

Babel

Version 3.60
2021/06/02

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	8
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	15
1.12	The base option	17
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	33
1.18	Dates	34
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Transforms	38
1.22	Selection based on BCP 47 tags	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	48
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	49
1.31	Current and future work	50
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	51
3	The interface between the core of babel and the language definition files	52
3.1	Guidelines for contributed languages	53
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	58
4	Changes	61
4.1	Changes in babel version 3.9	61

II	Source code	62
5	Identification and loading of required files	62
6	locale directory	62
7	Tools	63
7.1	Multiple languages	67
7.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	67
7.3	base	69
7.4	Conditional loading of shorthands	72
7.5	Cross referencing macros	73
7.6	Marks	76
7.7	Preventing clashes with other packages	77
7.7.1	ifthen	77
7.7.2	varioref	77
7.7.3	hhline	78
7.7.4	hyperref	78
7.7.5	fancyhdr	78
7.8	Encoding and fonts	79
7.9	Basic bidi support	80
7.10	Local Language Configuration	86
8	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	90
8.1	Tools	90
9	Multiple languages	91
9.1	Selecting the language	93
9.2	Errors	102
9.3	Hooks	105
9.4	Setting up language files	106
9.5	Shorthands	108
9.6	Language attributes	118
9.7	Support for saving macro definitions	120
9.8	Short tags	121
9.9	Hyphens	121
9.10	Multiencoding strings	123
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	133
9.13	Layout	134
9.14	Load engine specific macros	135
9.15	Creating and modifying languages	135
10	Adjusting the Babel behavior	155
11	Loading hyphenation patterns	157
12	Font handling with <code>fontspec</code>	162

13	Hooks for XeTeX and LuaTeX	166
13.1	XeTeX	166
13.2	Layout	168
13.3	LuaTeX	170
13.4	Southeast Asian scripts	176
13.5	CJK line breaking	177
13.6	Arabic justification	179
13.7	Common stuff	183
13.8	Automatic fonts and ids switching	183
13.9	Layout	197
13.10	Auto bidi with basic and basic-r	201
14	Data for CJK	211
15	The ‘nil’ language	212
16	Support for Plain T_EX (plain.def)	212
16.1	Not renaming hyphen.tex	212
16.2	Emulating some L _A T _E X features	213
16.3	General tools	214
16.4	Encoding related macros	217
17	Acknowledgements	220

Troubleshooting

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

Part I

User guide

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

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

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

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

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

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

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

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

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

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

PDF_{TEX}

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

EXAMPLE With x_{etex} and 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 `\babelfont`, if used.)

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

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

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

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

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

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

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

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

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

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

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

Spaces after the environment are ignored.

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

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

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a

line. However, by default it never complied with the documented behavior and it is just a version as environment of `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while other `language*` does not.

1.9 More on selection

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

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

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

WARNING There is a clear drawback to this feature, namely, the ‘prefix’ `\text...` is heavily overloaded in \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` $[\text{include}=\langle commands \rangle, \text{exclude}=\langle commands \rangle, \text{fontenc}=\langle encoding \rangle]\{\langle language \rangle\}$

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things; for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionary 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 `\kernbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

\shorthandoff `*{\<shorthands-list>}`

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

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

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

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

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

\usesshorthands `*{\<char>}`

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

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

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

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

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

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

User shorthands override language ones, which in turn override system shorthands.

Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

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

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

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

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

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

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

`\languageshorthands` $\langle language \rangle$

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

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

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

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

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

`\babelshorthand` $\langle shorthand \rangle$

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

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

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

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

Languages with no shorthands Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh

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

⁶Thanks to Enrico Gregorio

Languages with only " as defined shorthand character Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

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

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

\ifbabelshorthand {<character>}{<true>}{<false>}

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

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

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

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

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

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

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

1.11 Package options

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

KeepShorthandsActive	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
activeacute	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
activegrave	Same for `.
shorthands=	<p>$\langle char \rangle \langle char \rangle \dots$ off</p> <p>The only language shorthands activated are those given, like, eg:</p> <pre>\usepackage[esperanto,french,shorthands=:;!]{babel}</pre> <p>If ' is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \string (otherwise they will be expanded by \LaTeX before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.</p>
safe=	<p>none ref bib</p> <p>Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \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).</p>
math=	<p>active normal</p> <p>Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.</p>
config=	<p>$\langle file \rangle$</p> <p>Load $\langle file \rangle.cfg$ instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).</p>
main=	<p>$\langle language \rangle$</p> <p>Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.</p>
headfoot=	<p>$\langle language \rangle$</p> <p>By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.</p>

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

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

1.12 The base option

With this package option babel just loads some basic macros (those in switch.def), defines \AfterBabelLanguage and exits. It also selects the hyphenation patterns for the

⁸You can use alternatively the package silence.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

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

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

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

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

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

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

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

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

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

1.13 ini files

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

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

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

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

LUATEX/XETEX

```
\documentclass{book}

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

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

```

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

```

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

```

Or also:

```

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

```

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

```

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

```

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

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

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

```

\newfontscript{Devanagari}{deva}

```

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

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{lᦺ lᦴ lᦶ lᦸ lᦺ lᦴ lᦶ lᦸ} % Random
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Modifying and adding values to ini files

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

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

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

\babelprovide [*<options>*]{*<language-name>*}

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

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

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

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

import= *<language-tag>*

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

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

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

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

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

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

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

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

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

Loads only the strings. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remerber there is an alternative syntax for the latter:

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

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

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

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

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

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

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

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

Same for `\Alph`.

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

onchar= `ids` | `fonts`

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

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

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

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

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

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

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

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

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

mapfont= `direction`

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

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

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

1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

- `\localenumeral{<style>}{<number>}`, like `\localenumeral{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. Çileyay Pêşîn 2019*, but with `variant=iza fa` it prints *31'ê Çileyay Pêşînê 2019*.

1.19 Accessing language info

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

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

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

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

\localeinfo `{\field}`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.21 Transforms

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

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

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

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

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

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

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

Arabic, Persian	kashida.plain	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.
Serbian	transliteration.gajica	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.

\babelposthyphenation {<hyphenrules-name>}{<lua-pattern>}{<replacement>}

New 3.37-3.39 With *luatex* it is now possible to define non-standard hyphenation rules, like $f-f \rightarrow ff-f$, repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. Only a few rules are currently provided (see below), but they can be defined as shown in the following example, where {1} is the first captured char (between () in the pattern):

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

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads ([îû]), the replacement could be {1|îû|íú}, which maps î to í, and û to ú, so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`. See the [babel site](#) for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account).

You are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.

\babelprehyphenation {<locale-name>}{<lua-pattern>}{<replacement>}

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

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

```
\babelprovide[hyphenrules=+]{russian-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

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` `{⟨text⟩}`

New 3.9i This macro makes sure `⟨text⟩` 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

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

¹⁷But still defined for backwards compatibility.

for “ordinary” text (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}
```

```

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

    وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الغريقي) بـ
    Arabia أو Aravia (بالغريقية 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 \textit{fuṣḥā l-‘aṣr} (MSA) and
    \textit{fuṣḥā t-turāth} (CA).

\end{document}

```

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

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

```

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

```

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the `bidi` package, which

provides its own mechanism to control these elements). You may use several options with a dot-separated list (eg, `layout=counters.contents.sectioning`). This list will be expanded in future releases. Note not all options are required by all engines.

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection`), `\section`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it can depend on the counter format.

With counters, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while 1.2 in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is `c2.c1`. Of course, you may always adjust the order by changing the language, if necessary.¹⁸

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

WARNING As of April 2019 there is a bug with `\parshape` in `luatex` (a `TEX` primitive) which makes lists to be horizontally misplaced if they are inside a `\vbox` (like `minipage`) and the current direction is different from the main one. A workaround is to restore the main language before the box and then set the local one inside.

