi

```

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

```

3702 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3703 \bbl@replace\bbl@tempa{.template}{}}%
3704 \def\bbl@toreplace{#1}{}%
3705 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3706 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3707 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3708 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3709 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3710 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3711 \ifin@
3712 \@nameuse{\bbl@patch\bbl@tempa}%
3713 \global\bbl@csarg\let{\bbl@tempa fmt#2}\bbl@toreplace
3714 \fi
3715 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3716 \ifin@
3717 \toks@\expandafter{\bbl@toreplace}%
3718 \bbl@exp{\gdef\<fnun@\bbl@tempa>{\the\toks@}}%
3719 \fi}
3720 \def\bbl@ini@captions@aux#1#2{%
3721 \bbl@trim@def\bbl@tempa{#1}%
3722 \bbl@xin@{.template}{\bbl@tempa}%
3723 \ifin@
3724 \bbl@ini@captions@template{#2}\language
3725 \else
3726 \bbl@ifblank{#2}%
3727 {\bbl@exp{%
3728 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3729 {\bbl@trim\toks@{#2}}}%
3730 \bbl@exp{%
3731 \bbl@add{\bbl@savestrings{%
3732 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3733 \toks@\expandafter{\bbl@captionslist}%
3734 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}}%
3735 \ifin@else
3736 \bbl@exp{%
3737 \bbl@add{\bbl@extracaps@\language>{\<\bbl@tempa name>}}%
3738 \bbl@toglobal{\bbl@extracaps@\language>}}%
3739 \fi

```

```
3740 \fi}
```

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

```
3741 \def\bbl@list@the{%
3742   part,chapter,section,subsection,subsubsection,paragraph,%
3743   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3744   table,page,footnote,mpfootnote,mpfn}
3745 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3746   \bbl@ifunset{bbl@map@#1@\languagename}%
3747     {\@nameuse{#1}}%
3748     {\@nameuse{bbl@map@#1@\languagename}}}
3749 \def\bbl@inikv@labels#1#2{%
3750   \in@{.map}{#1}%
3751   \ifin@
3752     \ifx\bbl@KVP@labels\@nil\else
3753       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3754       \ifin@
3755         \def\bbl@tempc{#1}%
3756         \bbl@replace\bbl@tempc{.map}{}%
3757         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3758         \bbl@exp{%
3759           \gdef\bbl@map@\bbl@tempc @\languagename>%
3760           {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3761         \bbl@foreach\bbl@list@the{%
3762           \bbl@ifunset{the##1}{}%
3763           {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3764             \bbl@exp{%
3765               \\bbl@sreplace\<the##1>%
3766               {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3767             \\bbl@sreplace\<the##1>%
3768             {\<\@empty @\bbl@tempc>\<c##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3769             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3770               \toks@ \expandafter\expandafter\expandafter{%
3771                 \csname the##1\endcsname}%
3772               \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3773             \fi}}%
3774         \fi
3775       \fi
3776       %
3777     \else
3778       %
3779       % The following code is still under study. You can test it and make
3780       % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3781       % language dependent.
3782       \in@{enumerate.}{#1}%
3783       \ifin@
3784         \def\bbl@tempa{#1}%
3785         \bbl@replace\bbl@tempa{enumerate.}{}%
3786         \def\bbl@toreplace{#2}%
3787         \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3788         \bbl@replace\bbl@toreplace{[]}{\csname the}%
3789         \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3790         \toks@ \expandafter{\bbl@toreplace}%
3791         % TODO. Execute only once:
3792         \bbl@exp{%
3793           \\bbl@add\<extras\languagename>{%
3794             \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3795             \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3796           \\bbl@together\<extras\languagename>}%
3797       \fi
3798     \fi
3799   }
```

```

3797 \fi
3798 \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.

```

3799 \def\bbl@chapttype{chapter}
3800 \ifx\@makechapterhead\undefined
3801 \let\bbl@patchchapter\relax
3802 \else\ifx\thechapter\undefined
3803 \let\bbl@patchchapter\relax
3804 \else\ifx\ps@headings\undefined
3805 \let\bbl@patchchapter\relax
3806 \else
3807 \def\bbl@patchchapter{%
3808 \global\let\bbl@patchchapter\relax
3809 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3810 \bbl@tglobal\appendix
3811 \bbl@sreplace\ps@headings
3812 {\@chapapp\ thechapter}%
3813 {\bbl@chapterformat}%
3814 \bbl@tglobal\ps@headings
3815 \bbl@sreplace\chaptermark
3816 {\@chapapp\ thechapter}%
3817 {\bbl@chapterformat}%
3818 \bbl@tglobal\chaptermark
3819 \bbl@sreplace\@makechapterhead
3820 {\@chapapp\space\thechapter}%
3821 {\bbl@chapterformat}%
3822 \bbl@tglobal\@makechapterhead
3823 \gdef\bbl@chapterformat{%
3824 \bbl@ifunset{\bbl\bbl@chapttype fmt@\language}%
3825 {\@chapapp\space\thechapter}
3826 {\@nameuse{\bbl\bbl@chapttype fmt@\language}}}}
3827 \let\bbl@patchappendix\bbl@patchchapter
3828 \fi\fi\fi
3829 \ifx\@part\undefined
3830 \let\bbl@patchpart\relax
3831 \else
3832 \def\bbl@patchpart{%
3833 \global\let\bbl@patchpart\relax
3834 \bbl@sreplace\@part
3835 {\partname\nobreakspace\thepart}%
3836 {\bbl@partformat}%
3837 \bbl@tglobal\@part
3838 \gdef\bbl@partformat{%
3839 \bbl@ifunset{\bbl@partfmt@\language}%
3840 {\partname\nobreakspace\thepart}
3841 {\@nameuse{\bbl@partfmt@\language}}}}
3842 \fi

```

Date. TODO. Document

```

3843 % Arguments are _not_ protected.
3844 \let\bbl@calendar\@empty
3845 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3846 \def\bbl@localedate#1#2#3#4{%
3847 \begingroup
3848 \ifx\@empty#1\@empty\else

```

```

3849 \let\bbl@ld@calendar\@empty
3850 \let\bbl@ld@variant\@empty
3851 \edef\bbl@tempa{\zap@space#1 \@empty}%
3852 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3853 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3854 \edef\bbl@calendar{%
3855   \bbl@ld@calendar
3856   \ifx\bbl@ld@variant\@empty\else
3857     .\bbl@ld@variant
3858   \fi}%
3859 \bbl@replace\bbl@calendar{gregorian}{}%
3860 \fi
3861 \bbl@cased
3862 {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3863 \endgroup}
3864 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3865 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3866   \bbl@trim@def\bbl@tempa{#1.#2}%
3867   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3868   {\bbl@trim@def\bbl@tempa{#3}%
3869     \bbl@trim\toks@{#5}%
3870     \@temptokena\expandafter{\bbl@savedate}%
3871     \bbl@exp{% Reverse order - in ini last wins
3872       \def\\bbl@savedate{%
3873         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3874         \the\@temptokena}}}%
3875   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3876     {\lowercase{\def\bbl@tempb{#6}}%
3877       \bbl@trim@def\bbl@toreplace{#5}%
3878       \bbl@TG@@date
3879       \bbl@ifunset{\bbl@date@\language @}%
3880       {\bbl@exp{% TODO. Move to a better place.
3881         \gdef\<\language date>{\\protect\<\language date >}%
3882         \gdef\<\language date >####1####2####3{%
3883           \\bbl@usedategroupttrue
3884           \<\bbl@ensure@\language >{%
3885             \\localedate{####1}{####2}{####3}}}%
3886           \\bbl@add\\bbl@savetoday{%
3887             \\SetString\\today{%
3888               \<\language date>%
3889               {\the\year}{\the\month}{\the\day}}}}}%
3890       }%
3891       \global\bbl@csarg\let{date@\language @}\bbl@toreplace
3892       \ifx\bbl@tempb\@empty\else
3893         \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3894       \fi}%
3895     {}}}}

```

Dates will require some macros for the basic formatting. They may be redefined by language, so “semi-public” names (camel case) are used. Oddly enough, the CLDR places particles like “de” inconsistently in either in the date or in the month name. Note after `\bbl@replace \toks@` contains the resulting string, which is used by `\bbl@replace@finish@iii` (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3896 \let\bbl@calendar\@empty
3897 \newcommand\BabelDateSpace{\nobreakspace}
3898 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3899 \newcommand\BabelDated[1]{\number#1}
3900 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3901 \newcommand\BabelDateM[1]{\number#1}

```



```

3902 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3903 \newcommand\BabelDateMMMM[1]{\ifnum#1<10 0\fi\number#1}
3904 \csname month\romannumeral#1\bb1@calendar name\endcsname}%
3905 \newcommand\BabelDatey[1]{\number#1}%
3906 \newcommand\BabelDateyy[1]{\ifnum#1<10 0\number#1 %
3907 \else\ifnum#1<100 \number#1 %
3908 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3909 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3910 \else
3911 \bb1@error
3912 {Currently two-digit years are restricted to the\
3913 range 0-9999.}%
3914 {There is little you can do. Sorry.}%
3915 \fi\fi\fi\fi}%
3916 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3917 \def\bb1@replace@finish@iii#1{%
3918 \bb1@exp{\def\#1####1####2####3{\the\toks@}}%
3919 \def\bb1@TG@date{%
3920 \bb1@replace\bb1@toreplace{[ ]}{\BabelDateSpace{}}%
3921 \bb1@replace\bb1@toreplace{[. ]}{\BabelDateDot{}}%
3922 \bb1@replace\bb1@toreplace{[d]}{\BabelDated{####3}}%
3923 \bb1@replace\bb1@toreplace{[dd]}{\BabelDatedd{####3}}%
3924 \bb1@replace\bb1@toreplace{[M]}{\BabelDateM{####2}}%
3925 \bb1@replace\bb1@toreplace{[MM]}{\BabelDateMM{####2}}%
3926 \bb1@replace\bb1@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3927 \bb1@replace\bb1@toreplace{[y]}{\BabelDatey{####1}}%
3928 \bb1@replace\bb1@toreplace{[yy]}{\BabelDateyy{####1}}%
3929 \bb1@replace\bb1@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3930 \bb1@replace\bb1@toreplace{[y]}{\bb1@datecctr[####1]}%
3931 \bb1@replace\bb1@toreplace{[m]}{\bb1@datecctr[####2]}%
3932 \bb1@replace\bb1@toreplace{[d]}{\bb1@datecctr[####3]}%
3933 \bb1@replace@finish@iii\bb1@toreplace}
3934 \def\bb1@datecctr{\expandafter\bb1@xdatecctr\expandafter}
3935 \def\bb1@xdatecctr[#1|#2]{\localnumeral{#2}{#1}}

```

Transforms.

```

3937 \let\bb1@release@transforms\@empty
3938 \@namedef{bb1@inikv@transforms.prehyphenation}{%
3939 \bb1@transforms\babelprehyphenation}
3940 \@namedef{bb1@inikv@transforms.posthyphenation}{%
3941 \bb1@transforms\babelposthyphenation}
3942 \def\bb1@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3943 \begingroup
3944 \catcode`\%=12
3945 \catcode`\&=14
3946 \gdef\bb1@transforms#1#2#3{&%
3947 \ifx\bb1@KVP@transforms\@nil\else
3948 \directlua{
3949 str = [=[#2]=]
3950 str = str:gsub('%.%d+%.%d+$', '')
3951 tex.print([[ \def\string\babeltempa{]] .. str .. [[]])
3952 }&%
3953 \bb1@xin@{, \babeltempa,}{, \bb1@KVP@transforms,}&%
3954 \ifin@
3955 \in@{.0$}{#2$}&%
3956 \ifin@
3957 \g@addto@macro\bb1@release@transforms{&%
3958 \relax\bb1@transforms@aux#1{\language name}{#3}}&%

```

```

3959         \else
3960         \g@addto@macro\bbbl@release@transforms{, {#3}}&%
3961         \fi
3962     \fi
3963 \fi}
3964 \endgroup

```

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

```

3965 \def\bbbl@provide@lsys#1{%
3966     \bbbl@ifunset{bbbl@lname@#1}%
3967     {\bbbl@load@info{#1}}%
3968     }%
3969     \bbbl@csarg\let{lsys@#1}\@empty
3970     \bbbl@ifunset{bbbl@sname@#1}{\bbbl@csarg\gdef{sname@#1}{Default}}{}%
3971     \bbbl@ifunset{bbbl@sotf@#1}{\bbbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3972     \bbbl@csarg\bbbl@add@list{lsys@#1}{Script=\bbbl@cs{sname@#1}}%
3973     \bbbl@ifunset{bbbl@lname@#1}{}%
3974     {\bbbl@csarg\bbbl@add@list{lsys@#1}{Language=\bbbl@cs{lname@#1}}}%
3975     \ifcase\bbbl@engine\or\or
3976     \bbbl@ifunset{bbbl@prehc@#1}{}%
3977     {\bbbl@exp{\bbbl@ifblank{\bbbl@cs{prehc@#1}}}%
3978     }%
3979     {\ifx\bbbl@xenoHyph\@undefined
3980     \let\bbbl@xenoHyph\bbbl@xenoHyph@d
3981     \ifx\AtBeginDocument\@notprerr
3982     \expandafter\@secondoftwo % to execute right now
3983     \fi
3984     \AtBeginDocument{%
3985         \bbbl@patchfont{\bbbl@xenoHyph}%
3986         \expandafter\selectlanguage\expandafter{\language}%
3987     \fi}}%
3988     \fi
3989     \bbbl@csarg\bbbl@toGlobal{lsys@#1}}
3990 \def\bbbl@xenoHyph@d{%
3991     \bbbl@ifset{bbbl@prehc@\language}%
3992     {\ifnum\hyphenchar\font=\defaultHyphenChar
3993     \iffontchar\font\bbbl@cl{prehc}\relax
3994     \hyphenchar\font\bbbl@cl{prehc}\relax
3995     \else\iffontchar\font"200B
3996     \hyphenchar\font"200B
3997     \else
3998     \bbbl@warning
3999     {Neither 0 nor ZERO WIDTH SPACE are available\\%
4000     in the current font, and therefore the hyphen\\%
4001     will be printed. Try changing the fontspec's\\%
4002     'HyphenChar' to another value, but be aware\\%
4003     this setting is not safe (see the manual)}%
4004     \hyphenchar\font\defaultHyphenChar
4005     \fi\fi
4006     \fi}%
4007     {\hyphenchar\font\defaultHyphenChar}}
4008     % \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).

```

4009 \def\bbbl@load@info#1{%

```



```

4057 \def\bbbl@localecntr#1#2{\localenumeral{#2}{#1}}
4058 \newcommand\localecounter[2]{%
4059   \expandafter\bbbl@localecntr
4060   \expandafter{\number\csname c@#2\endcsname}{#1}}
4061 \def\bbbl@alphnumeral#1#2{%
4062   \expandafter\bbbl@alphnumeral@i\number#2 76543210\@@{#1}}
4063 \def\bbbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
4064   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
4065     \bbbl@alphnumeral@ii{#9}000000#1\or
4066     \bbbl@alphnumeral@ii{#9}00000#1#2\or
4067     \bbbl@alphnumeral@ii{#9}0000#1#2#3\or
4068     \bbbl@alphnumeral@ii{#9}000#1#2#3#4\else
4069     \bbbl@alphnum@invalid{>9999}%
4070   \fi}
4071 \def\bbbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
4072   \bbbl@ifunset{bbbl@cntr@#1.F.\number#5#6#7#8@\language}%
4073   {\bbbl@cs{cntr@#1.4@\language}#5%
4074     \bbbl@cs{cntr@#1.3@\language}#6%
4075     \bbbl@cs{cntr@#1.2@\language}#7%
4076     \bbbl@cs{cntr@#1.1@\language}#8%
4077     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4078       \bbbl@ifunset{bbbl@cntr@#1.S.321@\language}{}%
4079       {\bbbl@cs{cntr@#1.S.321@\language}}%
4080     \fi}%
4081   {\bbbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
4082 \def\bbbl@alphnum@invalid#1{%
4083   \bbbl@error{Alphabetic numeral too large (#1)}%
4084   {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.

```

4085 \newcommand\localeinfo[1]{%
4086   \bbbl@ifunset{bbbl\csname bbl@info@#1\endcsname @\language}%
4087   {\bbbl@error{I've found no info for the current locale.\%
4088     The corresponding ini file has not been loaded\%
4089     Perhaps it doesn't exist}%
4090     {See the manual for details.}}%
4091   {\bbbl@cs{\csname bbl@info@#1\endcsname @\language}}%
4092 % \@namedef{bbbl@info@name.locale}{lcname}
4093 \@namedef{bbbl@info@tag.ini}{lini}
4094 \@namedef{bbbl@info@name.english}{elname}
4095 \@namedef{bbbl@info@name.opentype}{lname}
4096 \@namedef{bbbl@info@tag.bcp47}{tbcpl}
4097 \@namedef{bbbl@info@language.tag.bcp47}{lbcpl}
4098 \@namedef{bbbl@info@tag.opentype}{lotf}
4099 \@namedef{bbbl@info@script.name}{esname}
4100 \@namedef{bbbl@info@script.name.opentype}{sname}
4101 \@namedef{bbbl@info@script.tag.bcp47}{sbcp}
4102 \@namedef{bbbl@info@script.tag.opentype}{sotf}
4103 \let\bbbl@ensureinfo\@gobble
4104 \newcommand\BabelEnsureInfo{%
4105   \ifx\InputIfFileExists\@undefined\else
4106     \def\bbbl@ensureinfo##1{%
4107       \bbbl@ifunset{bbbl@lname@##1}{\bbbl@load@info{##1}}{}}%
4108   \fi
4109   \bbbl@foreach\bbbl@loaded{%
4110     \def\language{##1}%
4111     \bbbl@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.

4112 \newcommand\getlocaleproperty{%
4113   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4114 \def\bbl@getproperty@s#1#2#3{%
4115   \let#1\relax
4116   \def\bbl@elt##1##2##3{%
4117     \bbl@ifsamestring{##1/##2}{#3}%
4118     {\providecommand#1{##3}%
4119     \def\bbl@elt####1####2####3{}}}%
4120   {}}%
4121   \bbl@cs{inidata@#2}}%
4122 \def\bbl@getproperty@x#1#2#3{%
4123   \bbl@getproperty@s{#1}{#2}{#3}%
4124   \ifx#1\relax
4125     \bbl@error
4126     {Unknown key for locale '#2':\%
4127     #3\%
4128     \string#1 will be set to \relax}%
4129     {Perhaps you misspelled it.}%
4130   \fi}
4131 \let\bbl@ini@loaded\@empty
4132 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel bahavior

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

```

4133 \newcommand\babeladjust[1]{% TODO. Error handling.
4134   \bbl@forkv{#1}{%
4135     \bbl@ifunset{\bbl@ADJ@##1@##2}%
4136     {\bbl@cs{ADJ@##1}{##2}}%
4137     {\bbl@cs{ADJ@##1@##2}}}
4138 %
4139 \def\bbl@adjust@lua#1#2{%
4140   \ifvmode
4141     \ifnum\currentgrouplevel=\z@
4142       \directlua{ Babel.#2 }%
4143       \expandafter\expandafter\expandafter\@gobble
4144     \fi
4145   \fi
4146   {\bbl@error % The error is gobbled if everything went ok.
4147     {Currently, #1 related features can be adjusted only\%
4148     in the main vertical list.}%
4149     {Maybe things change in the future, but this is what it is.}}}
4150 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
4151   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4152 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
4153   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4154 \@namedef{\bbl@ADJ@bidi.text@on}{%
4155   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4156 \@namedef{\bbl@ADJ@bidi.text@off}{%
4157   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4158 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
4159   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4160 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
4161   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4162 %

```

```

4163 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4164   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4165 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4166   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4167 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4168   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4169 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4170   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4171 \@namedef{bbl@ADJ@justify.arabic@on}{%
4172   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4173 \@namedef{bbl@ADJ@justify.arabic@off}{%
4174   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4175 %
4176 \def\bbl@adjust@layout#1{%
4177   \ifvmode
4178     #1%
4179     \expandafter\@gobble
4180   \fi
4181   {\bbl@error   % The error is gobbled if everything went ok.
4182    {Currently, layout related features can be adjusted only\\%
4183     in vertical mode.}%
4184    {Maybe things change in the future, but this is what it is.}}}
4185 \@namedef{bbl@ADJ@layout.tabular@on}{%
4186   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4187 \@namedef{bbl@ADJ@layout.tabular@off}{%
4188   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4189 \@namedef{bbl@ADJ@layout.lists@on}{%
4190   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4191 \@namedef{bbl@ADJ@layout.lists@off}{%
4192   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4193 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4194   \bbl@activateposthyphen}
4195 %
4196 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4197   \bbl@bcpallowedtrue}
4198 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4199   \bbl@bcpallowedfalse}
4200 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
4201   \def\bbl@bcp@prefix{#1}}
4202 \def\bbl@bcp@prefix{bcp47-}
4203 \@namedef{bbl@ADJ@autoload.options#1}{%
4204   \def\bbl@autoload@options{#1}}
4205 \let\bbl@autoload@bcptoptions\@empty
4206 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
4207   \def\bbl@autoload@bcptoptions{#1}}
4208 \newif\ifbbl@bcptoname
4209 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4210   \bbl@bcptonametrue}
4211 \BabelEnsureInfo{
4212 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4213   \bbl@bcptonamefalse}
4214 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4215   \directlua{ Babel.ignore_pre_char = function(node)
4216     return (node.lang == \the\csname l@nohyphenation\endcsname)
4217   end }}
4218 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4219   \directlua{ Babel.ignore_pre_char = function(node)
4220     return false
4221   end }}

```

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

```
4222 \ifx\directlua\@undefined\else
4223   \ifx\bbl@luapatterns\@undefined
4224     \input luababel.def
4225   \fi
4226 \fi
4227 \</core>
```

A proxy file for switch.def

```
4228 <*kernel>
4229 \let\bbl@onlyswitch\@empty
4230 \input babel.def
4231 \let\bbl@onlyswitch\@undefined
4232 \</kernel>
4233 <*patterns>
```

11 Loading hyphenation patterns

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

```
4234 <<Make sure ProvidesFile is defined>>
4235 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
4236 \xdef\bbl@format{\jobname}
4237 \def\bbl@version{<<version>>}
4238 \def\bbl@date{<<date>>}
4239 \ifx\AtBeginDocument\@undefined
4240   \def\@empty{}
4241 \fi
4242 <<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.

```
4243 \def\process@line#1#2 #3 #4 {%
4244   \ifx=#1%
4245     \process@synonym{#2}%
4246   \else
4247     \process@language{#1#2}{#3}{#4}%
4248   \fi
4249   \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.

```
4250 \toks@{}
4251 \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.

```
4252 \def\process@synonym#1{%
4253   \ifnum\last@language=\m@ne
4254     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4255   \else
4256     \expandafter\chardef\curname 1@#1\endcurname\last@language
```

```

4257 \wlog{\string\l@#1=\string\language\the\last@language}%
4258 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4259 \csname\language\language\hyphenmins\endcsname
4260 \let\bbl@elt\relax
4261 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
4262 \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` and `\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.

```

4263 \def\process@language#1#2#3{%
4264 \expandafter\addlanguage\csname l@#1\endcsname
4265 \expandafter\language\csname l@#1\endcsname
4266 \edef\language{#1}%
4267 \bbl@hook@everylanguage{#1}%
4268 % > luatex
4269 \bbl@get@enc#1::@@@
4270 \begingroup
4271 \lefthyphenmin\m@ne
4272 \bbl@hook@loadpatterns{#2}%
4273 % > luatex
4274 \ifnum\lefthyphenmin=\m@ne
4275 \else
4276 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4277 \the\lefthyphenmin\the\righthyphenmin}%
4278 \fi
4279 \endgroup
4280 \def\bbl@tempa{#3}%
4281 \ifx\bbl@tempa\@empty\else
4282 \bbl@hook@loadexceptions{#3}%
4283 % > luatex
4284 \fi
4285 \let\bbl@elt\relax
4286 \edef\bbl@languages{%
4287 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%

```



```

4288 \ifnum\the\language=\z@
4289 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4290 \set@hyphenmins\tw@\thr@\relax
4291 \else
4292 \expandafter\expandafter\expandafter\set@hyphenmins
4293 \csname #1hyphenmins\endcsname
4294 \fi
4295 \the\toks@
4296 \toks@{}%
4297 \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.

```

4298 \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.

```

4299 \def\bbl@hook@everylanguage#1{}
4300 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4301 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4302 \def\bbl@hook@loadkernel#1{%
4303 \def\addlanguage{\csname newlanguage\endcsname}%
4304 \def\adddialect##1##2{%
4305 \global\chardef##1##2\relax
4306 \wlog{\string##1 = a dialect from \string\language##2}}%
4307 \def\iflanguage##1{%
4308 \expandafter\ifx\csname l@##1\endcsname\relax
4309 \nol@nerr{##1}%
4310 \else
4311 \ifnum\csname l@##1\endcsname=\language
4312 \expandafter\expandafter\expandafter\@firstoftwo
4313 \else
4314 \expandafter\expandafter\expandafter\@secondoftwo
4315 \fi
4316 \fi}%
4317 \def\providehyphenmins##1##2{%
4318 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4319 \@namedef{##1hyphenmins}{##2}%
4320 \fi}%
4321 \def\set@hyphenmins##1##2{%
4322 \lefthyphenmin##1\relax
4323 \righthyphenmin##2\relax}%
4324 \def\selectlanguage{%
4325 \errhelp{Selecting a language requires a package supporting it}%
4326 \errmessage{Not loaded}}%
4327 \let\foreignlanguage\selectlanguage
4328 \let\otherlanguage\selectlanguage
4329 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4330 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4331 \def\setlocale{%
4332 \errhelp{Find an armchair, sit down and wait}%
4333 \errmessage{Not yet available}}%
4334 \let\uselocale\setlocale
4335 \let\locale\setlocale
4336 \let\selectlocale\setlocale
4337 \let\localename\setlocale
4338 \let\textlocale\setlocale

```

```

4339 \let\textlanguage\setlocale
4340 \let\languagetext\setlocale}
4341 \begingroup
4342 \def\AddBabelHook#1#2{%
4343   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4344     \def\next{\toks1}%
4345   \else
4346     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4347   \fi
4348   \next}
4349 \ifx\directlua\@undefined
4350   \ifx\XeTeXinputencoding\@undefined\else
4351     \input xebabel.def
4352   \fi
4353 \else
4354   \input luababel.def
4355 \fi
4356 \openin1 = babel-\bbl@format.cfg
4357 \ifeof1
4358 \else
4359   \input babel-\bbl@format.cfg\relax
4360 \fi
4361 \closein1
4362 \endgroup
4363 \bbl@hook@loadkernel{switch.def}

```

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

```

4364 \openin1 = language.dat

```

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

```

4365 \def\language{english}%
4366 \ifeof1
4367   \message{I couldn't find the file language.dat,\space
4368           I will try the file hyphen.tex}
4369   \input hyphen.tex\relax
4370   \chardef\l@english\z@
4371 \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 .

```

4372 \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.

```

4373 \loop
4374   \endlinechar\m@ne
4375   \read1 to \bbl@line
4376   \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.

```

4377   \if T\ifeof1F\fi T\relax
4378   \ifx\bbl@line\@empty\else
4379     \edef\bbl@line{\bbl@line\space\space\space}%

```

```

4380      \expandafter\process@line\bbl@line\relax
4381      \fi
4382  \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.

```

4383  \begingroup
4384      \def\bbl@elt#1#2#3#4{%
4385          \global\language=#2\relax
4386          \gdef\language#1}%
4387      \def\bbl@elt##1##2##3##4{}}%
4388      \bbl@languages
4389  \endgroup
4390  \fi
4391  \closein1

```

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

```

4392  \if\the\toks@\else
4393      \errhelp{language.dat loads no language, only synonyms}
4394      \errmessage{Orphan language synonym}
4395  \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.

```

4396  \let\bbl@line\@undefined
4397  \let\process@line\@undefined
4398  \let\process@synonym\@undefined
4399  \let\process@language\@undefined
4400  \let\bbl@get@enc\@undefined
4401  \let\bbl@hyph@enc\@undefined
4402  \let\bbl@tempa\@undefined
4403  \let\bbl@hook@loadkernel\@undefined
4404  \let\bbl@hook@everylanguage\@undefined
4405  \let\bbl@hook@loadpatterns\@undefined
4406  \let\bbl@hook@loadexceptions\@undefined
4407  \</patterns>

```

Here the code for iniTeX ends.

12 Font handling with fontspec

Add the bidi handler just before luaoftload, 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].

```

4408 << *More package options >> ≡
4409 \chardef\bbl@bidimode\z@
4410 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4411 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4412 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4413 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4414 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4415 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4416 <</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.

```

4417 <<*Font selection>> ≡
4418 \bbl@trace{Font handling with fontspec}
4419 \ifx\ExplSyntaxOn\@undefined\else
4420   \ExplSyntaxOn
4421   \catcode`\ =10
4422   \def\bbl@loadfontspec{%
4423     \usepackage{fontspec}% TODO. Apply patch always
4424     \expandafter
4425     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4426       Font '\l_fontspec_fontname_tl' is using the\\%
4427       default features for language '##1'.\\%
4428       That's usually fine, because many languages\\%
4429       require no specific features, but if the output is\\%
4430       not as expected, consider selecting another font.}
4431     \expandafter
4432     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4433       Font '\l_fontspec_fontname_tl' is using the\\%
4434       default features for script '##2'.\\%
4435       That's not always wrong, but if the output is\\%
4436       not as expected, consider selecting another font.}}
4437   \ExplSyntaxOff
4438 \fi
4439 \@onlypreamble\babelfont
4440 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4441   \bbl@foreach{#1}{%
4442     \expandafter\ifx\csname date##1\endcsname\relax
4443       \IfFileExists{babel-##1.tex}%
4444       {\babelprovide{##1}}%
4445       {}%
4446     \fi}%
4447   \edef\bbl@tempa{#1}%
4448   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4449   \ifx\fontspec\@undefined
4450     \bbl@loadfontspec
4451   \fi
4452   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4453   \bbl@bblfont}
4454 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4455   \bbl@ifunset{\bbl@tempb family}%
4456   {\bbl@providefam{\bbl@tempb}}%
4457   {}%
4458   % For the default font, just in case:
4459   \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4460   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4461   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4462     \bbl@exp{%
4463       \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4464       \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4465       \<bbl@tempb default>\<bbl@tempb family>}}%
4466   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4467     \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:

```

4468 \def\bbl@providefam#1{%
4469   \bbl@exp{%
4470     \\\newcommand\<#1default>{}% Just define it

```

```

4471 \\\bbl@add@list\\\bbl@font@fams{#1}%
4472 \\\DeclareRobustCommand\<#1family>{%
4473   \\\not@math@alphabet\<#1family>\relax
4474   % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4475   \\\fontfamily\<#1default>%
4476   \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4477   \\\selectfont}%
4478   \\\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.

```

4479 \def\bbl@nostdfont#1{%
4480   \bbl@ifunset{bbl@WFF@f@family}%
4481   {\bbl@csarg\gdef{WFF@f@family}}}% Flag, to avoid dupl warns
4482   \bbl@infowarn{The current font is not a babel standard family:\\%
4483     #1%
4484     \fontname\font\\%
4485     There is nothing intrinsically wrong with this warning, and\\%
4486     you can ignore it altogether if you do not need these\\%
4487     families. But if they are used in the document, you should be\\%
4488     aware 'babel' will no set Script and Language for them, so\\%
4489     you may consider defining a new family with \string\babelfont.\\%
4490     See the manual for further details about \string\babelfont.\\%
4491     Reported}}
4492   {}}%
4493 \gdef\bbl@switchfont{%
4494   \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}}%
4495   \bbl@exp{% eg Arabic -> arabic
4496     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4497   \bbl@foreach\bbl@font@fams{%
4498     \bbl@ifunset{bbl@##1dflt@language}% (1) language?
4499     {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4500       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4501         {}}% 123=F - nothing!
4502         {\bbl@exp{% 3=T - from generic
4503           \global\let\<bbl@##1dflt@language>%
4504             \<bbl@##1dflt@>}}}%
4505           {\bbl@exp{% 2=T - from script
4506             \global\let\<bbl@##1dflt@language>%
4507               \<bbl@##1dflt@*\bbl@tempa>}}}%
4508           {}}% 1=T - language, already defined
4509   \def\bbl@tempa{\bbl@nostdfont}}}%
4510   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4511     \bbl@ifunset{bbl@##1dflt@language}%
4512     {\bbl@cs{famrst@##1}%
4513       \global\bbl@csarg\let{famrst@##1}\relax}%
4514     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4515       \\\bbl@add\\\originalTeX{%
4516         \\\bbl@font@rst{\bbl@cl{##1dflt}}}%
4517         \<##1default>\<##1family>{##1}}}%
4518       \\\bbl@font@set\<bbl@##1dflt@language>% the main part!
4519       \<##1default>\<##1family>}}}%
4520   \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.