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

`\babelsublr` `{\lr-text}`

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.25 Language attributes

`\languageattribute`

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

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

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

1.26 Hooks

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

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

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{<name>}`, `\DisableBabelHook{<name>}`.

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.27 Languages supported by babel with ldf files

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

Afrikaans afrikaans

Azerbaijani azerbaijani

Basque basque

Breton breton

Bulgarian bulgarian

Catalan catalan

Croatian croatian

Czech czech

Danish danish

Dutch dutch

English english, USenglish, american, UKenglish, british, canadian, australian, newzealand

Esperanto esperanto

Estonian estonian

Finnish finnish

French french, francais, canadien, acadian

Galician galician

German austrian, german, germanb, ngerman, naustrian
Greek greek, polutonikogreek
Hebrew hebrew
Icelandic icelandic
Indonesian indonesian (bahasa, indon, bahasai)
Interlingua interlingua
Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu (bahasam)
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, brazilian (portuges, brazil)¹⁹
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
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).

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

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

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

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

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

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

1.29 Tweaking some features

\babeladjust $\langle key-value-list \rangle$

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

1.30 Tips, workarounds, known issues and notes

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

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

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because T_EX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of T_EX, not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, 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 no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make T_EX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

microtype Adjusts the typesetting according to some languages (kerning and spacing).

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²¹

But that is the easy part, because they don't require modifying the L^AT_EX internals.

Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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

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

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with `language.dat`

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

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

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:

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

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

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

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

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

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

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are

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

discussed below. You must define all or none for a language (or a dialect); defining, say, `\date⟨lang⟩` but not `\captions⟨lang⟩` does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

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

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

3.1 Guidelines for contributed languages

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

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

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

²⁶But not removed, for backward compatibility.

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

The following page provides a starting point for ldf files:

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

<https://github.com/latex3/babel/blob/master/news-guides/guides/list-of-locale-templates.md>.

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

\noextras<lang> Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of `\extras<lang>`, a macro that brings TeX into a

	predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras<lang></code> .
<code>\bbl@declare@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \TeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \TeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions<lang></code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

```

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

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

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

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

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

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

```

```

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

`\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, 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.

²⁷This mechanism was introduced by Bernd Raichle.

3.8 Encoding-dependent strings

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

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

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

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

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

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

If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

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

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

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

The $\langle\text{category}\rangle$ 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]
```

²⁸In future releases further categories may be added.

```

\SetString{\chaptername}{utf8-string}

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

\EndBabelCommands

```

A real example is:

```

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

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

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

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

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

\EndBabelCommands Marks the end of the series of blocks.

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

\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.

\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 L^AT_EX, we can set for Turkish:

```
\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]  
\SetCase  
  {\uccode"10=`I\relax}  
  {\lccode`I="10\relax}  
  
\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]  
\SetCase  
  {\uccode`i=`İ\relax  
   \uccode`ı=`I\relax}  
  {\lccode`İ=`i\relax  
   \lccode`I=`ı\relax}  
  
\StartBabelCommands{turkish}{}  
\SetCase  
  {\uccode`i="9D\relax  
   \uccode"19=`I\relax}  
  {\lccode"9D=`i\relax  
   \lccode`I="19\relax}
```

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

```
\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

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

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

```
\SetHyphenMap{\BabelLowerMM{"100}{\lccode}{\lccode}{\lccode}}
```

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

4 Changes

4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop can happen. It worked incorrectly with `^` (if activated) and also if deactivated.

- Active chars were not reset at the end of language options, and that led to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with babel were not recognized when called as global options.

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the `ini` file.

version of the `ini` file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file `charset`

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

date.long fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMM 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=3.60>>
2 <<date=2021/06/02>>
```

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

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

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

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

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

`\bbl@afterelse` `\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}
```

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

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

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

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

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

```
68 \def\bbl@ifblank#1{%
```

```

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

```

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

```

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

```

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

```

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

```

\bbl@replace

```

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

```

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

```

105 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
106 \bbl@exp{\def\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax%
107 \def\bbl@tempa{#1}%
108 \def\bbl@tempb{#2}%
109 \def\bbl@tempe{#3}}
110 \def\bbl@sreplace#1#2#3{%
111 \begingroup

```

```

112 \expandafter\bb1@parsedef\meaning#1\relax
113 \def\bb1@tempc{#2}%
114 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
115 \def\bb1@tempd{#3}%
116 \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
117 \bb1@xin@{\bb1@tempc}{\bb1@tempe}% If not in macro, do nothing
118 \ifin@
119 \bb1@exp{\bb1@replace\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
120 \def\bb1@tempc{% Expanded an executed below as 'uplevel'
121 \\\makeatletter % "internal" macros with @ are assumed
122 \\\scantokens{%
123 \bb1@tempa\\@namedef{\bb1@stripslash#1}\bb1@tempb{\bb1@tempe}}%
124 \catcode64=\the\catcode64\relax}% Restore @
125 \else
126 \let\bb1@tempc\@empty % Not \relax
127 \fi
128 \bb1@exp{% For the 'uplevel' assignments
129 \endgroup
130 \bb1@tempc}} % empty or expand to set #1 with changes
131 \fi

```

Two further tools. `\bb1@samestring` first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). `\bb1@engine` takes the following values: 0 is pdf \TeX , 1 is luatex, and 2 is xetex. You may use the latter in your language style if you want.

```

132 \def\bb1@ifsamestring#1#2{%
133 \begingroup
134 \protected@edef\bb1@tempb{#1}%
135 \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
136 \protected@edef\bb1@tempc{#2}%
137 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
138 \ifx\bb1@tempb\bb1@tempc
139 \aftergroup\@firstoftwo
140 \else
141 \aftergroup\@secondoftwo
142 \fi
143 \endgroup}
144 \chardef\bb1@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\bb1@bsphack{%
155 \ifhmode
156 \hskip\z@skip
157 \def\bb1@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158 \else
159 \let\bb1@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\bb1@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}
173 <</Basic macros>>

```

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

```

174 <<*Make sure ProvidesFile is defined>> ≡
175 \ifx\ProvidesFile\@undefined
176   \def\ProvidesFile#1[#2 #3 #4]{%
177     \wlog{File: #1 #4 #3 <#2>}%
178     \let\ProvidesFile\@undefined}
179 \fi
180 <</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.

```

181 <<*Define core switching macros>> ≡
182 \ifx\language\@undefined
183   \csname newcount\endcsname\language
184 \fi
185 <</Define core switching macros>>

```

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

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

```

186 <<*Define core switching macros>> ≡
187 <<*Define core switching macros>> ≡
188 \countdef\last@language=19 % TODO. why? remove?
189 \def\addlanguage{\csname newlanguage\endcsname}
190 <</Define core switching macros>>

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or \LaTeX 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it). Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

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

This file also takes care of a number of compatibility issues with other packages and defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user. The first two options are for debugging.

```

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

```

```

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

```

273 \ifx\bbl@languages\undefined\else
274   \begingroup
275     \catcode\^^I=12
276     \@ifpackagewith{babel}{showlanguages}{%
277       \begingroup
278         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
279         \wlog{<*languages>}%
280         \bbl@languages
281         \wlog{</languages>}%
282       \endgroup}{%
283     \endgroup
284     \def\bbl@elt#1#2#3#4{%
285       \ifnum#2=\z@
286         \gdef\bbl@nulllanguage{#1}%
287         \def\bbl@elt##1##2##3##4{}}%
288     \fi}%
289 \bbl@languages
290 \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`.

```

291 \bbl@trace{Defining option 'base'}
292 \@ifpackagewith{babel}{base}{%

```

```

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

```

348 \DeclareOption{KeepShorthandsActive}{}
349 \DeclareOption{activeacute}{}
350 \DeclareOption{activegrave}{}
351 \DeclareOption{debug}{}
352 \DeclareOption{noconfigs}{}
353 \DeclareOption{showlanguages}{}
354 \DeclareOption{silent}{}
355 \DeclareOption{mono}{}
356 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
357 \chardef\bbl@iniflag\z@
358 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
359 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
360 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
361 % A separate option
362 \let\bbl@autoload@options\@empty
363 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
364 % Don't use. Experimental. TODO.
365 \newif\ifbbl@single
366 \DeclareOption{selectors=off}{\bbl@singletrue}
367 <<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.

```

368 \let\bbl@opt@shorthands\@nnil
369 \let\bbl@opt@config\@nnil
370 \let\bbl@opt@main\@nnil
371 \let\bbl@opt@headfoot\@nnil
372 \let\bbl@opt@layout\@nnil

```

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

```

373 \def\bbl@tempa#1=#2\bbl@tempa{%
374   \bbl@csarg\ifx{opt@#1}\@nnil
375     \bbl@csarg\edef{opt@#1}{#2}%
376   \else
377     \bbl@error
378     {Bad option '#1=#2'. Either you have misspelled the\\%
379     key or there is a previous setting of '#1'. Valid\\%
380     keys are, among others, 'shorthands', 'main', 'bidi',\\%
381     'strings', 'config', 'headfoot', 'safe', 'math'.}%
382     {See the manual for further details.}
383   \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.

```

384 \let\bbl@language@opts\@empty
385 \DeclareOption*{%
386   \bbl@xin@{\string=}{\CurrentOption}%
387   \ifin@
388     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
389   \else
390     \bbl@add@list\bbl@language@opts{\CurrentOption}%
391   \fi}

```

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

```

392 \ProcessOptions*

```

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=...`

```
393 \bbl@trace{Conditional loading of shorthands}
394 \def\bbl@sh@string#1{%
395   \ifx#1\@empty\else
396     \ifx#1t\string~%
397     \else\ifx#1c\string,%
398     \else\string#1%
399   \fi\fi
400   \expandafter\bbl@sh@string
401 \fi}
402 \ifx\bbl@opt@shorthands\@nnil
403   \def\bbl@ifshorthand#1#2#3{#2}%
404 \else\ifx\bbl@opt@shorthands\@empty
405   \def\bbl@ifshorthand#1#2#3{#3}%
406 \else
```

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

```
407 \def\bbl@ifshorthand#1{%
408   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
409   \ifin@
410     \expandafter\@firstoftwo
411   \else
412     \expandafter\@secondoftwo
413   \fi}
```

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

```
414 \edef\bbl@opt@shorthands{%
415   \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.

```
416 \bbl@ifshorthand{'}%
417   {\PassOptionsToPackage{activeacute}{babel}}{}
418 \bbl@ifshorthand{`}%
419   {\PassOptionsToPackage{activegrave}{babel}}{}
420 \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.

```
421 \ifx\bbl@opt@headfoot\@nnil\else
422   \g@addto@macro\@resetactivechars{%
423     \set@typeset@protect
424     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
425     \let\protect\noexpand}
426 \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.

```
427 \ifx\bbl@opt@safe\@undefined
428   \def\bbl@opt@safe{BR}
429 \fi
430 \ifx\bbl@opt@main\@nnil\else
431   \edef\bbl@language@opts{%

```

```

432 \ifx\bbl@language@opts@empty\else\bbl@language@opts,\fi
433 \bbl@opt@main}
434 \fi

```

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

```

435 \bbl@trace{Defining IfBabelLayout}
436 \ifx\bbl@opt@layout@nnil
437 \newcommand\IfBabelLayout[3]{#3}%
438 \else
439 \newcommand\IfBabelLayout[1]{%
440 \expandafter\in@{.#1.}{.\bbl@opt@layout.}%
441 \ifin@
442 \expandafter\@firstoftwo
443 \else
444 \expandafter\@secondoftwo
445 \fi}
446 \fi

```

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

```

447 \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.

```

448 << *More package options >> ≡
449 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
450 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
451 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
452 << /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.

```

453 \bbl@trace{Cross referencing macros}
454 \ifx\bbl@opt@safe\@empty\else
455 \def\@newl@bel#1#2#3{%
456 {\@safe@activestrue
457 \bbl@ifunset{#1@#2}%
458 \relax
459 {\gdef\@multiplelabels{%
460 \@latex@warning@no@line{There were multiply-defined labels}}%
461 \@latex@warning@no@line{Label `#2' multiply defined}}%
462 \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.

```

463 \CheckCommand*\@testdef[3]{%
464 \def\reserved@a{#3}%

```

```

465 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
466 \else
467 \@tempswatrue
468 \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.

```

469 \def\@testdef#1#2#3{% TODO. With @samestring?
470 \@safe@activestrue
471 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
472 \def\bbl@tempb{#3}%
473 \@safe@activetrue
474 \ifx\bbl@tempa\relax
475 \else
476 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
477 \fi
478 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
479 \ifx\bbl@tempa\bbl@tempb
480 \else
481 \@tempswatrue
482 \fi}
483 \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.

```

484 \bbl@xin@{R}\bbl@opt@safe
485 \ifin@
486 \bbl@redefineroobust\ref#1{%
487 \@safe@activestrue\org@ref{#1}\@safe@activetrue}
488 \bbl@redefineroobust\pageref#1{%
489 \@safe@activestrue\org@pageref{#1}\@safe@activetrue}
490 \else
491 \let\org@ref\ref
492 \let\org@pageref\pageref
493 \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.

```

494 \bbl@xin@{B}\bbl@opt@safe
495 \ifin@
496 \bbl@redefine\@citex[#1]#2{%
497 \@safe@activestrue\edef\@tempa{#2}\@safe@activetrue}
498 \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.

```

499 \AtBeginDocument{%
500 \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.)

```
501 \def\@citex[#1][#2]#3{%
502   \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
503   \org@@citex[#1][#2]{\@tempa}}%
504 }{}}
```

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

```
505 \AtBeginDocument{%
506   \@ifpackageloaded{cite}{%
507     \def\@citex[#1]#2{%
508       \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
509     }{}}
```

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

```
510 \bbl@redefine\nocite#1{%
511   \@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.

```
512 \bbl@redefine\bibcite{%
513   \bbl@cite@choice
514   \bibcite}
```

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

```
515 \def\bbl@bibcite#1#2{%
516   \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.

```
517 \def\bbl@cite@choice{%
518   \global\let\bibcite\bbl@bibcite
519   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
520   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
521   \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.

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

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

```
523 \bbl@redefine\@bibitem#1{%
524   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
525 \else
526   \let\org@nocite\nocite
527   \let\org@@citex\@citex
528   \let\org@bibcite\bibcite
529   \let\org@@bibitem\@bibitem
530 \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.

```

531 \bbl@trace{Marks}
532 \IfBabelLayout{sectioning}
533   {\ifx\bbl@opt@headfoot\@nnil
534     \g@addto@macro\@resetactivechars{%
535       \set@typeset@protect
536       \expandafter\select@language@\x\expandafter{\bbl@main@language}%
537       \let\protect\noexpand
538       \ifcase\bbl@bidimode\else % Only with bidi. See also above
539         \edef\thepage{%
540           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
541       \fi}%
542   \fi}
543 {\ifbbl@single\else
544   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
545   \markright#1{%
546     \bbl@ifblank{#1}%
547     {\org@markright{}}%
548     {\toks@{#1}%
549      \bbl@exp{%
550        \\org@markright{\protect\\foreignlanguage{\language}%
551          {\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`