```

4521 \ifx\fontfamily\@undefined\else % if latex
4522 \ifcase\bbl@engine % if pdftex
4523   \let\bbl@ckeckstdfonts\relax

```

```

4524 \else
4525   \def\bbbl@ckeckstdfonts{%
4526     \begingroup
4527     \global\let\bbbl@ckeckstdfonts\relax
4528     \let\bbbl@tempa\@empty
4529     \bbbl@foreach\bbbl@font@fams{%
4530       \bbbl@ifunset{\bbbl@##1dflt@}%
4531       {\nameuse{##1family}%
4532        \bbbl@csarg\gdef{WFF@\f@family}{}% Flag
4533        \bbbl@exp{\bbbl@add\bbbl@tempa{* \<##1family>= \f@family\\}%
4534         \space\space\fontname\font\\}%
4535        \bbbl@csarg\xdef{##1dflt@}{\f@family}%
4536        \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4537       {}}%
4538     \ifx\bbbl@tempa\@empty\else
4539       \bbbl@infowarn{The following font families will use the default\\%
4540        settings for all or some languages:\\%
4541        \bbbl@tempa
4542        There is nothing intrinsically wrong with it, but\\%
4543        'babel' will no set Script and Language, which could\\%
4544        be relevant in some languages. If your document uses\\%
4545        these families, consider redefining them with \string\babelfont.\\%
4546        Reported}%
4547     \fi
4548   \endgroup}
4549 \fi
4550 \fi

```

Now the macros defining the font with fontspec.

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

```

4551 \def\bbbl@font@set#1#2#3{% eg \bbbl@rmdflt@lang \rmdefault \rmfamily
4552   \bbbl@xin@{<>}{#1}%
4553   \ifin@
4554     \bbbl@exp{\bbbl@fontspec@set\#1\expandafter\@gobbletwo\#1\#3}%
4555   \fi
4556   \bbbl@exp{%
4557     \def\#2{#1}% eg, \rmdefault{\bbbl@rmdflt@lang}
4558     \bbbl@ifsamestring{#2}{\f@family}%
4559     {\#3%
4560      \bbbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}%
4561     \let\bbbl@tempa\relax}%
4562   {}}
4563 % TODO - next should be global?, but even local does its job. I'm
4564 % still not sure -- must investigate:
4565 \def\bbbl@fontspec@set#1#2#3#4{% eg \bbbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4566   \let\bbbl@tempa\bbbl@mapselect
4567   \let\bbbl@mapselect\relax
4568   \let\bbbl@tempa@fam#4% eg, '\rmfamily', to be restored below
4569   \let#4\@empty % Make sure \renewfontfamily is valid
4570   \bbbl@exp{%
4571     \let\bbbl@tempa@pfam\<\bbbl@stripslash#4\space>% eg, '\rmfamily '
4572     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbbl@cl{sname}}%
4573     {\newfontscript{\bbbl@cl{sname}}{\bbbl@cl{sotf}}}%
4574     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbbl@cl{lname}}%
4575     {\newfontlanguage{\bbbl@cl{lname}}{\bbbl@cl{lotf}}}%
4576     \renewfontfamily\#4%
4577     [\bbbl@cs{lsys@\language@name},#2]{#3}% ie \bbbl@exp{.}{#3}

```

```

4578 \begingroup
4579   #4%
4580   \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4581 \endgroup
4582 \let#4\bbl@temp@fam
4583 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4584 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4585 \def\bbl@font@rst#1#2#3#4{%
4586   \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.

```

4587 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4588 \newcommand\babelFSstore[2][{%
4589   \bbl@ifblank{#1}%
4590   {\bbl@csarg\def{sname@#2}{Latin}}%
4591   {\bbl@csarg\def{sname@#2}{#1}}%
4592   \bbl@provide@dirs{#2}%
4593   \bbl@csarg\ifnum{wdir@#2}>\z@
4594     \let\bbl@beforeforeign\leavevmode
4595     \EnableBabelHook{babel-bidi}%
4596   \fi
4597   \bbl@foreach{#2}{%
4598     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4599     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4600     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4601 \def\bbl@FSstore#1#2#3#4{%
4602   \bbl@csarg\edef{#2default#1}{#3}%
4603   \expandafter\addto\csname extras#1\endcsname{%
4604     \let#4#3%
4605     \ifx#3\f@family
4606       \edef#3{\csname bbl@#2default#1\endcsname}%
4607       \fontfamily{#3}\selectfont
4608     \else
4609       \edef#3{\csname bbl@#2default#1\endcsname}%
4610       \fi}%
4611   \expandafter\addto\csname noextras#1\endcsname{%
4612     \ifx#3\f@family
4613       \fontfamily{#4}\selectfont
4614       \fi
4615     \let#3#4}}
4616 \let\bbl@langfeatures\@empty
4617 \def\babelFSfeatures{% make sure \fontspec is redefined once
4618   \let\bbl@ori@fontspec\fontspec
4619   \renewcommand\fontspec[1][{%
4620     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4621   \let\babelFSfeatures\bbl@FSfeatures
4622   \babelFSfeatures}
4623 \def\bbl@FSfeatures#1#2{%
4624   \expandafter\addto\csname extras#1\endcsname{%
4625     \babel@save\bbl@langfeatures
4626     \edef\bbl@langfeatures{#2,}}
4627 <</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.

```
4628 <<*Footnote changes>> ≡
4629 \bbl@trace{Bidi footnotes}
4630 \ifnum\bbl@bidimode>\z@
4631   \def\bbl@footnote#1#2#3{%
4632     \@ifnextchar[%
4633       {\bbl@footnote@o{#1}{#2}{#3}}%
4634       {\bbl@footnote@x{#1}{#2}{#3}}}
4635   \long\def\bbl@footnote@x#1#2#3#4{%
4636     \bgroup
4637       \select@language@x{\bbl@main@language}%
4638       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4639     \egroup}
4640   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4641     \bgroup
4642       \select@language@x{\bbl@main@language}%
4643       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4644     \egroup}
4645   \def\bbl@footnotetext#1#2#3{%
4646     \@ifnextchar[%
4647       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4648       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4649   \long\def\bbl@footnotetext@x#1#2#3#4{%
4650     \bgroup
4651       \select@language@x{\bbl@main@language}%
4652       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4653     \egroup}
4654   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4655     \bgroup
4656       \select@language@x{\bbl@main@language}%
4657       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4658     \egroup}
4659   \def\BabelFootnote#1#2#3#4{%
4660     \ifx\bbl@fn@footnote\undefined
4661       \let\bbl@fn@footnote\footnote
4662     \fi
4663     \ifx\bbl@fn@footnotetext\undefined
4664       \let\bbl@fn@footnotetext\footnotetext
4665     \fi
4666     \bbl@ifblank{#2}%
4667       {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4668        \@namedef{\bbl@stripslash#1text}%
4669         {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4670       {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4671        \@namedef{\bbl@stripslash#1text}%
4672         {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4673   \fi
4674 <</Footnote changes>>
```

Now, the code.

```
4675 <*xetex>
4676 \def\BabelStringsDefault{unicode}
4677 \let\xebbl@stop\relax
4678 \AddBabelHook{xetex}{encodedcommands}{%
```



```

4679 \def\bbl@tempa{#1}%
4680 \ifx\bbl@tempa\@empty
4681 \XeTeXinputencoding"bytes"%
4682 \else
4683 \XeTeXinputencoding"#1"%
4684 \fi
4685 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4686 \AddBabelHook{xetex}{stopcommands}{%
4687 \xebbl@stop
4688 \let\xebbl@stop\relax}
4689 \def\bbl@intraspace#1 #2 #3\@@{%
4690 \bbl@csarg\gdef{xeisp@\language}%
4691 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4692 \def\bbl@intrapenalty#1\@@{%
4693 \bbl@csarg\gdef{xeipn@\language}%
4694 {\XeTeXlinebreakpenalty #1\relax}}
4695 \def\bbl@provide@intraspace{%
4696 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4697 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4698 \ifin@
4699 \bbl@ifunset{bbl@intsp@\language}{}%
4700 {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4701 \ifx\bbl@KVP@intraspace\@nil
4702 \bbl@exp{%
4703 \\\bbl@intraspace\bbl@cl{intsp}\\\@}%
4704 \fi
4705 \ifx\bbl@KVP@intrapenalty\@nil
4706 \bbl@intrapenalty0\@@
4707 \fi
4708 \fi
4709 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4710 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4711 \fi
4712 \ifx\bbl@KVP@intrapenalty\@nil\else
4713 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4714 \fi
4715 \bbl@exp{%
4716 % TODO. Execute only once (but redundant):
4717 \\\bbl@add\<extras\language>{%
4718 \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4719 \<bbl@xeisp@\language>%
4720 \<bbl@xeipn@\language>%
4721 \\\bbl@toglobal\<extras\language>%
4722 \\\bbl@add\<noextras\language>{%
4723 \XeTeXlinebreaklocale "en"%
4724 \\\bbl@toglobal\<noextras\language>}%
4725 \ifx\bbl@ispace\@undefined
4726 \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4727 \ifx\AtBeginDocument\@notprerr
4728 \expandafter\@secondoftwo % to execute right now
4729 \fi
4730 \AtBeginDocument{\bbl@patchfont{\bbl@xeno-hyph}}%
4731 \fi}%
4732 \fi}
4733 \ifx\DisableBabelHook\@undefined\endinput\fi
4734 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4735 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
4736 \DisableBabelHook{babel-fontspec}
4737 <<Font selection>>

```

```
4738 \input txtbabel.def
4739 </xetex>
```

13.2 Layout

In progress.

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

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

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

```
4740 (*texxet)
4741 \providecommand\bbl@provide@intraspace{}
4742 \bbl@trace{Redefinitions for bidi layout}
4743 \def\bbl@sspre@caption{%
4744   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4745 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4746 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4747 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4748 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4749   \def\@hangfrom#1{%
4750     \setbox\@tempboxa\hbox{{#1}}}%
4751     \hangindent\ifcase\bbl@thepardirwd\@tempboxa\else-\wd\@tempboxa\fi
4752     \noindent\box\@tempboxa}
4753 \def\raggedright{%
4754   \let\@centercr
4755   \bbl@startskip\z@skip
4756   \@rightskip\@flushglue
4757   \bbl@endskip\@rightskip
4758   \parindent\z@
4759   \parfillskip\bbl@startskip}
4760 \def\raggedleft{%
4761   \let\@centercr
4762   \bbl@startskip\@flushglue
4763   \bbl@endskip\z@skip
4764   \parindent\z@
4765   \parfillskip\bbl@endskip}
4766 \fi
4767 \IfBabelLayout{lists}
4768   {\bbl@sreplace\list
4769     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4770     \def\bbl@listleftmargin{%
4771       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4772     \ifcase\bbl@engine
4773       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4774       \def\p@enumiii{\p@enumii}\theenumii{}\fi
4775     \bbl@sreplace\@verbatim
4776       {\leftskip\@totalleftmargin}%
4777       {\bbl@startskip\textwidth
4778         \advance\bbl@startskip-\linewidth}%
4779     \bbl@sreplace\@verbatim
4780       {\rightskip\z@skip}%
4781       {\bbl@endskip\z@skip}}%
4782   {}
4783 \IfBabelLayout{contents}
4784   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4785     {\bbl@startskip\z@skip}}
```

```

4786 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4787 {}
4788 \IfBabelLayout{columns}%
4789 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4790 \def\bbl@outputbox#1{%
4791 \hb@xt@\textwidth{%
4792 \hskip\columnwidth
4793 \hfil
4794 {\normalcolor\vrule \@width\columnseprule}%
4795 \hfil
4796 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4797 \hskip-\textwidth
4798 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4799 \hskip\columnsep
4800 \hskip\columnwidth}}}%
4801 {}
4802 <<Footnote changes>>
4803 \IfBabelLayout{footnotes}%
4804 {\BabelFootnote\footnote\language\language{}{}}%
4805 \BabelFootnote\localfootnote\language\language{}{}}%
4806 \BabelFootnote\mainfootnote{}{}}{}%
4807 {}

```

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.

```

4808 \IfBabelLayout{counters}%
4809 {\let\bbl@latinarabic=\@arabic
4810 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4811 \let\bbl@asciroman=\@roman
4812 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4813 \let\bbl@asciiRoman=\@Roman
4814 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
4815 </texxet>

```

13.3 LuaTeX

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

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

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

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

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

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

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

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg. \babelpatterns).

```

4816 (*luatex)
4817 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4818 \bbl@trace{Read language.dat}
4819 \ifx\bbl@readstream\undefined
4820 \csname newread\endcsname\bbl@readstream
4821 \fi
4822 \begingroup
4823 \toks@{}
4824 \count@ \z@ % 0=start, 1=0th, 2=normal
4825 \def\bbl@process@line#1#2 #3 #4 {%
4826   \ifx=#1%
4827     \bbl@process@synonym{#2}%
4828   \else
4829     \bbl@process@language{#1#2}{#3}{#4}%
4830   \fi
4831   \ignorespaces}
4832 \def\bbl@manylang{%
4833   \ifnum\bbl@last>\@ne
4834     \bbl@info{Non-standard hyphenation setup}%
4835   \fi
4836   \let\bbl@manylang\relax}
4837 \def\bbl@process@language#1#2#3{%
4838   \ifcase\count@
4839     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4840   \or
4841     \count@\tw@
4842   \fi
4843   \ifnum\count@=\tw@
4844     \expandafter\addlanguage\csname l@#1\endcsname
4845     \language\allocationnumber
4846     \chardef\bbl@last\allocationnumber
4847     \bbl@manylang
4848     \let\bbl@elt\relax
4849     \xdef\bbl@languages{%
4850       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4851   \fi
4852   \the\toks@
4853   \toks@{}}
4854 \def\bbl@process@synonym@aux#1#2{%
4855   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4856   \let\bbl@elt\relax
4857   \xdef\bbl@languages{%
4858     \bbl@languages\bbl@elt{#1}{#2}{\the\language}}%
4859 \def\bbl@process@synonym#1{%
4860   \ifcase\count@
4861     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4862   \or
4863     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%

```

```

4864 \else
4865 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4866 \fi}
4867 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4868 \chardef\l@english\z@
4869 \chardef\l@USenglish\z@
4870 \chardef\bbl@last\z@
4871 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
4872 \gdef\bbl@languages{%
4873 \bbl@elt{english}{0}{\hyphen.tex}{}%
4874 \bbl@elt{USenglish}{0}{}{}}
4875 \else
4876 \global\let\bbl@languages@format\bbl@languages
4877 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4878 \ifnum#2>\z@\else
4879 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4880 \fi}%
4881 \xdef\bbl@languages{\bbl@languages}%
4882 \fi
4883 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{} } % Define flags
4884 \bbl@languages
4885 \openin\bbl@readstream=language.dat
4886 \ifeof\bbl@readstream
4887 \bbl@warning{I couldn't find language.dat. No additional\%
4888 patterns loaded. Reported}%
4889 \else
4890 \loop
4891 \endlinechar\m@ne
4892 \read\bbl@readstream to \bbl@line
4893 \endlinechar`\^^M
4894 \if T\ifeof\bbl@readstream F\fi T\relax
4895 \ifx\bbl@line\empty\else
4896 \edef\bbl@line{\bbl@line\space\space\space}%
4897 \expandafter\bbl@process@line\bbl@line\relax
4898 \fi
4899 \repeat
4900 \fi
4901 \endgroup
4902 \bbl@trace{Macros for reading patterns files}
4903 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4904 \ifx\babelcatcodetablenum\undefined
4905 \ifx\newcatcodetable\undefined
4906 \def\babelcatcodetablenum{5211}
4907 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4908 \else
4909 \newcatcodetable\babelcatcodetablenum
4910 \newcatcodetable\bbl@pattcodes
4911 \fi
4912 \else
4913 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4914 \fi
4915 \def\bbl@luapatterns#1#2{%
4916 \bbl@get@enc#1::\@@
4917 \setbox\z@\hbox\bgroup
4918 \begingroup
4919 \savecatcodetable\babelcatcodetablenum\relax
4920 \initcatcodetable\bbl@pattcodes\relax
4921 \catcodetable\bbl@pattcodes\relax
4922 \catcode`\#6 \catcode`\$3 \catcode`\&4 \catcode`\^7

```

```

4923 \catcode`\_ =8 \catcode`\{ =1 \catcode`\} =2 \catcode`\- =13
4924 \catcode`\@ =11 \catcode`\^^I =10 \catcode`\^^J =12
4925 \catcode`\< =12 \catcode`\> =12 \catcode`\* =12 \catcode`\.=12
4926 \catcode`\- =12 \catcode`\ / =12 \catcode`\[ =12 \catcode`\] =12
4927 \catcode`\' =12 \catcode`\' =12 \catcode`\ " =12
4928 \input #1\relax
4929 \catcodetable\babelcatcodetablenum\relax
4930 \endgroup
4931 \def\bbl@tempa{#2}%
4932 \ifx\bbl@tempa@empty\else
4933 \input #2\relax
4934 \fi
4935 \egroup}%
4936 \def\bbl@patterns@lua#1{%
4937 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4938 \csname l@#1\endcsname
4939 \edef\bbl@tempa{#1}%
4940 \else
4941 \csname l@#1:\f@encoding\endcsname
4942 \edef\bbl@tempa{#1:\f@encoding}%
4943 \fi\relax
4944 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4945 \@ifundefined{bbl@hyphendata@the\language}%
4946 {\def\bbl@elt##1##2##3##4{%
4947 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4948 \def\bbl@tempb{##3}%
4949 \ifx\bbl@tempb@empty\else % if not a synonymous
4950 \def\bbl@tempc{##3}{##4}%
4951 \fi
4952 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4953 \fi}%
4954 \bbl@languages
4955 \@ifundefined{bbl@hyphendata@the\language}%
4956 {\bbl@info{No hyphenation patterns were set for\%
4957 language '\bbl@tempa'. Reported}}%
4958 {\expandafter\expandafter\expandafter\bbl@luapatterns
4959 \csname bbl@hyphendata@the\language\endcsname}}}%
4960 \endinput\fi
4961 % Here ends \ifx\AddBabelHook\undefined
4962 % A few lines are only read by hyphen.cfg
4963 \ifx\DisableBabelHook\undefined
4964 \AddBabelHook{luatex}{everylanguage}{%
4965 \def\process@language##1##2##3{%
4966 \def\process@line####1####2 ####3 ####4 {}}%
4967 \AddBabelHook{luatex}{loadpatterns}{%
4968 \input #1\relax
4969 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4970 {{#1}}}%
4971 \AddBabelHook{luatex}{loadexceptions}{%
4972 \input #1\relax
4973 \def\bbl@tempb##1##2{{##1}{##2}}%
4974 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4975 {\expandafter\expandafter\expandafter\bbl@tempb
4976 \csname bbl@hyphendata@the\language\endcsname}}%
4977 \endinput\fi
4978 % Here stops reading code for hyphen.cfg
4979 % The following is read the 2nd time it's loaded
4980 \begingroup % TODO - to a lua file
4981 \catcode`\% =12

```