```

552   \ifx\@mkboth\markboth
553     \def\bbl@tempc{\let\@mkboth\markboth}
554   \else
555     \def\bbl@tempc{}
556   \fi
557   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
558   \markboth#1#2{%
559     \protected@edef\bbl@tempb##1{%
560       \protect\foreignlanguage
561       {\language}{\protect\bbl@restore@actives##1}%
562     \bbl@ifblank{#1}%
563     {\toks@{}}%
564     {\toks@\expandafter{\bbl@tempb{#1}}}%
565     \bbl@ifblank{#2}%
566     {\@temptokena{}}%
567     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
568     \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
569     \bbl@tempc
570   \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.

```
571 \bbl@trace{Preventing clashes with other packages}
572 \bbl@xin@{R}\bbl@opt@safe
573 \ifin@
574 \AtBeginDocument{%
575   \@ifpackageloaded{ifthen}{%
576     \bbl@redefine@long\ifthenelse#1#2#3{%
577       \let\bbl@temp@pref\pageref
578       \let\pageref\org@pageref
579       \let\bbl@temp@ref\ref
580       \let\ref\org@ref
581       \@safe@activestrue
582       \org@ifthenelse{#1}%
583         {\let\pageref\bbl@temp@pref
584          \let\ref\bbl@temp@ref
585          \@safe@activesfalse
586          #2}%
587         {\let\pageref\bbl@temp@pref
588          \let\ref\bbl@temp@ref
589          \@safe@activesfalse
590          #3}%
591     }%
592   }{}%
593 }
```

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`.

```
594 \AtBeginDocument{%
595   \@ifpackageloaded{varioref}{%
596     \bbl@redefine\@@vpageref#1[#2]#3{%
597       \@safe@activestrue
598       \org@@@vpageref{#1}[#2]{#3}%
599       \@safe@activesfalse}%
600     \bbl@redefine\vrefpagemum#1#2{%
601       \@safe@activestrue
602       \org@vrefpagemum{#1}{#2}%
603       \@safe@activesfalse}%
604   }
```

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.

```
604 \expandafter\def\csname Ref \endcsname#1{%
605 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
606 }{}%
607 }
608 \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.

```
609 \AtEndOfPackage{%
610 \AtBeginDocument{%
611 \ifpackageloaded{hhline}%
612 {\expandafter\ifx\csname normal@char\string\endcsname\relax
613 \else
614 \makeatletter
615 \def\@currname{hhline}\input{hhline.sty}\makeatother
616 \fi}%
617 {}}}
```

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.

```
618 % \AtBeginDocument{%
619 % \ifx\pdfstringdefDisableCommands\@undefined\else
620 % \pdfstringdefDisableCommands{\languageshorthands{system}}%
621 % \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.

```
622 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
623 \lowercase{\foreignlanguage{#1}}}
```

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

```
624 \def\substitutefontfamily#1#2#3{%
625 \lowercase{\immediate\openout15=#1#2.fd\relax}%
626 \immediate\write15{%
627 \string\ProvidesFile{#1#2.fd}%
628 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
629 \space generated font description file]^^J
```

```

630 \string\DeclareFontFamily{#1}{#2}{}^^J
631 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
632 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
633 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
634 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
635 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
636 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
637 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
638 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
639 }%
640 \closeout15
641 }
642 \@onlypreamble\substitutefontfamily

```

7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing `\@filelist` to search for `<enc>enc.def`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

`\ensureascii`

```

643 \bbl@trace{Encoding and fonts}
644 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
645 \newcommand\BabelNonText{TS1,T3,TS3}
646 \let\org@TeX\TeX
647 \let\org@LaTeX\LaTeX
648 \let\ensureascii\@firstofone
649 \AtBeginDocument{%
650   \in@false
651   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
652     \ifin@%else
653       \lowercase{\bbl@xin@{,#1enc.def},{,\@filelist,}}%
654     \fi}%
655   \ifin@ % if a text non-ascii has been loaded
656     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
657     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
658     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
659     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
660     \def\bbl@tempc#1ENC.DEF#2\@@{%
661       \ifx\@empty#2%else
662         \bbl@ifunset{T@#1}%
663         {}%
664         {\bbl@xin@{,#1},{,\BabelNonASCII,\BabelNonText,}}%
665       \ifin@
666         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
667         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
668       \else
669         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
670       \fi}%
671     \fi}%
672   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
673   \bbl@xin@{,\cf@encoding,},{,\BabelNonASCII,\BabelNonText,}%
674   \ifin@%else
675     \edef\ensureascii#1{%
676       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}%
677   \fi

```

```
678 \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.

```
679 \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.

```
680 \AtBeginDocument{%
681   \ifpackageloaded{fontspec}%
682     {\xdef\latinencoding{%
683       \ifx\UTFencname\undefined
684         EU\ifcase\bb1@engine\or2\or1\fi
685       \else
686         \UTFencname
687       \fi}}%
688   {\gdef\latinencoding{OT1}%
689     \ifx\cf@encoding\bb1@t@one
690       \xdef\latinencoding{\bb1@t@one}%
691     \else
692       \ifx\@fontenc@load@list\undefined
693         \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bb1@t@one}}{}%
694       \else
695         \def\@elt#1{, #1,}%
696         \edef\bb1@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
697         \let\@elt\relax
698         \bb1@xin@{, T1, }\bb1@tempa
699         \ifin@
700           \xdef\latinencoding{\bb1@t@one}%
701         \fi
702       \fi
703     \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.

```
704 \DeclareRobustCommand{\latintext}{%
705   \fontencoding{\latinencoding}\selectfont
706   \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.

```
707 \ifx\undefined\DeclareTextFontCommand
708   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
709 \else
710   \DeclareTextFontCommand{\textlatin}{\latintext}
711 \fi
```

7.9 Basic bidi support

Work in progress. This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This babel module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents

for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at ARABI (by Youssef Jabri), which is compatible with babel.

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like `rlbabel` did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- `pdfTeX` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `LuaTeX-ja` shows, vertical typesetting is possible, too.

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

```

712 \ifodd\bbl@engine
713   \def\bbl@activate@preotf{%
714     \let\bbl@activate@preotf\relax % only once
715     \directlua{
716       Babel = Babel or {}
717       %
718       function Babel.pre_otfload_v(head)
719         if Babel.numbers and Babel.digits_mapped then
720           head = Babel.numbers(head)
721         end
722         if Babel.bidi_enabled then
723           head = Babel.bidi(head, false, dir)
724         end
725         return head
726       end
727       %
728       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
729         if Babel.numbers and Babel.digits_mapped then
730           head = Babel.numbers(head)
731         end
732         if Babel.bidi_enabled then
733           head = Babel.bidi(head, false, dir)
734         end
735         return head
736       end
737       %
738       luatexbase.add_to_callback('pre_linebreak_filter',
739         Babel.pre_otfload_v,
740         'Babel.pre_otfload_v',
741         luatexbase.priority_in_callback('pre_linebreak_filter',
742           'luaotfload.node_processor') or nil)
743       %
744       luatexbase.add_to_callback('hpack_filter',
745         Babel.pre_otfload_h,
746         'Babel.pre_otfload_h',
747         luatexbase.priority_in_callback('hpack_filter',
748           'luaotfload.node_processor') or nil)
749     }}
750 \fi

```

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

```

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

```

```

807 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
808 \fi
809 \AtEndOfPackage{%
810 \EnableBabelHook{babel-bidi}%
811 \ifodd\bbl@engine\else
812 \bbl@xebidipar
813 \fi}
814 \fi

```

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

```

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

```

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

```

857 \ifodd\bbl@engine % luatex=1
858 \chardef\bbl@thetextdir\z@
859 \chardef\bbl@thepardir\z@
860 \def\bbl@getluadir#1{%

```

```

861 \directlua{
862   if tex.#1dir == 'TLT' then
863     tex.sprint('0')
864   elseif tex.#1dir == 'TRT' then
865     tex.sprint('1')
866   end}}
867 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
868   \ifcase#3\relax
869     \ifcase\bbl@getluadir{#1}\relax\else
870       #2 TLT\relax
871     \fi
872   \else
873     \ifcase\bbl@getluadir{#1}\relax
874       #2 TRT\relax
875     \fi
876   \fi}
877 \def\bbl@textdir#1{%
878   \bbl@setluadir{text}\textdir{#1}%
879   \chardef\bbl@thetextdir#1\relax
880   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
881 \def\bbl@pardir#1{%
882   \bbl@setluadir{par}\pardir{#1}%
883   \chardef\bbl@thepardir#1\relax}
884 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
885 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
886 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
887 % Sadly, we have to deal with boxes in math with basic.
888 % Activated every math with the package option bidi=:
889 \ifnum\bbl@bidimode>\z@
890   \def\bbl@mathboxdir{%
891     \ifcase\bbl@thetextdir\relax
892       \everyhbox{\bbl@mathboxdir@aux L}%
893     \else
894       \everyhbox{\bbl@mathboxdir@aux R}%
895     \fi}
896   \def\bbl@mathboxdir@aux#1{%
897     \@ifnextchar\egroup{}\textdir T#1T\relax}}
898   \frozen@everymath\expandafter{%
899     \expandafter\bbl@mathboxdir\the\frozen@everymath}
900   \frozen@everydisplay\expandafter{%
901     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
902   \fi
903 \else % pdftex=0, xetex=2
904   \newcount\bbl@dirlevel
905   \chardef\bbl@thetextdir\z@
906   \chardef\bbl@thepardir\z@
907   \def\bbl@textdir#1{%
908     \ifcase#1\relax
909       \chardef\bbl@thetextdir\z@
910       \bbl@textdir@i\beginL\endL
911     \else
912       \chardef\bbl@thetextdir@ne
913       \bbl@textdir@i\beginR\endR
914     \fi}
915   \def\bbl@textdir@i#1#2{%
916     \ifhmode
917       \ifnum\currentgrouplevel>\z@
918         \ifnum\currentgrouplevel=\bbl@dirlevel
919           \bbl@error{Multiple bidi settings inside a group}%

```

```

920      {I'll insert a new group, but expect wrong results.}%
921      \bgroup\aftergroup#2\aftergroup\egroup
922    \else
923      \ifcase\currentgrouptype\or % 0 bottom
924        \aftergroup#2% 1 simple {}
925      \or
926        \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
927      \or
928        \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
929      \or\or\or % vbox vtop align
930      \or
931        \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
932      \or\or\or\or\or\or % output math disc insert vcent mathchoice
933      \or
934        \aftergroup#2% 14 \beginngroup
935      \else
936        \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
937      \fi
938    \fi
939    \bbl@dirlevel\currentgrouplevel
940  \fi
941  #1%
942  \fi}
943  \def\bbl@pdir#1{\chardef\bbl@thepardir#1\relax}
944  \let\bbl@bodydir\@gobble
945  \let\bbl@pagedir\@gobble
946  \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).

```

947  \def\bbl@xebidipar{%
948    \let\bbl@xebidipar\relax
949    \TeXeTstate\@ne
950  \def\bbl@xeeverypar{%
951    \ifcase\bbl@thepardir
952      \ifcase\bbl@thetextdir\else\beginR\fi
953    \else
954      {\setbox\z@\lastbox\beginR\box\z@}%
955    \fi}%
956  \let\bbl@severypar\everypar
957  \newtoks\everypar
958  \everypar=\bbl@severypar
959  \bbl@severypar{\bbl@xeeverypar\the\everypar}}
960  \ifnum\bbl@bidimode>200
961    \let\bbl@textdir@i\@gobbletwo
962    \let\bbl@xebidipar\@empty
963    \AddBabelHook{bidi}{foreign}{%
964      \def\bbl@tempa{\def\BabelText###1}%
965      \ifcase\bbl@thetextdir
966        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{###1}}}%
967      \else
968        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{###1}}}%
969      \fi}
970    \def\bbl@pdir#1{\ifcase#1\relax\setLR\else\setRL\fi}
971  \fi
972  \fi

```

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

```

973 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
974 \AtBeginDocument{%
975   \ifx\pdfstringdefDisableCommands\@undefined\else
976     \ifx\pdfstringdefDisableCommands\relax\else
977       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
978     \fi
979   \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`.

```

980 \bbl@trace{Local Language Configuration}
981 \ifx\loadlocalcfg\@undefined
982   \@ifpackagewith{babel}{noconfigs}%
983     {\let\loadlocalcfg\@gobble}%
984   {\def\loadlocalcfg#1{%
985     \InputIfFileExists{#1.cfg}%
986     {\typeout{*****^J%
987               * Local config file #1.cfg used^^J%
988               *}}%
989     \@empty}}
990 \fi

```

Just to be compatible with \LaTeX 2.09 we add a few more lines of code. TODO. Necessary? Correct place? Used by some ldf file?

```

991 \ifx\@unexpandable@protect\@undefined
992   \def\@unexpandable@protect{\noexpand\protect\noexpand}
993   \long\def\protected@write#1#2#3{%
994     \begingroup
995       \let\thepage\relax
996       #2%
997       \let\protect\@unexpandable@protect
998       \edef\reserved@a{\write#1{#3}}%
999       \reserved@a
1000    \endgroup
1001    \if@nobreak\ifvmode\nobreak\fi\fi}
1002 \fi
1003 %
1004 % \subsection{Language options}
1005 %
1006 % Languages are loaded when processing the corresponding option
1007 % \textit{except} if a |main| language has been set. In such a
1008 % case, it is not loaded until all options has been processed.
1009 % The following macro inputs the ldf file and does some additional
1010 % checks (|\input| works, too, but possible errors are not caught).
1011 %
1012 % \begin{macrocode}
1013 \bbl@trace{Language options}
1014 \let\bbl@afterlang\relax
1015 \let\BabelModifiers\relax
1016 \let\bbl@loaded\@empty
1017 \def\bbl@load@language#1{%
1018   \InputIfFileExists{#1.ldf}%
1019   {\edef\bbl@loaded{\CurrentOption

```

```

1020     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1021     \expandafter\let\expandafter\bbl@afterlang
1022     \csname\CurrentOption.ldf-h@@k\endcsname
1023     \expandafter\let\expandafter\BabelModifiers
1024     \csname bbl@mod@\CurrentOption\endcsname}%
1025     {\bbl@error{%
1026         Unknown option '\CurrentOption'. Either you misspelled it\\%
1027         or the language definition file \CurrentOption.ldf was not found}}{%
1028         Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1029         activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1030         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.

```

1031 \def\bbl@try@load@lang#1#2#3{%
1032   \IfFileExists{\CurrentOption.ldf}%
1033   {\bbl@load@language{\CurrentOption}}}%
1034   {#1\bbl@load@language{#2}#3}}
1035 \DeclareOption{hebrew}{%
1036   \input{rlbabel.def}%
1037   \bbl@load@language{hebrew}}
1038 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1039 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1040 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1041 \DeclareOption{polutonikogreek}{%
1042   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1043 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1044 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1045 \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.

```

1046 \ifx\bbl@opt@config\@nnil
1047   \@ifpackagewith{babel}{noconfigs}{}%
1048   {\InputIfFileExists{bblopts.cfg}%
1049     {\typeout{*****^J%
1050               * Local config file bblopts.cfg used^^J%
1051               *}}}%
1052   }{}%
1053 \else
1054   \InputIfFileExists{\bbl@opt@config.cfg}%
1055   {\typeout{*****^J%
1056             * Local config file \bbl@opt@config.cfg used^^J%
1057             *}}}%
1058   {\bbl@error{%
1059     Local config file '\bbl@opt@config.cfg' not found}}{%
1060     Perhaps you misspelled it.}}%
1061 \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.