```

4982 \catcode`\'=12
4983 \catcode`\ "=12
4984 \catcode`\:=12
4985 \directlua{
4986   Babel = Babel or {}
4987   function Babel.bytes(line)
4988     return line:gsub(".",
4989       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4990   end
4991   function Babel.begin_process_input()
4992     if luatexbase and luatexbase.add_to_callback then
4993       luatexbase.add_to_callback('process_input_buffer',
4994         Babel.bytes, 'Babel.bytes')
4995     else
4996       Babel.callback = callback.find('process_input_buffer')
4997       callback.register('process_input_buffer', Babel.bytes)
4998     end
4999   end
5000   function Babel.end_process_input ()
5001     if luatexbase and luatexbase.remove_from_callback then
5002       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5003     else
5004       callback.register('process_input_buffer', Babel.callback)
5005     end
5006   end
5007   function Babel.addpatterns(pp, lg)
5008     local lg = lang.new(lg)
5009     local pats = lang.patterns(lg) or ''
5010     lang.clear_patterns(lg)
5011     for p in pp:gmatch('[^%s]+') do
5012       ss = ''
5013       for i in string.utfcharacters(p:gsub('%d', '')) do
5014         ss = ss .. '%d?' .. i
5015       end
5016       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5017       ss = ss:gsub('%%.%%d%?$', '%%.')
5018       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5019       if n == 0 then
5020         tex.sprint(
5021           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5022           .. p .. [[]])
5023         pats = pats .. ' ' .. p
5024       else
5025         tex.sprint(
5026           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5027           .. p .. [[]])
5028       end
5029     end
5030     lang.patterns(lg, pats)
5031   end
5032 }
5033 \endgroup
5034 \ifx\newattribute\@undefined\else
5035   \newattribute\bbl@attr@locale
5036   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5037   \AddBabelHook{luatex}{beforeextras}{%
5038     \setattribute\bbl@attr@locale\localeid}
5039 \fi
5040 \def\BabelStringsDefault{unicode}

```

```

5041 \let\luabbl@stop\relax
5042 \AddBabelHook{luatex}{encodedcommands}{%
5043   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5044   \ifx\bbl@tempa\bbl@tempb\else
5045     \directlua{Babel.begin_process_input()}%
5046     \def\luabbl@stop{%
5047       \directlua{Babel.end_process_input()}}%
5048   \fi}%
5049 \AddBabelHook{luatex}{stopcommands}{%
5050   \luabbl@stop
5051   \let\luabbl@stop\relax}
5052 \AddBabelHook{luatex}{patterns}{%
5053   \@ifundefined{bbl@hyphendata@the\language}%
5054   {\def\bbl@elt##1##2##3##4{%
5055     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5056     \def\bbl@tempb{##3}%
5057     \ifx\bbl@tempb\@empty\else % if not a synonymous
5058       \def\bbl@tempc{##3}{##4}}%
5059     \fi
5060     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5061   \fi}%
5062   \bbl@languages
5063   \@ifundefined{bbl@hyphendata@the\language}%
5064   {\bbl@info{No hyphenation patterns were set for\%
5065     language '#2'. Reported}}%
5066   {\expandafter\expandafter\expandafter\bbl@luapatterns
5067     \csname bbl@hyphendata@the\language\endcsname}}}%
5068   \@ifundefined{bbl@patterns@}{}%
5069   \begingroup
5070     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5071     \ifin@else
5072       \ifx\bbl@patterns@\@empty\else
5073         \directlua{ Babel.addpatterns(
5074           [[\bbl@patterns@]], \number\language) }%
5075       \fi
5076       \@ifundefined{bbl@patterns@#1}%
5077       \@empty
5078       {\directlua{ Babel.addpatterns(
5079         [[\space\csname bbl@patterns@#1\endcsname]],
5080         \number\language) }}%
5081       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5082     \fi
5083   \endgroup}%
5084   \bbl@exp{%
5085     \bbl@ifunset{bbl@prehc@\languagename}{}%
5086     {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5087     {\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.

```

5088 \@onlypreamble\babelpatterns
5089 \AtEndOfPackage{%
5090   \newcommand\babelpatterns[2][\@empty]{%
5091     \ifx\bbl@patterns@\relax
5092       \let\bbl@patterns@\@empty
5093     \fi
5094     \ifx\bbl@pttnlist\@empty\else
5095       \bbl@warning{%

```



```

5096         You must not intermingle \string\selectlanguage\space and\\%
5097         \string\babelpatterns\space or some patterns will not\\%
5098         be taken into account. Reported}%
5099     \fi
5100     \ifx\@empty#1%
5101         \protected@edef\bbl@patterns@\bbl@patterns@\space#2}%
5102     \else
5103         \edef\bbl@tempb{\zap@space#1 \@empty}%
5104         \bbl@for\bbl@tempa\bbl@tempb{%
5105             \bbl@fixname\bbl@tempa
5106             \bbl@iflanguage\bbl@tempa{%
5107                 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5108                     \@ifundefined{bbl@patterns@\bbl@tempa}%
5109                     \@empty
5110                     {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5111                     #2}}}%
5112     \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.

```

5113% TODO - to a lua file
5114 \directlua{
5115     Babel = Babel or {}
5116     Babel.linebreaking = Babel.linebreaking or {}
5117     Babel.linebreaking.before = {}
5118     Babel.linebreaking.after = {}
5119     Babel.locale = {} % Free to use, indexed by \localeid
5120     function Babel.linebreaking.add_before(func)
5121         tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5122         table.insert(Babel.linebreaking.before, func)
5123     end
5124     function Babel.linebreaking.add_after(func)
5125         tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5126         table.insert(Babel.linebreaking.after, func)
5127     end
5128 }
5129 \def\bbl@intraspace#1 #2 #3\@{%%
5130     \directlua{
5131         Babel = Babel or {}
5132         Babel.intraspaces = Babel.intraspaces or {}
5133         Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5134             {b = #1, p = #2, m = #3}
5135         Babel.locale_props[\the\localeid].intraspace = %
5136             {b = #1, p = #2, m = #3}
5137     }}
5138 \def\bbl@intrapenalty#1\@{%%
5139     \directlua{
5140         Babel = Babel or {}
5141         Babel.intrapenalties = Babel.intrapenalties or {}
5142         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5143         Babel.locale_props[\the\localeid].intrapenalty = #1
5144     }}
5145 \begingroup
5146 \catcode`\%=12

```

```

5147 \catcode`\^=14
5148 \catcode`\'=12
5149 \catcode`\~=12
5150 \gdef\bbl@seaintraspace{^
5151   \let\bbl@seaintraspace\relax
5152   \directlua{
5153     Babel = Babel or {}
5154     Babel.sea_enabled = true
5155     Babel.sea_ranges = Babel.sea_ranges or {}
5156     function Babel.set_chranges (script, chrng)
5157       local c = 0
5158       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5159         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5160         c = c + 1
5161       end
5162     end
5163     function Babel.sea_disc_to_space (head)
5164       local sea_ranges = Babel.sea_ranges
5165       local last_char = nil
5166       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5167       for item in node.traverse(head) do
5168         local i = item.id
5169         if i == node.id'glyph' then
5170           last_char = item
5171         elseif i == 7 and item.subtype == 3 and last_char
5172           and last_char.char > 0x0C99 then
5173           quad = font.getfont(last_char.font).size
5174           for lg, rg in pairs(sea_ranges) do
5175             if last_char.char > rg[1] and last_char.char < rg[2] then
5176               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5177               local intraspace = Babel.intraspaces[lg]
5178               local intrapenalty = Babel.intrapenalties[lg]
5179               local n
5180               if intrapenalty ~= 0 then
5181                 n = node.new(14, 0)    ^% penalty
5182                 n.penalty = intrapenalty
5183                 node.insert_before(head, item, n)
5184               end
5185               n = node.new(12, 13)    ^% (glue, spaceskip)
5186               node.setglue(n, intraspace.b * quad,
5187                 intraspace.p * quad,
5188                 intraspace.m * quad)
5189               node.insert_before(head, item, n)
5190               node.remove(head, item)
5191             end
5192           end
5193         end
5194       end
5195     end
5196   }^^
5197   \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.

```
5198 \catcode`\%=14
5199 \gdef\bbl@cjkintraspacespace{%
5200   \let\bbl@cjkintraspacespace\relax
5201   \directlua{
5202     Babel = Babel or {}
5203     require('babel-data-cjk.lua')
5204     Babel.cjk_enabled = true
5205     function Babel.cjk_linebreak(head)
5206       local GLYPH = node.id'glyph'
5207       local last_char = nil
5208       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5209       local last_class = nil
5210       local last_lang = nil
5211
5212       for item in node.traverse(head) do
5213         if item.id == GLYPH then
5214
5215           local lang = item.lang
5216
5217           local LOCALE = node.get_attribute(item,
5218             Babel.attr_locale)
5219           local props = Babel.locale_props[LOCALE]
5220
5221           local class = Babel.cjk_class[item.char].c
5222
5223           if props.cjk_quotes and props.cjk_quotes[item.char] then
5224             class = props.cjk_quotes[item.char]
5225           end
5226
5227           if class == 'cp' then class = 'cl' end % ]] as CL
5228           if class == 'id' then class = 'I' end
5229
5230           local br = 0
5231           if class and last_class and Babel.cjk_breaks[last_class][class] then
5232             br = Babel.cjk_breaks[last_class][class]
5233           end
5234
5235           if br == 1 and props.linebreak == 'c' and
5236             lang ~= \the\l@nohyphenation\space and
5237             last_lang ~= \the\l@nohyphenation then
5238             local intrapenalty = props.intrapenalty
5239             if intrapenalty ~= 0 then
5240               local n = node.new(14, 0)      % penalty
5241               n.penalty = intrapenalty
5242               node.insert_before(head, item, n)
5243             end
5244             local intraspacespace = props.intraspacespace
5245             local n = node.new(12, 13)      % (glue, spaceskip)
5246             node.setglue(n, intraspacespace.b * quad,
5247               intraspacespace.p * quad,
5248               intraspacespace.m * quad)
5249             node.insert_before(head, item, n)
5250           end
5251
5252           if font.getfont(item.font) then
5253             quad = font.getfont(item.font).size
5254           end
5255         end
5256       end
5257     }
5258   }
5259 }
```

```

5255         last_class = class
5256         last_lang = lang
5257     else % if penalty, glue or anything else
5258         last_class = nil
5259     end
5260 end
5261 lang.hyphenate(head)
5262 end
5263 }%
5264 \bbl@luahyphenate}
5265 \gdef\bbl@luahyphenate{%
5266 \let\bbl@luahyphenate\relax
5267 \directlua{
5268     luatexbase.add_to_callback('hyphenate',
5269     function (head, tail)
5270         if Babel.linebreaking.before then
5271             for k, func in ipairs(Babel.linebreaking.before) do
5272                 func(head)
5273             end
5274         end
5275         if Babel.cjk_enabled then
5276             Babel.cjk_linebreak(head)
5277         end
5278         lang.hyphenate(head)
5279         if Babel.linebreaking.after then
5280             for k, func in ipairs(Babel.linebreaking.after) do
5281                 func(head)
5282             end
5283         end
5284         if Babel.sea_enabled then
5285             Babel.sea_disc_to_space(head)
5286         end
5287     end,
5288     'Babel.hyphenate')
5289 }
5290 }
5291 \endgroup
5292 \def\bbl@provide@intraspace{%
5293 \bbl@ifunset{\bbl@intsp@language}{}%
5294 {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5295 \bbl@xin@{c}{\bbl@cl{lnbrk}}%
5296 \ifin@ % cjk
5297 \bbl@cjk@intraspace
5298 \directlua{
5299     Babel = Babel or {}
5300     Babel.locale_props = Babel.locale_props or {}
5301     Babel.locale_props[\the\localeid].linebreak = 'c'
5302 }%
5303 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5304 \ifx\bbl@KVP@intrapenalty\@nil
5305 \bbl@intrapenalty0\@
5306 \fi
5307 \else % sea
5308 \bbl@sea@intraspace
5309 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5310 \directlua{
5311     Babel = Babel or {}
5312     Babel.sea_ranges = Babel.sea_ranges or {}
5313     Babel.set_chranges('\bbl@cl{sbcp}',

```

```

5314                                     '\bbl@cl{chrng}')
5315     }%
5316     \ifx\bbl@KVP@intrapenalty\@nil
5317     \bbl@intrapenalty0\@@
5318     \fi
5319     \fi
5320 \fi
5321 \ifx\bbl@KVP@intrapenalty\@nil\else
5322     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5323 \fi}}

```

13.6 Arabic justification

```

5324 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5325 \def\bblar@chars{%
5326     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5327     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5328     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5329 \def\bblar@elongated{%
5330     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5331     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5332     0649,064A}
5333 \begingroup
5334 \catcode\_ =11 \catcode\` =11
5335 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5336 \endgroup
5337 \gdef\bbl@arabicjust{%
5338     \let\bbl@arabicjust\relax
5339     \newattribute\bblar@kashida
5340     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5341     \bblar@kashida=\z@
5342     \bbl@patchfont{{\bbl@parsejalt}}}%
5343     \directlua{
5344         Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5345         Babel.arabic.elong_map[\the\localeid] = {}
5346         luatexbase.add_to_callback('post_linebreak_filter',
5347             Babel.arabic.justify, 'Babel.arabic.justify')
5348         luatexbase.add_to_callback('hpack_filter',
5349             Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5350     }}%
5351 % Save both node lists to make replacement. TODO. Save also widths to
5352 % make computations
5353 \def\bblar@fetchjalt#1#2#3#4{%
5354     \bbl@exp{\bbl@foreach{#1}}{%
5355         \bbl@ifunset{bblar@JE@##1}%
5356         {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5357         {\setbox\z@\hbox{^^^200d\char"@nameuse{bblar@JE@##1}#2}}%
5358     \directlua{%
5359         local last = nil
5360         for item in node.traverse(tex.box[0].head) do
5361             if item.id == node.id'glyph' and item.char > 0x600 and
5362             not (item.char == 0x200D) then
5363                 last = item
5364             end
5365         end
5366         Babel.arabic.#3['##1#4'] = last.char
5367     }}
5368 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5369 % perhaps other tables (falt?, csw?). What about kaf? And diacritic

```

```

5370% positioning?
5371 \gdef\bbl@parsejalt{%
5372   \ifx\addfontfeature\undefined\else
5373     \bbl@xin@{/e}{/\bbl@c1{\lnbrk}}%
5374     \ifin@
5375       \directlua{%
5376         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5377           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5378           tex.print([[string\curname\space bbl@parsejalti\endcsname]])
5379         end
5380       }%
5381   \fi
5382 \fi}
5383 \gdef\bbl@parsejalti{%
5384   \begingroup
5385     \let\bbl@parsejalt\relax      % To avoid infinite loop
5386     \edef\bbl@tempb{\fontid\font}%
5387     \bblar@nofswarn
5388     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5389     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5390     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5391     \addfontfeature{RawFeature+=jalt}%
5392     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5393     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5394     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5395     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5396     \directlua{%
5397       for k, v in pairs(Babel.arabic.from) do
5398         if Babel.arabic.dest[k] and
5399           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5400           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5401             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5402         end
5403       end
5404     }%
5405   \endgroup}
5406%
5407 \begingroup
5408 \catcode`#=11
5409 \catcode`~=11
5410 \directlua{
5411
5412 Babel.arabic = Babel.arabic or {}
5413 Babel.arabic.from = {}
5414 Babel.arabic.dest = {}
5415 Babel.arabic.justify_factor = 0.95
5416 Babel.arabic.justify_enabled = true
5417
5418 function Babel.arabic.justify(head)
5419   if not Babel.arabic.justify_enabled then return head end
5420   for line in node.traverse_id(node.id'hlist', head) do
5421     Babel.arabic.justify_hlist(head, line)
5422   end
5423   return head
5424 end
5425
5426 function Babel.arabic.justify_hbox(head, gc, size, pack)
5427   local has_inf = false
5428   if Babel.arabic.justify_enabled and pack == 'exactly' then

```

```

5429     for n in node.traverse_id(12, head) do
5430         if n.stretch_order > 0 then has_inf = true end
5431     end
5432     if not has_inf then
5433         Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5434     end
5435 end
5436 return head
5437 end
5438
5439 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5440     local d, new
5441     local k_list, k_item, pos_inline
5442     local width, width_new, full, k_curr, wt_pos, goal, shift
5443     local subst_done = false
5444     local elong_map = Babel.arabic.elong_map
5445     local last_line
5446     local GLYPH = node.id'glyph'
5447     local KASHIDA = Babel.attr_kashida
5448     local LOCALE = Babel.attr_locale
5449
5450     if line == nil then
5451         line = {}
5452         line.glue_sign = 1
5453         line.glue_order = 0
5454         line.head = head
5455         line.shift = 0
5456         line.width = size
5457     end
5458
5459     % Exclude last line. todo. But-- it discards one-word lines, too!
5460     % ? Look for glue = 12:15
5461     if (line.glue_sign == 1 and line.glue_order == 0) then
5462         elongs = {}      % Stores elongated candidates of each line
5463         k_list = {}      % And all letters with kashida
5464         pos_inline = 0   % Not yet used
5465
5466         for n in node.traverse_id(GLYPH, line.head) do
5467             pos_inline = pos_inline + 1 % To find where it is. Not used.
5468
5469             % Elongated glyphs
5470             if elong_map then
5471                 local locale = node.get_attribute(n, LOCALE)
5472                 if elong_map[locale] and elong_map[locale][n.font] and
5473                     elong_map[locale][n.font][n.char] then
5474                     table.insert(elongs, {node = n, locale = locale} )
5475                     node.set_attribute(n.prev, KASHIDA, 0)
5476                 end
5477             end
5478
5479             % Tatwil
5480             if Babel.kashida_wts then
5481                 local k_wt = node.get_attribute(n, KASHIDA)
5482                 if k_wt > 0 then % todo. parameter for multi inserts
5483                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5484                 end
5485             end
5486
5487         end % of node.traverse_id

```

```

5488
5489   if #elongs == 0 and #k_list == 0 then goto next_line end
5490   full = line.width
5491   shift = line.shift
5492   goal = full * Babel.arabic.justify_factor % A bit crude
5493   width = node.dimensions(line.head) % The 'natural' width
5494
5495   % == Elongated ==
5496   % Original idea taken from 'chickenize'
5497   while (#elongs > 0 and width < goal) do
5498     subst_done = true
5499     local x = #elongs
5500     local curr = elongs[x].node
5501     local oldchar = curr.char
5502     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5503     width = node.dimensions(line.head) % Check if the line is too wide
5504     % Substitute back if the line would be too wide and break:
5505     if width > goal then
5506       curr.char = oldchar
5507       break
5508     end
5509     % If continue, pop the just substituted node from the list:
5510     table.remove(elongs, x)
5511   end
5512
5513   % == Tatwil ==
5514   if #k_list == 0 then goto next_line end
5515
5516   width = node.dimensions(line.head) % The 'natural' width
5517   k_curr = #k_list
5518   wt_pos = 1
5519
5520   while width < goal do
5521     subst_done = true
5522     k_item = k_list[k_curr].node
5523     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5524       d = node.copy(k_item)
5525       d.char = 0x0640
5526       line.head, new = node.insert_after(line.head, k_item, d)
5527       width_new = node.dimensions(line.head)
5528       if width > goal or width == width_new then
5529         node.remove(line.head, new) % Better compute before
5530         break
5531       end
5532       width = width_new
5533     end
5534     if k_curr == 1 then
5535       k_curr = #k_list
5536       wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5537     else
5538       k_curr = k_curr - 1
5539     end
5540   end
5541
5542   ::next_line::
5543
5544   % Must take into account marks and ins, see luatex manual.
5545   % Have to be executed only if there are changes. Investigate
5546   % what's going on exactly.