```

1062 \let\bbl@tempc\relax
1063 \bbl@foreach\bbl@language@opts{%
1064   \ifcase\bbl@iniflag % Default

```

```

1065 \bbl@ifunset{ds@#1}%
1066   {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1067   {}}%
1068 \or    % provide=*
1069   \@gobble % case 2 same as 1
1070 \or    % provide+=*
1071   \bbl@ifunset{ds@#1}%
1072     {\IfFileExists{#1.ldf}{}%
1073      {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}{}}}%
1074     {}}%
1075 \bbl@ifunset{ds@#1}%
1076   {\def\bbl@tempc{#1}%
1077    \DeclareOption{#1}{%
1078      \ifnum\bbl@iniflag>\@ne
1079        \bbl@ldfinit
1080        \babelprovide[import]{#1}%
1081        \bbl@afterldf{}}%
1082      \else
1083        \bbl@load@language{#1}%
1084      \fi}}%
1085   {}}%
1086 \or    % provide*=*
1087   \def\bbl@tempc{#1}%
1088   \bbl@ifunset{ds@#1}%
1089     {\DeclareOption{#1}{%
1090       \bbl@ldfinit
1091       \babelprovide[import]{#1}%
1092       \bbl@afterldf{}}}%
1093     {}}%
1094 \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.

```

1095 \let\bbl@tempb\@nnil
1096 \bbl@foreach\@classoptionslist{%
1097   \bbl@ifunset{ds@#1}%
1098     {\IfFileExists{#1.ldf}%
1099      {\def\bbl@tempb{#1}%
1100       \DeclareOption{#1}{%
1101         \ifnum\bbl@iniflag>\@ne
1102           \bbl@ldfinit
1103           \babelprovide[import]{#1}%
1104           \bbl@afterldf{}}%
1105         \else
1106           \bbl@load@language{#1}%
1107         \fi}}%
1108      {\IfFileExists{babel-#1.tex}% TODO. Copypaste pattern
1109       {\def\bbl@tempb{#1}%
1110        \DeclareOption{#1}{%
1111          \ifnum\bbl@iniflag>\@ne
1112            \bbl@ldfinit
1113            \babelprovide[import]{#1}%
1114            \bbl@afterldf{}}%
1115          \else
1116            \bbl@load@language{#1}%
1117          \fi}}%
1118       {}}}%
1119   {}}

```

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

```

1120 \ifnum\bbl@iniflag=\z@\else
1121   \ifx\bbl@opt@main\@nnil
1122     \ifx\bbl@tempc\relax
1123       \let\bbl@opt@main\bbl@tempb
1124     \else
1125       \let\bbl@opt@main\bbl@tempc
1126     \fi
1127   \fi
1128 \fi
1129 \ifx\bbl@opt@main\@nnil\else
1130   \expandafter
1131   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1132   \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1133 \fi

```

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

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

```

1134 \def\AfterBabelLanguage#1{%
1135   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1136 \DeclareOption*{}
1137 \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.

```

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

```

```

1168 \fi
1169 \def\AfterBabelLanguage{%
1170   \bbl@error
1171   {Too late for \string\AfterBabelLanguage}%
1172   {Languages have been loaded, so I can do nothing}}

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

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

```

8 The kernel of Babel (babel.def, common)

The kernel of the babel system is currently stored in babel.def. The file babel.def contains most of the code. The file hyphen.cfg is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns.

Because plain \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

```

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

```

The file babel.def expects some definitions made in the $\LaTeX 2_{\epsilon}$ style file. So, In $\LaTeX 2.09$ and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore an alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```

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

```

```

1201 \def\AfterBabelLanguage#1#2{}
1202 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1203 \let\bbl@afterlang\relax
1204 \def\bbl@opt@safe{BR}
1205 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
1206 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1207 \expandafter\newif\csname ifbbl@single\endcsname
1208 \chardef\bbl@bidimode\z@
1209 \fi

```

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

```

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

```

And continue.

9 Multiple languages

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

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

```

1214 <<Define core switching macros>>

```

`\adddialect` The macro `\adddialect` can be used to add the name of a dialect or variant language, for which an already defined hyphenation table can be used.

```

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

```

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

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

```

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

```

```

1240         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1241         \uppercase\expandafter{\bbl@tempd}}}%
1242         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1243         \lowercase\expandafter{\bbl@tempd}}}%
1244         \@empty
1245         \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}}%
1246     \bbl@tempd
1247     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1248 \def\bbl@iflanguage#1{%
1249     \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.

We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with `\bbl@bcpcase`, casing is the correct one, so that `sr-latn-ba` becomes `fr-Latn-BA`. Note #4 may contain some `\@empty`’s, but they are eventually removed. `\bbl@bcpllookup` either returns the found ini or it is `\relax`.

```

1250 \def\bbl@bcpcase#1#2#3#4\@#5{%
1251     \ifx\@empty#3%
1252         \uppercase{\def#5{#1#2}}%
1253     \else
1254         \uppercase{\def#5{#1}}%
1255         \lowercase{\edef#5{#5#2#3#4}}%
1256     \fi}
1257 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
1258     \let\bbl@bcp\relax
1259     \lowercase{\def\bbl@tempa{#1}}%
1260     \ifx\@empty#2%
1261         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1262     \else\ifx\@empty#3%
1263         \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
1264         \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1265             {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1266             {}%
1267         \ifx\bbl@bcp\relax
1268             \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1269         \fi
1270     \else
1271         \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
1272         \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
1273         \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1274             {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1275             {}%
1276         \ifx\bbl@bcp\relax
1277             \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1278                 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1279                 {}%
1280         \fi
1281         \ifx\bbl@bcp\relax
1282             \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1283                 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1284                 {}%
1285         \fi
1286         \ifx\bbl@bcp\relax
1287             \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1288         \fi
1289     \fi\fi}
1290 \let\bbl@initoload\relax
1291 \def\bbl@provide@locale{%
1292     \ifx\babelprovide\@undefined

```

```

1293 \bbl@error{For a language to be defined on the fly 'base'\\%
1294 is not enough, and the whole package must be\\%
1295 loaded. Either delete the 'base' option or\\%
1296 request the languages explicitly}%
1297 {See the manual for further details.}%
1298 \fi
1299 % TODO. Option to search if loaded, with \LocaleForEach
1300 \let\bbl@auxname\language % Still necessary. TODO
1301 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1302 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1303 \ifbbl@bcp@allowed
1304 \expandafter\ifx\csname date\language\endcsname\relax
1305 \expandafter
1306 \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@@
1307 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
1308 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1309 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1310 \expandafter\ifx\csname date\language\endcsname\relax
1311 \let\bbl@initoload\bbl@bcp
1312 \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\language}}%
1313 \let\bbl@initoload\relax
1314 \fi
1315 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1316 \fi
1317 \fi
1318 \fi
1319 \expandafter\ifx\csname date\language\endcsname\relax
1320 \IfFileExists{babel-\language.tex}%
1321 {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\language}}}%
1322 {}%
1323 \fi}

```

`\iflanguage` Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, `\iflanguage`, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of `\language`. Then, depending on the result of the comparison, it executes either the second or the third argument.

```

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

```

9.1 Selecting the language

`\selectlanguage` The macro `\selectlanguage` checks whether the language is already defined before it performs its actual task, which is to update `\language` and activate language-specific definitions.

```

1331 \let\bbl@select@type\z@
1332 \edef\selectlanguage{%
1333 \noexpand\protect
1334 \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

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

```

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

```
1336 \let\xstring\string
```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

`\bbl@pop@language` But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's `aftergroup` mechanism to help us. The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

`\bbl@language@stack` The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```
1337 \def\bbl@language@stack{}
```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

`\bbl@push@language` The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:
`\bbl@pop@language`

```
1338 \def\bbl@push@language{%
1339   \ifx\language\undefined\else
1340     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1341   \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`.

```
1342 \def\bbl@pop@lang#1+#2\@{%
1343   \edef\language{#1}%
1344   \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).

```
1345 \let\bbl@ifrestoring\@secondoftwo
1346 \def\bbl@pop@language{%
1347   \expandafter\bbl@pop@lang\bbl@language@stack\@
1348   \let\bbl@ifrestoring\@firstoftwo
1349   \expandafter\bbl@set@language\expandafter{\language}%
1350   \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).

```
1351 \chardef\localeid\z@
1352 \def\bbl@id@last{0} % No real need for a new counter
1353 \def\bbl@id@assign{%
1354   \bbl@ifunset{bbl@id@\language}%
1355   {\count@\bbl@id@last\relax
```

```

1356 \advance\count@ \@ne
1357 \bbl@csarg\chardef{id@@\language}\count@
1358 \edef\bbl@id@last{\the\count@}%
1359 \ifcase\bbl@engine\or
1360 \directlua{
1361     Babel = Babel or {}
1362     Babel.locale_props = Babel.locale_props or {}
1363     Babel.locale_props[\bbl@id@last] = {}
1364     Babel.locale_props[\bbl@id@last].name = '\language'
1365 }%
1366 \fi}%
1367 {}%
1368 \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of \selectlanguage.

```

1369 \expandafter\def\csname selectlanguage \endcsname#1{%
1370 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@ \fi
1371 \bbl@push@language
1372 \aftergroup\bbl@pop@language
1373 \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.

```

1374 \def\BabelContentsFiles{toc,lof,lot}
1375 \def\bbl@set@language#1{% from selectlanguage, pop@
1376 % The old buggy way. Preserved for compatibility.
1377 \edef\language{%
1378 \ifnum\escapechar=\expandafter`\string#1\@empty
1379 \else\string#1\@empty\fi}%
1380 \ifcat\relax\noexpand#1%
1381 \expandafter\ifx\csname date\language\endcsname\relax
1382 \edef\language{#1}%
1383 \let\localename\language
1384 \else
1385 \bbl@info{Using '\string\language' instead of 'language' is\\%
1386 deprecated. If what you want is to use a\\%
1387 macro containing the actual locale, make\\%
1388 sure it does not match any language.\\%
1389 Reported}%
1390 % I'll\\%
1391 % try to fix '\string\localename', but I cannot promise\\%
1392 % anything. Reported}%
1393 \ifx\scantokens\undefined
1394 \def\localename{??}%
1395 \else
1396 \scantokens\expandafter{\expandafter
1397 \def\expandafter\localename\expandafter{\language}}%
1398 \fi
1399 \fi
1400 \else
1401 \def\localename{#1}% This one has the correct catcodes
1402 \fi
1403 \select@language{\language}%

```

```

1404 % write to auxs
1405 \expandafter\ifx\csname date\language\endcsname\relax\else
1406   \if@filesw
1407     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1408       % \bbl@savelastskip
1409       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1410       % \bbl@restorelastskip
1411     \fi
1412     \bbl@usehooks{write}}}%
1413   \fi
1414 \fi}
1415 % The following is used above to deal with skips before the write
1416 % whatsit. Adapted from hyperref, but it might fail, so for the moment
1417 % it's not activated. TODO.
1418 \def\bbl@savelastskip{%
1419   \let\bbl@restorelastskip\relax
1420   \ifvmode
1421     \ifdim\lastskip=\z@
1422       \let\bbl@restorelastskip\nobreak
1423     \else
1424       \bbl@exp{%
1425         \def\\bbl@restorelastskip{%
1426           \skip@=\the\lastskip
1427           \\nobreak \vskip-\skip@ \vskip\skip@}}%
1428       \fi
1429     \fi}
1430 \newif\ifbbl@bcpallowed
1431 \bbl@bcpallowedfalse
1432 \def\select@language#1{% from set@, babel@aux
1433   % set hymap
1434   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1435   % set name
1436   \edef\language{#1}%
1437   \bbl@fixname\language
1438   % TODO. name@map must be here?
1439   \bbl@provide@locale
1440   \bbl@iflanguage\language{%
1441     \expandafter\ifx\csname date\language\endcsname\relax
1442       \bbl@error
1443       {Unknown language '\language'. Either you have\\%
1444        misspelled its name, it has not been installed,\\%
1445        or you requested it in a previous run. Fix its name,\\%
1446        install it or just rerun the file, respectively. In\\%
1447        some cases, you may need to remove the aux file}%
1448       {You may proceed, but expect wrong results}%
1449     \else
1450       % set type
1451       \let\bbl@select@type\z@
1452       \expandafter\bbl@switch\expandafter{\language}%
1453     \fi}}
1454 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1455   \select@language{#1}%
1456   \bbl@foreach\BabelContentsFiles{%
1457     \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1458 \def\babel@toc#1#2{%
1459   \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.

```

1460 \newif\ifbbl@usedategroup
1461 \def\bbl@switch#1{% from select@, foreign@
1462 % make sure there is info for the language if so requested
1463 \bbl@ensureinfo{#1}%
1464 % restore
1465 \originalTeX
1466 \expandafter\def\expandafter\originalTeX\expandafter{%
1467 \csname noextras#1\endcsname
1468 \let\originalTeX\empty
1469 \babel@beginsave}%
1470 \bbl@usehooks{afterreset}{}%
1471 \languageshorthands{none}%
1472 % set the locale id
1473 \bbl@id@assign
1474 % switch captions, date
1475 % No text is supposed to be added here, so we remove any
1476 % spurious spaces.
1477 \bbl@bsphack
1478 \ifcase\bbl@select@type
1479 \csname captions#1\endcsname\relax
1480 \csname date#1\endcsname\relax
1481 \else
1482 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1483 \ifin@
1484 \csname captions#1\endcsname\relax
1485 \fi
1486 \bbl@xin@{,date,}{, \bbl@select@opts,}%
1487 \ifin@ % if \foreign... within \<lang>date
1488 \csname date#1\endcsname\relax
1489 \fi
1490 \fi
1491 \bbl@esphack
1492 % switch extras
1493 \bbl@usehooks{beforeextras}{}%
1494 \csname extras#1\endcsname\relax
1495 \bbl@usehooks{afterextras}{}%
1496 % > babel-ensure
1497 % > babel-sh-<short>
1498 % > babel-bidi
1499 % > babel-fontspec
1500 % hyphenation - case mapping
1501 \ifcase\bbl@opt@hyphenmap\or
1502 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1503 \ifnum\bbl@hymapsel>4\else
1504 \csname\language @bbl@hyphenmap\endcsname
1505 \fi
1506 \chardef\bbl@opt@hyphenmap\z@
1507 \else

```

```

1508 \ifnum\bb1@hymapsel>\bb1@opt@hyphenmap\else
1509 \csname\language @\bb1@hyphenmap\endcsname
1510 \fi
1511 \fi
1512 \let\bb1@hymapsel\cclv
1513 % hyphenation - select rules
1514 \ifnum\csname l@\language\endcsname=\l@unhyphenated
1515 \edef\bb1@tempa{u}%
1516 \else
1517 \edef\bb1@tempa{\bb1@cl{lnbrk}}%
1518 \fi
1519 % linebreaking - handle u, e, k (v in the future)
1520 \bb1@xin@{/u}{/\bb1@tempa}%
1521 \ifin@else\bb1@xin@{/e}{/\bb1@tempa}\fi % elongated forms
1522 \ifin@else\bb1@xin@{/k}{/\bb1@tempa}\fi % only kashida
1523 \ifin@else\bb1@xin@{/v}{/\bb1@tempa}\fi % variable font
1524 \ifin@
1525 % unhyphenated/kashida/elongated = allow stretching
1526 \language\l@unhyphenated
1527 \babel@savevariable\emergencystretch
1528 \emergencystretch\maxdimen
1529 \babel@savevariable\hbadness
1530 \hbadness\@M
1531 \else
1532 % other = select patterns
1533 \bb1@patterns{#1}%
1534 \fi
1535 % hyphenation - mins
1536 \babel@savevariable\lefthyphenmin
1537 \babel@savevariable\righthyphenmin
1538 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1539 \set@hyphenmins\tw@\thr@\relax
1540 \else
1541 \expandafter\expandafter\expandafter\set@hyphenmins
1542 \csname #1hyphenmins\endcsname\relax
1543 \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.