```



```

5547     if subst_done and not gc then
5548         d = node.hpack(line.head, full, 'exactly')
5549         d.shift = shift
5550         node.insert_before(head, line, d)
5551         node.remove(head, line)
5552     end
5553 end % if process line
5554 end
5555 }
5556 \endgroup
5557 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5558 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5559 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5560 \DisableBabelHook{babel-fontspec}
5561 <<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.

```

5562 % TODO - to a lua file
5563 \directlua{
5564 Babel.script_blocks = {
5565   ['dflt'] = {},
5566   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5567               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5568   ['Armn'] = {{0x0530, 0x058F}},
5569   ['Beng'] = {{0x0980, 0x09FF}},
5570   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5571   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5572   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5573               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5574   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5575   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5576               {0xAB00, 0xAB2F}},
5577   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5578   % Don't follow strictly Unicode, which places some Coptic letters in
5579   % the 'Greek and Coptic' block
5580   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5581   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5582               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5583               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5584               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5585               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5586               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5587   ['Hebr'] = {{0x0590, 0x05FF}},
5588   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5589               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5590   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5591   ['Knda'] = {{0x0C80, 0x0CFF}},
5592   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5593               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF}},

```

```

5594         {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5595 ['Lao'] = {{0x0E80, 0x0EFF}},
5596 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5597             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5598             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5599 ['Mahj'] = {{0x11150, 0x1117F}},
5600 ['Mlym'] = {{0x0D00, 0x0D7F}},
5601 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5602 ['Orya'] = {{0x0B00, 0x0B7F}},
5603 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5604 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5605 ['Taml'] = {{0x0B80, 0x0BFF}},
5606 ['Telu'] = {{0x0C00, 0x0C7F}},
5607 ['Tfng'] = {{0x2D30, 0x2D7F}},
5608 ['Thai'] = {{0x0E00, 0x0E7F}},
5609 ['Tibt'] = {{0x0F00, 0x0FFF}},
5610 ['Vaii'] = {{0xA500, 0xA63F}},
5611 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5612 }
5613
5614 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5615 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5616 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5617
5618 function Babel.locale_map(head)
5619   if not Babel.locale_mapped then return head end
5620
5621   local LOCALE = Babel.attr_locale
5622   local GLYPH = node.id('glyph')
5623   local inmath = false
5624   local toloc_save
5625   for item in node.traverse(head) do
5626     local toloc
5627     if not inmath and item.id == GLYPH then
5628       % Optimization: build a table with the chars found
5629       if Babel.chr_to_loc[item.char] then
5630         toloc = Babel.chr_to_loc[item.char]
5631       else
5632         for lc, maps in pairs(Babel.loc_to_scr) do
5633           for _, rg in pairs(maps) do
5634             if item.char >= rg[1] and item.char <= rg[2] then
5635               Babel.chr_to_loc[item.char] = lc
5636               toloc = lc
5637               break
5638             end
5639           end
5640         end
5641       end
5642       % Now, take action, but treat composite chars in a different
5643       % fashion, because they 'inherit' the previous locale. Not yet
5644       % optimized.
5645       if not toloc and
5646         (item.char >= 0x0300 and item.char <= 0x036F) or
5647         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5648         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5649         toloc = toloc_save
5650       end
5651       if toloc and toloc > -1 then
5652         if Babel.locale_props[toloc].lg then

```

```

5653         item.lang = Babel.locale_props[toloc].lg
5654         node.set_attribute(item, LOCALE, toloc)
5655     end
5656     if Babel.locale_props[toloc]['/'..item.font] then
5657         item.font = Babel.locale_props[toloc]['/'..item.font]
5658     end
5659     toloc_save = toloc
5660 end
5661 elseif not inmath and item.id == 7 then
5662     item.replace = item.replace and Babel.locale_map(item.replace)
5663     item.pre      = item.pre and Babel.locale_map(item.pre)
5664     item.post     = item.post and Babel.locale_map(item.post)
5665 elseif item.id == node.id'math' then
5666     inmath = (item.subtype == 0)
5667 end
5668 end
5669 return head
5670 end
5671 }

```

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

```

5672 \newcommand\babelcharproperty[1]{%
5673   \count@=#1\relax
5674   \ifvmode
5675     \expandafter\bbl@chprop
5676   \else
5677     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5678               vertical mode (preamble or between paragraphs)}%
5679     {See the manual for futher info}%
5680   \fi}
5681 \newcommand\bbl@chprop[3][\the\count@]{%
5682   \@tempcnta=#1\relax
5683   \bbl@ifunset{\bbl@chprop@#2}%
5684   {\bbl@error{No property named '#2'. Allowed values are\\%
5685             direction (bc), mirror (bmg), and linebreak (lb)}%
5686    {See the manual for futher info}}%
5687   {%
5688   \loop
5689     \bbl@cs{chprop@#2}{#3}%
5690   \ifnum\count@<\@tempcnta
5691     \advance\count@\@ne
5692   \repeat}
5693 \def\bbl@chprop@direction#1{%
5694   \directlua{
5695     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5696     Babel.characters[\the\count@]['d'] = '#1'
5697   }}
5698 \let\bbl@chprop@bc\bbl@chprop@direction
5699 \def\bbl@chprop@mirror#1{%
5700   \directlua{
5701     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5702     Babel.characters[\the\count@]['m'] = '\number#1'
5703   }}
5704 \let\bbl@chprop@bmg\bbl@chprop@mirror
5705 \def\bbl@chprop@linebreak#1{%
5706   \directlua{
5707     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5708     Babel.cjk_characters[\the\count@]['c'] = '#1'

```

```

5709 }}
5710 \let\bbl@chprop@lb\bbl@chprop@linebreak
5711 \def\bbl@chprop@locale#1{%
5712   \directlua{
5713     Babel.chr_to_loc = Babel.chr_to_loc or {}
5714     Babel.chr_to_loc[\the\count@] =
5715       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5716   }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow). The Lua code is below.

```

5717 \directlua{
5718   Babel.nohyphenation = \the\l@nohyphenation
5719 }

```

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).`

```

5720 \begingroup
5721 \catcode`\~ = 12
5722 \catcode`\% = 12
5723 \catcode`\& = 14
5724 \gdef\babelposthyphenation#1#2#3{&%
5725   \bbl@activateposthyphen
5726   \begingroup
5727     \def\babeltempa{\bbl@add@list\babeltempb}&%
5728     \let\babeltempb\@empty
5729     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5730     \bbl@replace\bbl@tempa{,}{ ,}&%
5731     \expandafter\bbl@foreach\expandafter{\bbl@tempa}&%
5732     \bbl@ifsamestring{##1}{remove}&%
5733     {\bbl@add@list\babeltempb{nil}}&%
5734     {\directlua{
5735       local rep = {[#1]=}
5736       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5737       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5738       rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5739       rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5740       rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5741       rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5742       tex.print([[ \string\babeltempa{ } ] .. rep .. [{}]])
5743     }}&%
5744   \directlua{
5745     local lbkr = Babel.linebreaking.replacements[1]
5746     local u = unicode.utf8
5747     local id = \the\csname l@#1\endcsname
5748     &% Convert pattern:
5749     local patt = string.gsub([=[#2]=], '%s', '')
5750     if not u.find(patt, '()', nil, true) then
5751       patt = '()' .. patt .. '()'
5752     end
5753     patt = string.gsub(patt, '%(%)%', '^()')
5754     patt = string.gsub(patt, '%$(%)', '()$')

```

```

5755     patt = u.gsub(patt, '{(.)}',
5756         function (n)
5757             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5758         end)
5759     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5760         function (n)
5761             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5762         end)
5763     lbkr[id] = lbkr[id] or {}
5764     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5765 }&%
5766 \endgroup}
5767 % TODO. Copypaste pattern.
5768 \gdef\babelprehyphenation#1#2#3{&%
5769     \bbl@activateprehyphen
5770     \begingroup
5771     \def\babeltempa{\bbl@add@list\babeltempb}&%
5772     \let\babeltempb\@empty
5773     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5774     \bbl@replace\bbl@tempa{,}{ ,}&%
5775     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5776         \bbl@ifsamestring{##1}{remove}&%
5777         {\bbl@add@list\babeltempb{nil}}&%
5778         {\directlua{
5779             local rep = {[##1]=}
5780             rep = rep.gsub('^%s*(remove)%s*$', 'remove = true')
5781             rep = rep.gsub('^%s*(insert)%s*', 'insert = true, ')
5782             rep = rep.gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5783             rep = rep.gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5784                 'space = {' .. '%2, %3, %4' .. '}')
5785             rep = rep.gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5786                 'spacefactor = {' .. '%2, %3, %4' .. '}')
5787             rep = rep.gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5788             tex.print([[ \string\babeltempa{[]} .. rep .. []]])
5789         }}&%
5790     \directlua{
5791         local lbkr = Babel.linebreaking.replacements[0]
5792         local u = unicode.utf8
5793         local id = \the\csname bbl@id@@#1\endcsname
5794         &% Convert pattern:
5795         local patt = string.gsub(==[#2]==, '%s', '')
5796         local patt = string.gsub(patt, '|', ' ')
5797         if not u.find(patt, '()', nil, true) then
5798             patt = '()' .. patt .. '()'
5799         end
5800         &% patt = string.gsub(patt, '%(%)%', '^()')
5801         &% patt = string.gsub(patt, '([^\%])%$%$', '%1()$')
5802         patt = u.gsub(patt, '{(.)}',
5803             function (n)
5804                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5805             end)
5806         patt = u.gsub(patt, '{(%x%x%x%x+)}',
5807             function (n)
5808                 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5809             end)
5810         lbkr[id] = lbkr[id] or {}
5811         table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5812     }&%
5813 \endgroup}

```

```

5814 \endgroup
5815 \def\bbl@activateposthyphen{%
5816   \let\bbl@activateposthyphen\relax
5817   \directlua{
5818     require('babel-transforms.lua')
5819     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5820   }}
5821 \def\bbl@activateprehyphen{%
5822   \let\bbl@activateprehyphen\relax
5823   \directlua{
5824     require('babel-transforms.lua')
5825     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5826   }}

```

13.9 Bidi

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

```

5827 \def\bbl@activate@preotf{%
5828   \let\bbl@activate@preotf\relax % only once
5829   \directlua{
5830     Babel = Babel or {}
5831     %
5832     function Babel.pre_otfload_v(head)
5833       if Babel.numbers and Babel.digits_mapped then
5834         head = Babel.numbers(head)
5835       end
5836       if Babel.bidi_enabled then
5837         head = Babel.bidi(head, false, dir)
5838       end
5839       return head
5840     end
5841     %
5842     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5843       if Babel.numbers and Babel.digits_mapped then
5844         head = Babel.numbers(head)
5845       end
5846       if Babel.bidi_enabled then
5847         head = Babel.bidi(head, false, dir)
5848       end
5849       return head
5850     end
5851     %
5852     luatexbase.add_to_callback('pre_linebreak_filter',
5853       Babel.pre_otfload_v,
5854       'Babel.pre_otfload_v',
5855       luatexbase.priority_in_callback('pre_linebreak_filter',
5856         'luaotfload.node_processor') or nil)
5857     %
5858     luatexbase.add_to_callback('hpack_filter',
5859       Babel.pre_otfload_h,
5860       'Babel.pre_otfload_h',
5861       luatexbase.priority_in_callback('hpack_filter',
5862         'luaotfload.node_processor') or nil)
5863   }}

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math

with the package option bidi=.

```

5864 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5865 \let\bbl@beforeforeign\leavevmode
5866 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5867 \RequirePackage{luatexbase}
5868 \bbl@activate@preotf
5869 \directlua{
5870   require('babel-data-bidi.lua')
5871   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5872     require('babel-bidi-basic.lua')
5873   \or
5874     require('babel-bidi-basic-r.lua')
5875   \fi}
5876 % TODO - to locale_props, not as separate attribute
5877 \newattribute\bbl@attr@dir
5878 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5879 % TODO. I don't like it, hackish:
5880 \bbl@exp{\output{\bodydir\pagedir\the\output}}
5881 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5882 \fi\fi
5883 \chardef\bbl@thetextdir\z@
5884 \chardef\bbl@thepardir\z@
5885 \def\bbl@getluadir#1{%
5886   \directlua{
5887     if tex.#1dir == 'TLT' then
5888       tex.sprint('0')
5889     elseif tex.#1dir == 'TRT' then
5890       tex.sprint('1')
5891     end}}
5892 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5893   \ifcase#3\relax
5894     \ifcase\bbl@getluadir{#1}\relax\else
5895       #2 TLT\relax
5896     \fi
5897   \else
5898     \ifcase\bbl@getluadir{#1}\relax
5899       #2 TRT\relax
5900     \fi
5901   \fi}
5902 \def\bbl@textdir#1{%
5903   \bbl@setluadir{text}\textdir{#1}%
5904   \chardef\bbl@thetextdir#1\relax
5905   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5906 \def\bbl@pardir#1{%
5907   \bbl@setluadir{par}\pardir{#1}%
5908   \chardef\bbl@thepardir#1\relax}
5909 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5910 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5911 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
5912 %
5913 \ifnum\bbl@bidimode>\z@
5914   \def\bbl@mathboxdir{%
5915     \ifcase\bbl@thetextdir\relax
5916       \everyhbox{\bbl@mathboxdir@aux L}%
5917     \else
5918       \everyhbox{\bbl@mathboxdir@aux R}%
5919     \fi}
5920   \def\bbl@mathboxdir@aux#1{%

```

```

5921 \@ifnextchar\egroup{}\textdir T#1T\relax}
5922 \frozen@everymath\expandafter{%
5923 \expandafter\bbl@mathboxdir\the\frozen@everymath}
5924 \frozen@everydisplay\expandafter{%
5925 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5926 \fi

```

13.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5927 \bbl@trace{Redefinitions for bidi layout}
5928 \ifx\@eqnnum\undefined\else
5929 \ifx\bbl@attr@dir\undefined\else
5930 \edef\@eqnnum{%
5931 \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5932 \unexpanded\expandafter{\@eqnnum}}
5933 \fi
5934 \fi
5935 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5936 \ifnum\bbl@bidimode>\z@
5937 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5938 \bbl@exp{%
5939 \mathdir\the\bodydir
5940 #1% Once entered in math, set boxes to restore values
5941 \<ifmmode>%
5942 \everyvbox{%
5943 \the\everyvbox
5944 \bodydir\the\bodydir
5945 \mathdir\the\mathdir
5946 \everyhbox{\the\everyhbox}%
5947 \everyvbox{\the\everyvbox}}%
5948 \everyhbox{%
5949 \the\everyhbox
5950 \bodydir\the\bodydir
5951 \mathdir\the\mathdir
5952 \everyhbox{\the\everyhbox}%
5953 \everyvbox{\the\everyvbox}}%
5954 \<fi>}}%
5955 \def\@hangfrom#1{%
5956 \setbox\@tempboxa\hbox{{#1}}%
5957 \hangindent\wd\@tempboxa
5958 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5959 \shapemode\@ne
5960 \fi
5961 \noindent\box\@tempboxa}
5962 \fi
5963 \IfBabelLayout{tabular}

```



```

5964 {\let\bbl@OL@tabular\@tabular
5965 \bbl@replace\@tabular{$}\bbl@nextfake$}%
5966 \let\bbl@NL@tabular\@tabular
5967 \AtBeginDocument{%
5968 \ifx\bbl@NL@tabular\@tabular\else
5969 \bbl@replace\@tabular{$}\bbl@nextfake$}%
5970 \let\bbl@NL@tabular\@tabular
5971 \fi}}
5972 {}
5973 \IfBabelLayout{lists}
5974 {\let\bbl@OL@list\list
5975 \bbl@sreplace\list{\parshape}\bbl@listparshape}%
5976 \let\bbl@NL@list\list
5977 \def\bbl@listparshape#1#2#3{%
5978 \parshape #1 #2 #3 %
5979 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5980 \shapemode\tw@
5981 \fi}}
5982 {}
5983 \IfBabelLayout{graphics}
5984 {\let\bbl@pictresetdir\relax
5985 \def\bbl@pictsetdir#1{%
5986 \ifcase\bbl@thetextdir
5987 \let\bbl@pictresetdir\relax
5988 \else
5989 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5990 \or\textdir TLT
5991 \else\bodydir TLT \textdir TLT
5992 \fi
5993 % \(\text|par)dir required in pgf:
5994 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5995 \fi}%
5996 \ifx\AddToHook\@undefined\else
5997 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
5998 \directlua{
5999 Babel.get_picture_dir = true
6000 Babel.picture_has_bidi = 0
6001 function Babel.picture_dir (head)
6002 if not Babel.get_picture_dir then return head end
6003 for item in node.traverse(head) do
6004 if item.id == node.id'glyph' then
6005 local itemchar = item.char
6006 % TODO. Copy paste pattern from Babel.bidi (-r)
6007 local chardata = Babel.characters[itemchar]
6008 local dir = chardata and chardata.d or nil
6009 if not dir then
6010 for nn, et in ipairs(Babel.ranges) do
6011 if itemchar < et[1] then
6012 break
6013 elseif itemchar <= et[2] then
6014 dir = et[3]
6015 break
6016 end
6017 end
6018 end
6019 if dir and (dir == 'al' or dir == 'r') then
6020 Babel.picture_has_bidi = 1
6021 end
6022 end

```

```

6023         end
6024         return head
6025     end
6026     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6027         "Babel.picture_dir")
6028 }%
6029 \AtBeginDocument{%
6030     \long\def\put(#1,#2)#3{%
6031         \@killglue
6032         % Try:
6033         \ifx\bbbl@pictresetdir\relax
6034             \def\bbbl@tempc{0}%
6035         \else
6036             \directlua{
6037                 Babel.get_picture_dir = true
6038                 Babel.picture_has_bidi = 0
6039             }%
6040             \setbox\z@\hb@xt@\z@{%
6041                 \@defaultunitsset\@tempdimc{#1}\unitlength
6042                 \kern\@tempdimc
6043                 #3\hss}%
6044             \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6045         \fi
6046         % Do:
6047         \@defaultunitsset\@tempdimc{#2}\unitlength
6048         \raise\@tempdimc\hb@xt@\z@{%
6049             \@defaultunitsset\@tempdimc{#1}\unitlength
6050             \kern\@tempdimc
6051             {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6052         \ignorespaces}%
6053         \MakeRobust\put}%
6054 \fi
6055 \AtBeginDocument
6056 {\ifx\tikz@atbegin@node\@undefined\else
6057     \ifx\AddToHook\@undefined\else % TODO. Still tentative.
6058         \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6059         \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6060     \fi
6061     \let\bbbl@OL@pgfpicture\pgfpicture
6062     \bbbl@sreplace\pgfpicture{\pgfpicturetrue}%
6063     {\bbbl@pictsetdir\z@\pgfpicturetrue}%
6064     \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6065     \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6066     \bbbl@sreplace\tikz{\beginpgroup}%
6067     {\beginpgroup\bbbl@pictsetdir\tw@}%
6068 \fi
6069 \ifx\AddToHook\@undefined\else
6070     \AddToHook{env/tcolorbox/begin}{\bbbl@pictsetdir\@ne}%
6071 \fi
6072 }}
6073 {}

```

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.

```

6074 \IfBabelLayout{counters}%
6075 {\let\bbbl@OL@@textsuperscript\textsuperscript
6076     \bbbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6077     \let\bbbl@latinarabic=\@arabic

```

```

6078 \let\bbl@OL@@arabic\@arabic
6079 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6080 \@ifpackagewith{babel}{\bidi=default}%
6081 {\let\bbl@asciroman=\@roman
6082 \let\bbl@OL@@roman\@roman
6083 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6084 \let\bbl@asciiRoman=\@Roman
6085 \let\bbl@OL@@roman\@Roman
6086 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6087 \let\bbl@OL@labelenumii\labelenumii
6088 \def\labelenumii{}\theenumii}%
6089 \let\bbl@OL@p@enumiii\p@enumiii
6090 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
6091 <<Footnote changes>>
6092 \IfBabelLayout{footnotes}%
6093 {\let\bbl@OL@footnote\footnote
6094 \BabelFootnote\footnote\language{}{}\}%
6095 \BabelFootnote\localfootnote\language{}{}\}%
6096 \BabelFootnote\mainfootnote{}\}\}\}\}
6097 {}

```

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.

```

6098 \IfBabelLayout{extras}%
6099 {\let\bbl@OL@underline\underline
6100 \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake\$@@underline}%
6101 \let\bbl@OL@LaTeX2e\LaTeX2e
6102 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6103 \if b\expandafter\@car\@series\@nil\boldmath\fi
6104 \babelsublr{%
6105 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6106 {}
6107 </luatex>

```

13.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

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

```

6108 (*transforms)
6109 Babel.linebreaking.replacements = {}
6110 Babel.linebreaking.replacements[0] = {} -- pre
6111 Babel.linebreaking.replacements[1] = {} -- post
6112
6113 -- Discretionaries contain strings as nodes
6114 function Babel.str_to_nodes(fn, matches, base)
6115   local n, head, last
6116   if fn == nil then return nil end
6117   for s in string.utfvalues(fn(matches)) do
6118     if base.id == 7 then

```

```

6119     base = base.replace
6120 end
6121 n = node.copy(base)
6122 n.char = s
6123 if not head then
6124     head = n
6125 else
6126     last.next = n
6127 end
6128 last = n
6129 end
6130 return head
6131 end
6132
6133 Babel.fetch_subtext = {}
6134
6135 Babel.ignore_pre_char = function(node)
6136     return (node.lang == Babel.nohyphenation)
6137 end
6138
6139 -- Merging both functions doesn't seem feasible, because there are too
6140 -- many differences.
6141 Babel.fetch_subtext[0] = function(head)
6142     local word_string = ''
6143     local word_nodes = {}
6144     local lang
6145     local item = head
6146     local inmath = false
6147
6148     while item do
6149
6150         if item.id == 11 then
6151             inmath = (item.subtype == 0)
6152         end
6153
6154         if inmath then
6155             -- pass
6156
6157         elseif item.id == 29 then
6158             local locale = node.get_attribute(item, Babel.attr_locale)
6159
6160             if lang == locale or lang == nil then
6161                 lang = lang or locale
6162                 if Babel.ignore_pre_char(item) then
6163                     word_string = word_string .. Babel.us_char
6164                 else
6165                     word_string = word_string .. unicode.utf8.char(item.char)
6166                 end
6167                 word_nodes[#word_nodes+1] = item
6168             else
6169                 break
6170             end
6171
6172         elseif item.id == 12 and item.subtype == 13 then
6173             word_string = word_string .. ' '
6174             word_nodes[#word_nodes+1] = item
6175
6176         -- Ignore leading unrecognized nodes, too.
6177         elseif word_string ~= '' then

```

```

6178     word_string = word_string .. Babel.us_char
6179     word_nodes[#word_nodes+1] = item -- Will be ignored
6180 end
6181
6182     item = item.next
6183 end
6184
6185 -- Here and above we remove some trailing chars but not the
6186 -- corresponding nodes. But they aren't accessed.
6187 if word_string:sub(-1) == ' ' then
6188     word_string = word_string:sub(1,-2)
6189 end
6190 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6191 return word_string, word_nodes, item, lang
6192 end
6193
6194 Babel.fetch_subtext[1] = function(head)
6195     local word_string = ''
6196     local word_nodes = {}
6197     local lang
6198     local item = head
6199     local inmath = false
6200
6201     while item do
6202
6203         if item.id == 11 then
6204             inmath = (item.subtype == 0)
6205         end
6206
6207         if inmath then
6208             -- pass
6209
6210         elseif item.id == 29 then
6211             if item.lang == lang or lang == nil then
6212                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6213                     lang = lang or item.lang
6214                     word_string = word_string .. unicode.utf8.char(item.char)
6215                     word_nodes[#word_nodes+1] = item
6216                 end
6217             else
6218                 break
6219             end
6220
6221         elseif item.id == 7 and item.subtype == 2 then
6222             word_string = word_string .. '='
6223             word_nodes[#word_nodes+1] = item
6224
6225         elseif item.id == 7 and item.subtype == 3 then
6226             word_string = word_string .. '|'
6227             word_nodes[#word_nodes+1] = item
6228
6229         -- (1) Go to next word if nothing was found, and (2) implicitly
6230         -- remove leading USs.
6231         elseif word_string == '' then
6232             -- pass
6233
6234         -- This is the responsible for splitting by words.
6235         elseif (item.id == 12 and item.subtype == 13) then
6236             break

```

```

6237
6238     else
6239         word_string = word_string .. Babel.us_char
6240         word_nodes[#word_nodes+1] = item -- Will be ignored
6241     end
6242
6243     item = item.next
6244 end
6245
6246 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6247 return word_string, word_nodes, item, lang
6248 end
6249
6250 function Babel.pre_hyphenate_replace(head)
6251     Babel.hyphenate_replace(head, 0)
6252 end
6253
6254 function Babel.post_hyphenate_replace(head)
6255     Babel.hyphenate_replace(head, 1)
6256 end
6257
6258 Babel.us_char = string.char(31)
6259
6260 function Babel.hyphenate_replace(head, mode)
6261     local u = unicode.utf8
6262     local lbkr = Babel.linebreaking.replacements[mode]
6263
6264     local word_head = head
6265
6266     while true do -- for each subtext block
6267
6268         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6269
6270         if Babel.debug then
6271             print()
6272             print((mode == 0) and '@@@<' or '@@@>', w)
6273         end
6274
6275         if nw == nil and w == '' then break end
6276
6277         if not lang then goto next end
6278         if not lbkr[lang] then goto next end
6279
6280         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6281         -- loops are nested.
6282         for k=1, #lbkr[lang] do
6283             local p = lbkr[lang][k].pattern
6284             local r = lbkr[lang][k].replace
6285
6286             if Babel.debug then
6287                 print('*****', p, mode)
6288             end
6289
6290             -- This variable is set in some cases below to the first *byte*
6291             -- after the match, either as found by u.match (faster) or the
6292             -- computed position based on sc if w has changed.
6293             local last_match = 0
6294             local step = 0
6295

```

```

6296     -- For every match.
6297 while true do
6298     if Babel.debug then
6299         print('====')
6300     end
6301     local new -- used when inserting and removing nodes
6302
6303     local matches = { u.match(w, p, last_match) }
6304
6305     if #matches < 2 then break end
6306
6307     -- Get and remove empty captures (with ()'s, which return a
6308     -- number with the position), and keep actual captures
6309     -- (from (...)), if any, in matches.
6310     local first = table.remove(matches, 1)
6311     local last  = table.remove(matches, #matches)
6312     -- Non re-fetched substrings may contain \31, which separates
6313     -- subsubstrings.
6314     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6315
6316     local save_last = last -- with A()BC()D, points to D
6317
6318     -- Fix offsets, from bytes to unicode. Explained above.
6319     first = u.len(w:sub(1, first-1)) + 1
6320     last  = u.len(w:sub(1, last-1)) -- now last points to C
6321
6322     -- This loop stores in n small table the nodes
6323     -- corresponding to the pattern. Used by 'data' to provide a
6324     -- predictable behavior with 'insert' (now w_nodes is modified on
6325     -- the fly), and also access to 'remove'd nodes.
6326     local sc = first-1 -- Used below, too
6327     local data_nodes = {}
6328
6329     for q = 1, last-first+1 do
6330         data_nodes[q] = w_nodes[sc+q]
6331     end
6332
6333     -- This loop traverses the matched substring and takes the
6334     -- corresponding action stored in the replacement list.
6335     -- sc = the position in substr nodes / string
6336     -- rc = the replacement table index
6337     local rc = 0
6338
6339     while rc < last-first+1 do -- for each replacement
6340         if Babel.debug then
6341             print('.....', rc + 1)
6342         end
6343         sc = sc + 1
6344         rc = rc + 1
6345
6346         if Babel.debug then
6347             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6348             local ss = ''
6349             for itt in node.traverse(head) do
6350                 if itt.id == 29 then
6351                     ss = ss .. unicode.utf8.char(itt.char)
6352                 else
6353                     ss = ss .. '{' .. itt.id .. '}'
6354                 end

```

```

6355         end
6356         print('*****', ss)
6357
6358     end
6359
6360     local crep = r[rc]
6361     local item = w_nodes[sc]
6362     local item_base = item
6363     local placeholder = Babel.us_char
6364     local d
6365
6366     if crep and crep.data then
6367         item_base = data_nodes[crep.data]
6368     end
6369
6370     if crep then
6371         step = crep.step or 0
6372     end
6373
6374     if crep and next(crep) == nil then -- = {}
6375         last_match = save_last    -- Optimization
6376         goto next
6377
6378     elseif crep == nil or crep.remove then
6379         node.remove(head, item)
6380         table.remove(w_nodes, sc)
6381         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6382         sc = sc - 1 -- Nothing has been inserted.
6383         last_match = utf8.offset(w, sc+1+step)
6384         goto next
6385
6386     elseif crep and crep.kashida then -- Experimental
6387         node.set_attribute(item,
6388             Babel.attr_kashida,
6389             crep.kashida)
6390         last_match = utf8.offset(w, sc+1+step)
6391         goto next
6392
6393     elseif crep and crep.string then
6394         local str = crep.string(matches)
6395         if str == '' then -- Gather with nil
6396             node.remove(head, item)
6397             table.remove(w_nodes, sc)
6398             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6399             sc = sc - 1 -- Nothing has been inserted.
6400         else
6401             local loop_first = true
6402             for s in string.utfvalues(str) do
6403                 d = node.copy(item_base)
6404                 d.char = s
6405                 if loop_first then
6406                     loop_first = false
6407                     head, new = node.insert_before(head, item, d)
6408                     if sc == 1 then
6409                         word_head = head
6410                     end
6411                     w_nodes[sc] = d
6412                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6413                 else

```



```

6414         sc = sc + 1
6415         head, new = node.insert_before(head, item, d)
6416         table.insert(w_nodes, sc, new)
6417         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6418     end
6419     if Babel.debug then
6420         print('.....', 'str')
6421         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6422     end
6423     end -- for
6424     node.remove(head, item)
6425 end -- if ''
6426 last_match = utf8.offset(w, sc+1+step)
6427 goto next
6428
6429 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6430     d = node.new(7, 0) -- (disc, discretionary)
6431     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6432     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6433     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6434     d.attr = item_base.attr
6435     if crep.pre == nil then -- TeXbook p96
6436         d.penalty = crep.penalty or tex.hyphenpenalty
6437     else
6438         d.penalty = crep.penalty or tex.exhyphenpenalty
6439     end
6440     placeholder = '|'
6441     head, new = node.insert_before(head, item, d)
6442
6443 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6444     -- ERROR
6445
6446 elseif crep and crep.penalty then
6447     d = node.new(14, 0) -- (penalty, userpenalty)
6448     d.attr = item_base.attr
6449     d.penalty = crep.penalty
6450     head, new = node.insert_before(head, item, d)
6451
6452 elseif crep and crep.space then
6453     -- 655360 = 10 pt = 10 * 65536 sp
6454     d = node.new(12, 13) -- (glue, spaceskip)
6455     local quad = font.getfont(item_base.font).size or 655360
6456     node.setglue(d, crep.space[1] * quad,
6457                  crep.space[2] * quad,
6458                  crep.space[3] * quad)
6459     if mode == 0 then
6460         placeholder = ' '
6461     end
6462     head, new = node.insert_before(head, item, d)
6463
6464 elseif crep and crep.spacefactor then
6465     d = node.new(12, 13) -- (glue, spaceskip)
6466     local base_font = font.getfont(item_base.font)
6467     node.setglue(d,
6468                  crep.spacefactor[1] * base_font.parameters['space'],
6469                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6470                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6471     if mode == 0 then
6472         placeholder = ' '

```

```

6473         end
6474         head, new = node.insert_before(head, item, d)
6475
6476     elseif mode == 0 and crep and crep.space then
6477         -- ERROR
6478
6479     end -- ie replacement cases
6480
6481     -- Shared by disc, space and penalty.
6482     if sc == 1 then
6483         word_head = head
6484     end
6485     if crep.insert then
6486         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6487         table.insert(w_nodes, sc, new)
6488         last = last + 1
6489     else
6490         w_nodes[sc] = d
6491         node.remove(head, item)
6492         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6493     end
6494
6495     last_match = utf8.offset(w, sc+1+step)
6496
6497     ::next::
6498
6499     end -- for each replacement
6500
6501     if Babel.debug then
6502         print('.....', '/')
6503         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6504     end
6505
6506     end -- for match
6507
6508     end -- for patterns
6509
6510     ::next::
6511     word_head = nw
6512 end -- for substring
6513 return head
6514 end
6515
6516 -- This table stores capture maps, numbered consecutively
6517 Babel.capture_maps = {}
6518
6519 -- The following functions belong to the next macro
6520 function Babel.capture_func(key, cap)
6521     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[(") .. "]"
6522     local cnt
6523     local u = unicode.utf8
6524     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(-)}}', Babel.capture_func_map)
6525     if cnt == 0 then
6526         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
6527             function (n)
6528                 return u.char(tonumber(n, 16))
6529             end)
6530     end
6531     ret = ret:gsub("%[%[%]"]%.", '')

```

```

6532 ret = ret:gsub("%.%.%[%[%]]", '')
6533 return key .. [[=function(m) return ]] .. ret .. [[ end]]
6534 end
6535
6536 function Babel.capt_map(from, mapno)
6537 return Babel.capture_maps[mapno][from] or from
6538 end
6539
6540 -- Handle the {n|abc|ABC} syntax in captures
6541 function Babel.capture_func_map(capno, from, to)
6542 local u = unicode.utf8
6543 from = u.gsub(from, '{(%x%x%x%x+)}',
6544 function (n)
6545 return u.char(tonumber(n, 16))
6546 end)
6547 to = u.gsub(to, '{(%x%x%x%x+)}',
6548 function (n)
6549 return u.char(tonumber(n, 16))
6550 end)
6551 local froms = {}
6552 for s in string.utfcharacters(from) do
6553 table.insert(froms, s)
6554 end
6555 local cnt = 1
6556 table.insert(Babel.capture_maps, {})
6557 local mlen = table.getn(Babel.capture_maps)
6558 for s in string.utfcharacters(to) do
6559 Babel.capture_maps[mlen][froms[cnt]] = s
6560 cnt = cnt + 1
6561 end
6562 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6563 (mlen) .. ").." .. "[["
6564 end
6565
6566 -- Create/Extend reversed sorted list of kashida weights:
6567 function Babel.capture_kashida(key, wt)
6568 wt = tonumber(wt)
6569 if Babel.kashida_wts then
6570 for p, q in ipairs(Babel.kashida_wts) do
6571 if wt == q then
6572 break
6573 elseif wt > q then
6574 table.insert(Babel.kashida_wts, p, wt)
6575 break
6576 elseif table.getn(Babel.kashida_wts) == p then
6577 table.insert(Babel.kashida_wts, wt)
6578 end
6579 end
6580 else
6581 Babel.kashida_wts = { wt }
6582 end
6583 return 'kashida = ' .. wt
6584 end
6585 </transforms>

```

13.12 Lua: Auto bidi with basic and basic-r

The file babel-data-bidi.lua currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the basic-r bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

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

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

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

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

```

6586 (*basic-r)
6587 Babel = Babel or {}
6588
6589 Babel.bidi_enabled = true
6590
6591 require('babel-data-bidi.lua')
6592
6593 local characters = Babel.characters
6594 local ranges = Babel.ranges
6595
6596 local DIR = node.id("dir")
6597
6598 local function dir_mark(head, from, to, outer)
6599   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6600   local d = node.new(DIR)
6601   d.dir = '+' .. dir
6602   node.insert_before(head, from, d)
6603   d = node.new(DIR)
6604   d.dir = '-' .. dir
6605   node.insert_after(head, to, d)
6606 end
6607
6608 function Babel.bidi(head, ispar)
6609   local first_n, last_n          -- first and last char with nums
6610   local last_es                  -- an auxiliary 'last' used with nums

```

```

6611 local first_d, last_d          -- first and last char in L/R block
6612 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):

```

6613 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6614 local strong_lr = (strong == 'l') and 'l' or 'r'
6615 local outer = strong
6616
6617 local new_dir = false
6618 local first_dir = false
6619 local inmath = false
6620
6621 local last_lr
6622
6623 local type_n = ''
6624
6625 for item in node.traverse(head) do
6626
6627   -- three cases: glyph, dir, otherwise
6628   if item.id == node.id'glyph'
6629     or (item.id == 7 and item.subtype == 2) then
6630
6631     local itemchar
6632     if item.id == 7 and item.subtype == 2 then
6633       itemchar = item.replace.char
6634     else
6635       itemchar = item.char
6636     end
6637     local chardata = characters[itemchar]
6638     dir = chardata and chardata.d or nil
6639     if not dir then
6640       for nn, et in ipairs(ranges) do
6641         if itemchar < et[1] then
6642           break
6643         elseif itemchar <= et[2] then
6644           dir = et[3]
6645           break
6646         end
6647       end
6648     end
6649     dir = dir or 'l'
6650     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).

```

6651   if new_dir then
6652     attr_dir = 0
6653     for at in node.traverse(item.attr) do
6654       if at.number == Babel.attr_dir then
6655         attr_dir = at.value % 3
6656       end
6657     end
6658     if attr_dir == 1 then
6659       strong = 'r'

```

```

6660         elseif attr_dir == 2 then
6661             strong = 'al'
6662         else
6663             strong = 'l'
6664         end
6665         strong_lr = (strong == 'l') and 'l' or 'r'
6666         outer = strong_lr
6667         new_dir = false
6668     end
6669
6670     if dir == 'nsm' then dir = strong end          -- W1

```

Numbers. The dual $\langle al \rangle / \langle r \rangle$ system for R is somewhat cumbersome.

```

6671     dir_real = dir          -- We need dir_real to set strong below
6672     if dir == 'al' then dir = 'r' end -- W3

```

By W2, there are no $\langle en \rangle$ $\langle et \rangle$ $\langle es \rangle$ if strong == $\langle al \rangle$, only $\langle an \rangle$. Therefore, there are not $\langle et en \rangle$ nor $\langle en et \rangle$, W5 can be ignored, and W6 applied:

```

6673     if strong == 'al' then
6674         if dir == 'en' then dir = 'an' end          -- W2
6675         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6676         strong_lr = 'r'          -- W3
6677     end

```

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

```

6678     elseif item.id == node.id'dir' and not inmath then
6679         new_dir = true
6680         dir = nil
6681     elseif item.id == node.id'math' then
6682         inmath = (item.subtype == 0)
6683     else
6684         dir = nil          -- Not a char
6685     end

```

Numbers in R mode. A sequence of $\langle en \rangle$, $\langle et \rangle$, $\langle an \rangle$, $\langle es \rangle$ and $\langle cs \rangle$ 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 $\langle an \rangle$ is relevant if $\langle al \rangle$.

```

6686     if dir == 'en' or dir == 'an' or dir == 'et' then
6687         if dir ~= 'et' then
6688             type_n = dir
6689         end
6690         first_n = first_n or item
6691         last_n = last_es or item
6692         last_es = nil
6693     elseif dir == 'es' and last_n then -- W3+W6
6694         last_es = item
6695     elseif dir == 'cs' then          -- it's right - do nothing
6696     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6697         if strong_lr == 'r' and type_n ~= '' then
6698             dir_mark(head, first_n, last_n, 'r')
6699         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6700             dir_mark(head, first_n, last_n, 'r')
6701             dir_mark(head, first_d, last_d, outer)
6702             first_d, last_d = nil, nil
6703         elseif strong_lr == 'l' and type_n ~= '' then
6704             last_d = last_n
6705         end

```

```

6706     type_n = ''
6707     first_n, last_n = nil, nil
6708 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:

```

6709     if dir == 'l' or dir == 'r' then
6710         if dir ~= outer then
6711             first_d = first_d or item
6712             last_d = item
6713         elseif first_d and dir ~= strong_lr then
6714             dir_mark(head, first_d, last_d, outer)
6715             first_d, last_d = nil, nil
6716         end
6717     end

```

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

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

```

6718     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6719         item.char = characters[item.char] and
6720             characters[item.char].m or item.char
6721     elseif (dir or new_dir) and last_lr ~= item then
6722         local mir = outer .. strong_lr .. (dir or outer)
6723         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6724             for ch in node.traverse(node.next(last_lr)) do
6725                 if ch == item then break end
6726                 if ch.id == node.id'glyph' and characters[ch.char] then
6727                     ch.char = characters[ch.char].m or ch.char
6728                 end
6729             end
6730         end
6731     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).

```

6732     if dir == 'l' or dir == 'r' then
6733         last_lr = item
6734         strong = dir_real          -- Don't search back - best save now
6735         strong_lr = (strong == 'l') and 'l' or 'r'
6736     elseif new_dir then
6737         last_lr = nil
6738     end
6739 end

```

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

```

6740     if last_lr and outer == 'r' then
6741         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6742             if characters[ch.char] then
6743                 ch.char = characters[ch.char].m or ch.char
6744             end
6745         end
6746     end
6747     if first_n then
6748         dir_mark(head, first_n, last_n, outer)

```

```

6749 end
6750 if first_d then
6751     dir_mark(head, first_d, last_d, outer)
6752 end

```

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

```

6753 return node.prev(head) or head
6754 end
6755 </basic-r>

```

And here the Lua code for bidi=basic:

```

6756 <(*basic)
6757 Babel = Babel or {}
6758
6759 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6760
6761 Babel.fontmap = Babel.fontmap or {}
6762 Babel.fontmap[0] = {}      -- l
6763 Babel.fontmap[1] = {}      -- r
6764 Babel.fontmap[2] = {}      -- al/an
6765
6766 Babel.bidi_enabled = true
6767 Babel.mirroring_enabled = true
6768
6769 require('babel-data-bidi.lua')
6770
6771 local characters = Babel.characters
6772 local ranges = Babel.ranges
6773
6774 local DIR = node.id('dir')
6775 local GLYPH = node.id('glyph')
6776
6777 local function insert_implicit(head, state, outer)
6778     local new_state = state
6779     if state.sim and state.eim and state.sim ~= state.eim then
6780         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6781         local d = node.new(DIR)
6782         d.dir = '+' .. dir
6783         node.insert_before(head, state.sim, d)
6784         local d = node.new(DIR)
6785         d.dir = '-' .. dir
6786         node.insert_after(head, state.eim, d)
6787     end
6788     new_state.sim, new_state.eim = nil, nil
6789     return head, new_state
6790 end
6791
6792 local function insert_numeric(head, state)
6793     local new
6794     local new_state = state
6795     if state.san and state.ean and state.san ~= state.ean then
6796         local d = node.new(DIR)
6797         d.dir = '+TLT'
6798         _, new = node.insert_before(head, state.san, d)
6799         if state.san == state.sim then state.sim = new end
6800         local d = node.new(DIR)
6801         d.dir = '-TLT'
6802         _, new = node.insert_after(head, state.ean, d)

```



```

6803     if state.ean == state.eim then state.eim = new end
6804 end
6805 new_state.san, new_state.ean = nil, nil
6806 return head, new_state
6807 end
6808
6809 -- TODO - \hbox with an explicit dir can lead to wrong results
6810 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6811 -- was s made to improve the situation, but the problem is the 3-dir
6812 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6813 -- well.
6814
6815 function Babel.bidi(head, ispar, hdir)
6816     local d    -- d is used mainly for computations in a loop
6817     local prev_d = ''
6818     local new_d = false
6819
6820     local nodes = {}
6821     local outer_first = nil
6822     local inmath = false
6823
6824     local glue_d = nil
6825     local glue_i = nil
6826
6827     local has_en = false
6828     local first_et = nil
6829
6830     local ATDIR = Babel.attr_dir
6831
6832     local save_outer
6833     local temp = node.get_attribute(head, ATDIR)
6834     if temp then
6835         temp = temp % 3
6836         save_outer = (temp == 0 and 'l') or
6837                     (temp == 1 and 'r') or
6838                     (temp == 2 and 'al')
6839     elseif ispar then -- Or error? Shouldn't happen
6840         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6841     else -- Or error? Shouldn't happen
6842         save_outer = ('TRT' == hdir) and 'r' or 'l'
6843     end
6844     -- when the callback is called, we are just _after_ the box,
6845     -- and the textdir is that of the surrounding text
6846     -- if not ispar and hdir ~= tex.textdir then
6847     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6848     -- end
6849     local outer = save_outer
6850     local last = outer
6851     -- 'al' is only taken into account in the first, current loop
6852     if save_outer == 'al' then save_outer = 'r' end
6853
6854     local fontmap = Babel.fontmap
6855
6856     for item in node.traverse(head) do
6857
6858         -- In what follows, #node is the last (previous) node, because the
6859         -- current one is not added until we start processing the neutrals.
6860
6861         -- three cases: glyph, dir, otherwise

```

```

6862 if item.id == GLYPH
6863     or (item.id == 7 and item.subtype == 2) then
6864
6865     local d_font = nil
6866     local item_r
6867     if item.id == 7 and item.subtype == 2 then
6868         item_r = item.replace    -- automatic discs have just 1 glyph
6869     else
6870         item_r = item
6871     end
6872     local chardata = characters[item_r.char]
6873     d = chardata and chardata.d or nil
6874     if not d or d == 'nsm' then
6875         for nn, et in ipairs(ranges) do
6876             if item_r.char < et[1] then
6877                 break
6878             elseif item_r.char <= et[2] then
6879                 if not d then d = et[3]
6880                 elseif d == 'nsm' then d_font = et[3]
6881                 end
6882                 break
6883             end
6884         end
6885     end
6886     d = d or 'l'
6887
6888     -- A short 'pause' in bidi for mapfont
6889     d_font = d_font or d
6890     d_font = (d_font == 'l' and 0) or
6891             (d_font == 'nsm' and 0) or
6892             (d_font == 'r' and 1) or
6893             (d_font == 'al' and 2) or
6894             (d_font == 'an' and 2) or nil
6895     if d_font and fontmap and fontmap[d_font][item_r.font] then
6896         item_r.font = fontmap[d_font][item_r.font]
6897     end
6898
6899     if new_d then
6900         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6901         if inmath then
6902             attr_d = 0
6903         else
6904             attr_d = node.get_attribute(item, ATDIR)
6905             attr_d = attr_d % 3
6906         end
6907         if attr_d == 1 then
6908             outer_first = 'r'
6909             last = 'r'
6910         elseif attr_d == 2 then
6911             outer_first = 'r'
6912             last = 'al'
6913         else
6914             outer_first = 'l'
6915             last = 'l'
6916         end
6917         outer = last
6918         has_en = false
6919         first_et = nil
6920         new_d = false

```

```

6921     end
6922
6923     if glue_d then
6924         if (d == 'l' and 'l' or 'r') ~= glue_d then
6925             table.insert(nodes, {glue_i, 'on', nil})
6926         end
6927         glue_d = nil
6928         glue_i = nil
6929     end
6930
6931     elseif item.id == DIR then
6932         d = nil
6933         new_d = true
6934
6935     elseif item.id == node.id'glue' and item.subtype == 13 then
6936         glue_d = d
6937         glue_i = item
6938         d = nil
6939
6940     elseif item.id == node.id'math' then
6941         inmath = (item.subtype == 0)
6942
6943     else
6944         d = nil
6945     end
6946
6947     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6948     if last == 'al' and d == 'en' then
6949         d = 'an'          -- W3
6950     elseif last == 'al' and (d == 'et' or d == 'es') then
6951         d = 'on'          -- W6
6952     end
6953
6954     -- EN + CS/ES + EN      -- W4
6955     if d == 'en' and #nodes >= 2 then
6956         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6957             and nodes[#nodes-1][2] == 'en' then
6958             nodes[#nodes][2] = 'en'
6959         end
6960     end
6961
6962     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6963     if d == 'an' and #nodes >= 2 then
6964         if (nodes[#nodes][2] == 'cs')
6965             and nodes[#nodes-1][2] == 'an' then
6966             nodes[#nodes][2] = 'an'
6967         end
6968     end
6969
6970     -- ET/EN                  -- W5 + W7->1 / W6->on
6971     if d == 'et' then
6972         first_et = first_et or (#nodes + 1)
6973     elseif d == 'en' then
6974         has_en = true
6975         first_et = first_et or (#nodes + 1)
6976     elseif first_et then      -- d may be nil here !
6977         if has_en then
6978             if last == 'l' then
6979                 temp = 'l'    -- W7

```

```

6980         else
6981             temp = 'en'    -- W5
6982         end
6983     else
6984         temp = 'on'        -- W6
6985     end
6986     for e = first_et, #nodes do
6987         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6988     end
6989     first_et = nil
6990     has_en = false
6991 end
6992
6993 -- Force mathdir in math if ON (currently works as expected only
6994 -- with 'l')
6995 if inmath and d == 'on' then
6996     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6997 end
6998
6999 if d then
7000     if d == 'al' then
7001         d = 'r'
7002         last = 'al'
7003     elseif d == 'l' or d == 'r' then
7004         last = d
7005     end
7006     prev_d = d
7007     table.insert(nodes, {item, d, outer_first})
7008 end
7009
7010 outer_first = nil
7011
7012 end
7013
7014 -- TODO -- repeated here in case EN/ET is the last node. Find a
7015 -- better way of doing things:
7016 if first_et then    -- dir may be nil here !
7017     if has_en then
7018         if last == 'l' then
7019             temp = 'l'    -- W7
7020         else
7021             temp = 'en'    -- W5
7022         end
7023     else
7024         temp = 'on'        -- W6
7025     end
7026     for e = first_et, #nodes do
7027         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7028     end
7029 end
7030
7031 -- dummy node, to close things
7032 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7033
7034 ----- NEUTRAL -----
7035
7036 outer = save_outer
7037 last = outer
7038

```

```

7039 local first_on = nil
7040
7041 for q = 1, #nodes do
7042     local item
7043
7044     local outer_first = nodes[q][3]
7045     outer = outer_first or outer
7046     last = outer_first or last
7047
7048     local d = nodes[q][2]
7049     if d == 'an' or d == 'en' then d = 'r' end
7050     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7051
7052     if d == 'on' then
7053         first_on = first_on or q
7054     elseif first_on then
7055         if last == d then
7056             temp = d
7057         else
7058             temp = outer
7059         end
7060         for r = first_on, q - 1 do
7061             nodes[r][2] = temp
7062             item = nodes[r][1] -- MIRRORING
7063             if Babel.mirroring_enabled and item.id == GLYPH
7064                 and temp == 'r' and characters[item.char] then
7065                 local font_mode = font.fonts[item.font].properties.mode
7066                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7067                     item.char = characters[item.char].m or item.char
7068                 end
7069             end
7070         end
7071         first_on = nil
7072     end
7073
7074     if d == 'r' or d == 'l' then last = d end
7075 end
7076
7077 ----- IMPLICIT, REORDER -----
7078
7079 outer = save_outer
7080 last = outer
7081
7082 local state = {}
7083 state.has_r = false
7084
7085 for q = 1, #nodes do
7086
7087     local item = nodes[q][1]
7088
7089     outer = nodes[q][3] or outer
7090
7091     local d = nodes[q][2]
7092
7093     if d == 'nsm' then d = last end -- W1
7094     if d == 'en' then d = 'an' end
7095     local isdir = (d == 'r' or d == 'l')
7096
7097     if outer == 'l' and d == 'an' then

```

```

7098     state.san = state.san or item
7099     state.ean = item
7100 elseif state.san then
7101     head, state = insert_numeric(head, state)
7102 end
7103
7104 if outer == 'l' then
7105     if d == 'an' or d == 'r' then      -- im -> implicit
7106         if d == 'r' then state.has_r = true end
7107         state.sim = state.sim or item
7108         state.eim = item
7109     elseif d == 'l' and state.sim and state.has_r then
7110         head, state = insert_implicit(head, state, outer)
7111     elseif d == 'l' then
7112         state.sim, state.eim, state.has_r = nil, nil, false
7113     end
7114 else
7115     if d == 'an' or d == 'l' then
7116         if nodes[q][3] then -- nil except after an explicit dir
7117             state.sim = item -- so we move sim 'inside' the group
7118         else
7119             state.sim = state.sim or item
7120         end
7121         state.eim = item
7122     elseif d == 'r' and state.sim then
7123         head, state = insert_implicit(head, state, outer)
7124     elseif d == 'r' then
7125         state.sim, state.eim = nil, nil
7126     end
7127 end
7128
7129 if isdir then
7130     last = d          -- Don't search back - best save now
7131 elseif d == 'on' and state.san then
7132     state.san = state.san or item
7133     state.ean = item
7134 end
7135
7136 end
7137
7138 return node.prev(head) or head
7139 end
7140 </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.

```
7141 ⟨*nil⟩
7142 \ProvidesLanguage{nil}[\⟨⟨date⟩⟩ \⟨⟨version⟩⟩ Nil language]
7143 \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.

```
7144 \ifx\l@nil\undefined
7145   \newlanguage\l@nil
7146   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7147   \let\bbl@elt\relax
7148   \edef\bbl@languages{% Add it to the list of languages
7149     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7150 \fi
```

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

```
7151 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7152 \let\captionnil\@empty
7153 \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.

```
7154 \ldf@finish{nil}
7155 ⟨/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`.

```
7156 ⟨*bplain | blplain⟩
7157 \catcode`\{=1 % left brace is begin-group character
7158 \catcode`\}=2 % right brace is end-group character
7159 \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).

```
7160 \openin 0 hyphen.cfg
7161 \ifeof0
7162 \else
7163   \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.

```
7164   \def\input #1 {%
7165     \let\input\input
7166     \a hyphen.cfg
7167     \let\input\undefined
7168   }
7169 \fi
7170 </bplain | bplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
7171 <bplain>\a plain.tex
7172 <bplain>\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
7173 <bplain>\def\fmtname{babel-plain}
7174 <bplain>\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`.

```
7175 <<*Emulate LaTeX>> ≡
7176 % == Code for plain ==
7177 \def\@empty{}
7178 \def\loadlocalcfg#1{%
7179   \openin0#1.cfg
7180   \ifeof0
7181     \closein0
7182   \else
7183     \closein0
7184     {\immediate\write16{*****}%
7185      \immediate\write16{* Local config file #1.cfg used}%
7186      \immediate\write16{*}%
7187     }
7188     \input #1.cfg\relax
7189   \fi
7190   \@endofldf}
```

16.3 General tools

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

```
7191 \long\def\@firstofone#1{#1}
7192 \long\def\@firstoftwo#1#2{#1}
7193 \long\def\@secondoftwo#1#2{#2}
7194 \def\@nnil{\@nil}
```



```

7195 \def\@gobbletwo#1#2{}
7196 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7197 \def\@star@or@long#1{%
7198   \@ifstar
7199   {\let\l@ngrel@x\relax#1}%
7200   {\let\l@ngrel@x\long#1}}
7201 \let\l@ngrel@x\relax
7202 \def\@car#1#2\@nil{#1}
7203 \def\@cdr#1#2\@nil{#2}
7204 \let\@typeset@protect\relax
7205 \let\protected@edef\edef
7206 \long\def\@gobble#1{}
7207 \edef\@backslashchar{\expandafter\@gobble\string\}
7208 \def\strip@prefix#1>{}
7209 \def\g@addto@macro#1#2{%
7210   \toks@\expandafter{#1#2}%
7211   \xdef#1{\the\toks@}}
7212 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7213 \def\@nameuse#1{\csname #1\endcsname}
7214 \def\@ifundefined#1{%
7215   \expandafter\ifx\csname#1\endcsname\relax
7216     \expandafter\@firstoftwo
7217   \else
7218     \expandafter\@secondoftwo
7219   \fi}
7220 \def\@expandtwoargs#1#2#3{%
7221   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7222 \def\zap@space#1 #2{%
7223   #1%
7224   \ifx#2\@empty\else\expandafter\zap@space\fi
7225   #2}
7226 \let\bbl@trace\@gobble

```

$\text{\LaTeX}_2\epsilon$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

7227 \ifx\@preamblecmds\@undefined
7228   \def\@preamblecmds{}
7229 \fi
7230 \def\@onlypreamble#1{%
7231   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7232     \@preamblecmds\do#1}}
7233 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

7234 \def\begin{document}{%
7235   \@begin{document}hook
7236   \global\let\@begin{document}hook\@undefined
7237   \def\do##1{\global\let##1\@undefined}%
7238   \@preamblecmds
7239   \global\let\do\noexpand}
7240 \ifx\@begin{document}hook\@undefined
7241   \def\@begin{document}hook{}
7242 \fi
7243 \@onlypreamble\@begin{document}hook
7244 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

7245 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}

```

```

7246 \@onlypreamble\AtEndOfPackage
7247 \def\@endofldf{}
7248 \@onlypreamble\@endofldf
7249 \let\bbl@afterlang\@empty
7250 \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.

```

7251 \catcode`\&=\z@
7252 \ifx&\if@files\@undefined
7253   \expandafter\let\csname if@files\expandafter\endcsname
7254     \csname iffalse\endcsname
7255 \fi
7256 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7257 \def\newcommand{\@star@or@long\new@command}
7258 \def\new@command#1{%
7259   \@testopt{\@newcommand#1}0}
7260 \def\@newcommand#1[#2]{%
7261   \@ifnextchar [{\@xargdef#1[#2]}%
7262     {\@argdef#1[#2]}}
7263 \long\def\@argdef#1[#2]#3{%
7264   \@yargdef#1\@ne{#2}{#3}}
7265 \long\def\@xargdef#1[#2][#3]#4{%
7266   \expandafter\def\expandafter#1\expandafter{%
7267     \expandafter\@protected@testopt\expandafter #1%
7268     \csname\string#1\expandafter\endcsname{#3}}%
7269   \expandafter\@yargdef \csname\string#1\endcsname
7270   \tw@{#2}{#4}}
7271 \long\def\@yargdef#1#2#3{%
7272   \@tempcnta#3\relax
7273   \advance \@tempcnta \@ne
7274   \let\@hash@\relax
7275   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7276   \@tempcntb #2%
7277   \@whilenum \@tempcntb < \@tempcnta
7278   \do{%
7279     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7280     \advance\@tempcntb \@ne}%
7281   \let\@hash@###
7282   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
7283 \def\providecommand{\@star@or@long\provide@command}
7284 \def\provide@command#1{%
7285   \begingroup
7286     \escapechar\m@ne\xdef\@gtempa{\string#1}%
7287   \endgroup
7288   \expandafter\@ifundefined\@gtempa
7289     {\def\reserved@a{\new@command#1}}%
7290     {\let\reserved@a\relax
7291     \def\reserved@a{\new@command\reserved@a}}%
7292   \reserved@a}%
7293 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7294 \def\declare@robustcommand#1{%
7295   \edef\reserved@a{\string#1}%
7296   \def\reserved@b{#1}%
7297   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7298   \edef#1{%

```

```

7299 \ifx\reserved@a\reserved@b
7300 \noexpand\x@protect
7301 \noexpand#1%
7302 \fi
7303 \noexpand\protect
7304 \expandafter\noexpand\csname
7305 \expandafter\@gobble\string#1 \endcsname
7306 }%
7307 \expandafter\new@command\csname
7308 \expandafter\@gobble\string#1 \endcsname
7309 }
7310 \def\x@protect#1{%
7311 \ifx\protect\@typeset@protect\else
7312 \x@protect#1%
7313 \fi
7314 }
7315 \catcode\&=\z@ % Trick to hide conditionals
7316 \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`.

```

7317 \def\bbl@tempa{\csname newif\endcsname&fin@}
7318 \catcode\&=4
7319 \ifx\in@\@undefined
7320 \def\in@#1#2{%
7321 \def\in@@##1#1##2##3\in@{%
7322 \ifx\in@@##2\in@false\else\in@true\fi}%
7323 \in@@#2#1\in@\in@@}
7324 \else
7325 \let\bbl@tempa\@empty
7326 \fi
7327 \bbl@tempa

```

\LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain \TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

7328 \def\@ifpackagewith#1#2#3#4{#3}

```

The \LaTeX macro `\@ifloaded` 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.

```

7329 \def\@ifloaded#1#2#3#4{}

```

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

```

7330 \ifx\@tempcnta\@undefined
7331 \csname newcount\endcsname\@tempcnta\relax
7332 \fi
7333 \ifx\@tempcntb\@undefined
7334 \csname newcount\endcsname\@tempcntb\relax
7335 \fi

```

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

```

7336 \ifx\bye\@undefined
7337 \advance\count10 by -2\relax
7338 \fi

```

```

7339 \ifx\@ifnextchar\@undefined
7340 \def\@ifnextchar#1#2#3{%
7341   \let\reserved@d=#1%
7342   \def\reserved@a{#2}\def\reserved@b{#3}%
7343   \futurelet\@let@token\@ifnch}
7344 \def\@ifnch{%
7345   \ifx\@let@token\@sptoken
7346     \let\reserved@c\@xifnch
7347   \else
7348     \ifx\@let@token\reserved@d
7349       \let\reserved@c\reserved@a
7350     \else
7351       \let\reserved@c\reserved@b
7352     \fi
7353   \fi
7354   \reserved@c}
7355 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
7356 \def\:\@xifnch \expandafter\def\:{\futurelet\@let@token\@ifnch}
7357 \fi
7358 \def\@testopt#1#2{%
7359   \@ifnextchar[#{1}{#1[#2]}}
7360 \def\@protected@testopt#1{%
7361   \ifx\protect\@typeset@protect
7362     \expandafter\@testopt
7363   \else
7364     \@x@protect#1%
7365   \fi}
7366 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7367   #2\relax}\fi}
7368 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7369   \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

7370 \def\DeclareTextCommand{%
7371   \@dec@text@cmd\providecommand
7372 }
7373 \def\ProvideTextCommand{%
7374   \@dec@text@cmd\providecommand
7375 }
7376 \def\DeclareTextSymbol#1#2#3{%
7377   \@dec@text@cmd\chardef#1{#2}#3\relax
7378 }
7379 \def\@dec@text@cmd#1#2#3{%
7380   \expandafter\def\expandafter#2%
7381     \expandafter{%
7382       \csname#3-cmd\expandafter\endcsname
7383       \expandafter#2%
7384       \csname#3\string#2\endcsname
7385     }%
7386 %   \let\@ifdefinable\@rc@ifdefinable
7387   \expandafter#1\csname#3\string#2\endcsname
7388 }
7389 \def\@current@cmd#1{%
7390   \ifx\protect\@typeset@protect\else
7391     \noexpand#1\expandafter\@gobble
7392   \fi

```

```

7393 }
7394 \def\@changed@cmd#1#2{%
7395   \ifx\protect\@typeset@protect
7396     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7397       \expandafter\ifx\csname ?\string#1\endcsname\relax
7398         \expandafter\def\csname ?\string#1\endcsname{%
7399           \@changed@x@err{#1}%
7400         }%
7401       \fi
7402     \global\expandafter\let
7403       \csname\cf@encoding \string#1\expandafter\endcsname
7404       \csname ?\string#1\endcsname
7405     \fi
7406     \csname\cf@encoding\string#1%
7407     \expandafter\endcsname
7408   \else
7409     \noexpand#1%
7410   \fi
7411 }
7412 \def\@changed@x@err#1{%
7413   \errhelp{Your command will be ignored, type <return> to proceed}%
7414   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7415 \def\DeclareTextCommandDefault#1{%
7416   \DeclareTextCommand#1?%
7417 }
7418 \def\ProvideTextCommandDefault#1{%
7419   \ProvideTextCommand#1?%
7420 }
7421 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7422 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7423 \def\DeclareTextAccent#1#2#3{%
7424   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7425 }
7426 \def\DeclareTextCompositeCommand#1#2#3#4{%
7427   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7428   \edef\reserved@b{\string##1}%
7429   \edef\reserved@c{%
7430     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7431   \ifx\reserved@b\reserved@c
7432     \expandafter\expandafter\expandafter\ifx
7433       \expandafter\@car\reserved@a\relax\relax\@nil
7434       \@text@composite
7435   \else
7436     \edef\reserved@b##1{%
7437       \def\expandafter\noexpand
7438         \csname#2\string#1\endcsname####1{%
7439           \noexpand\@text@composite
7440           \expandafter\noexpand\csname#2\string#1\endcsname
7441           ####1\noexpand\@empty\noexpand\@text@composite
7442           {##1}%
7443         }%
7444       }%
7445     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7446   \fi
7447   \expandafter\def\csname\expandafter\string\csname
7448     #2\endcsname\string#1-\string#3\endcsname{#4}
7449 \else
7450   \errhelp{Your command will be ignored, type <return> to proceed}%
7451   \errmessage{\string\DeclareTextCompositeCommand\space used on

```

```

7452      inappropriate command \protect#1}
7453    \fi
7454  }
7455  \def\@text@composite#1#2#3\@text@composite{%
7456    \expandafter\@text@composite@x
7457      \csname\string#1-\string#2\endcsname
7458  }
7459  \def\@text@composite@x#1#2{%
7460    \ifx#1\relax
7461      #2%
7462    \else
7463      #1%
7464    \fi
7465  }
7466  %
7467  \def\@strip@args#1:#2-#3\@strip@args{#2}
7468  \def\DeclareTextComposite#1#2#3#4{%
7469    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7470    \bgroup
7471      \lccode`\@=#4%
7472      \lowercase{%
7473    \egroup
7474      \reserved@a @%
7475    }%
7476  }
7477  %
7478  \def\UseTextSymbol#1#2{#2}
7479  \def\UseTextAccent#1#2#3{
7480  \def\@use@text@encoding#1{
7481  \def\DeclareTextSymbolDefault#1#2{%
7482    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7483  }
7484  \def\DeclareTextAccentDefault#1#2{%
7485    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7486  }
7487  \def\cf@encoding{OT1}

```

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

```

7488 \DeclareTextAccent{"}{OT1}{127}
7489 \DeclareTextAccent{'}{OT1}{19}
7490 \DeclareTextAccent{^}{OT1}{94}
7491 \DeclareTextAccent{`}{OT1}{18}
7492 \DeclareTextAccent{~}{OT1}{126}

```

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

```

7493 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7494 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7495 \DeclareTextSymbol{\textquoteleft}{OT1}{``'}
7496 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
7497 \DeclareTextSymbol{\i}{OT1}{16}
7498 \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`.

```

7499 \ifx\scriptsize\undefined
7500   \let\scriptsize\sevenrm
7501 \fi
7502 % End of code for plain
7503 <</Emulate LaTeX>>

```

A proxy file:
7504 \langle *plain \rangle
7505 \input babel.def
7506 \langle /plain \rangle

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).