```

1544 \long\def\otherlanguage#1{%
1545 \ifnum\bb1@hymapsel=\cclv\let\bb1@hymapsel\thr@\fi
1546 \csname selectlanguage \endcsname{#1}%
1547 \ignorespaces}

```

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

```

1548 \long\def\endotherlanguage{%
1549 \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`.

```

1550 \expandafter\def\csname otherlanguage*\endcsname{%
1551 \@ifnextchar[\bb1@otherlanguage@s{\bb1@otherlanguage@s[]}}
1552 \def\bb1@otherlanguage@s[#1]#2{%

```

```

1553 \ifnum\bb1@hymapsel=\@cc1v\chardef\bb1@hymapsel4\relax\fi
1554 \def\bb1@select@opts{#1}%
1555 \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”.

```

1556 \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`.

`\bb1@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.

```

1557 \providecommand\bb1@beforeforeign{}
1558 \edef\foreignlanguage{%
1559   \noexpand\protect
1560   \expandafter\noexpand\csname foreignlanguage \endcsname}
1561 \expandafter\def\csname foreignlanguage \endcsname{%
1562   \@ifstar\bb1@foreign@s\bb1@foreign@x}
1563 \providecommand\bb1@foreign@x[3][{}]{%
1564   \begingroup
1565     \def\bb1@select@opts{#1}%
1566     \let\BabelText\@firstofone
1567     \bb1@beforeforeign
1568     \foreign@language{#2}%
1569     \bb1@usehooks{foreign}{}%
1570     \BabelText{#3}% Now in horizontal mode!
1571   \endgroup}
1572 \def\bb1@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@par
1573   \begingroup
1574     {\par}%
1575     \let\bb1@select@opts\@empty
1576     \let\BabelText\@firstofone
1577     \foreign@language{#1}%
1578     \bb1@usehooks{foreign*}{}%
1579     \bb1@dirparastext
1580     \BabelText{#2}% Still in vertical mode!
1581     {\par}%
1582   \endgroup}

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the `otherlanguage*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bb1@switch`.

```

1583 \def\foreign@language#1{%
1584   % set name
1585   \edef\language#1}%
1586   \ifbbl@usedategroup
1587     \bbl@add\bbl@select@opts{,date,}%
1588     \bbl@usedategroupfalse
1589   \fi
1590   \bbl@fixname\language
1591   % TODO. name@map here?
1592   \bbl@provide@locale
1593   \bbl@iflanguage\language{%
1594     \expandafter\ifx\csname date\language\endcsname\relax
1595       \bbl@warning % TODO - why a warning, not an error?
1596       {Unknown language '#1'. Either you have\\%
1597        misspelled its name, it has not been installed,\\%
1598        or you requested it in a previous run. Fix its name,\\%
1599        install it or just rerun the file, respectively. In\\%
1600        some cases, you may need to remove the aux file.\\%
1601        I'll proceed, but expect wrong results.\\%
1602        Reported}%
1603     \fi
1604     % set type
1605     \let\bbl@select@type\@ne
1606     \expandafter\bbl@switch\expandafter{\language}}

```

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

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

```

1607 \let\bbl@hyphlist\@empty
1608 \let\bbl@hyphenation\relax
1609 \let\bbl@pttnlist\@empty
1610 \let\bbl@patterns\relax
1611 \let\bbl@hymapsel=\@ccclv
1612 \def\bbl@patterns#1{%
1613   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1614     \csname l@#1\endcsname
1615     \edef\bbl@tempa{#1}%
1616   \else
1617     \csname l@#1:\f@encoding\endcsname
1618     \edef\bbl@tempa{#1:\f@encoding}%
1619   \fi
1620   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1621   % > luatex
1622   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
1623     \begingroup
1624       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1625       \ifin@else
1626         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1627         \hyphenation{%
1628           \bbl@hyphenation@
1629           \@ifundefined{bbl@hyphenation@#1}%
1630             \@empty
1631             {\space\csname bbl@hyphenation@#1\endcsname}}%
1632         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%

```

```

1633     \fi
1634   \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*.

1635 \def\hyphenrules#1{%
1636   \edef\bbl@tempf{#1}%
1637   \bbl@fixname\bbl@tempf
1638   \bbl@iflanguage\bbl@tempf{%
1639     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1640     \ifx\languageshorthands\undefined\else
1641       \languageshorthands{none}%
1642     \fi
1643     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1644       \set@hyphenmins\tw@\thr@@\relax
1645     \else
1646       \expandafter\expandafter\expandafter\set@hyphenmins
1647       \csname\bbl@tempf hyphenmins\endcsname\relax
1648     \fi}}
1649 \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
\langle lang\rangle hyphenmins is already defined this command has no effect.

1650 \def\providehyphenmins#1#2{%
1651   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1652     \@namedef{#1hyphenmins}{#2}%
1653   \fi}

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

1654 \def\set@hyphenmins#1#2{%
1655   \lefthyphenmin#1\relax
1656   \righthyphenmin#2\relax}

\ProvidesLanguage The identification code for each file is something that was introduced in LATEX 2ε. 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.

1657 \ifx\ProvidesFile\undefined
1658   \def\ProvidesLanguage#1[#2 #3 #4]{%
1659     \wlog{Language: #1 #4 #3 <#2>}%
1660   }
1661 \else
1662   \def\ProvidesLanguage#1{%
1663     \begingroup
1664     \catcode`\ 10 %
1665     \@makeother\%
1666     \@ifnextchar[%]
1667       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1668   \def\@provideslanguage#1[#2]{%
1669     \wlog{Language: #1 #2}%
1670     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1671   \endgroup}
1672 \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`.

```
1673 \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.

```
1674 \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’:

```
1675 \providecommand\setlocale{%
1676   \bbl@error
1677   {Not yet available}%
1678   {Find an armchair, sit down and wait}}
1679 \let\uselocale\setlocale
1680 \let\locale\setlocale
1681 \let\selectlocale\setlocale
1682 \let\localename\setlocale
1683 \let\textlocale\setlocale
1684 \let\textlanguage\setlocale
1685 \let\languagegettext\setlocale
```

9.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.
When the format knows about `\PackageError` it must be $\LaTeX 2_{\epsilon}$, 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.

```
1686 \edef\bbl@nulllanguage{\string\language=0}
1687 \ifx\PackageError\@undefined % TODO. Move to Plain
1688   \def\bbl@error#1#2{%
1689     \begingroup
1690     \newlinechar=`^^J
1691     \def\{^^J(babel) }%
1692     \errhelp{#2}\errmessage{\{#1}%
1693   \endgroup}
1694   \def\bbl@warning#1{%
1695     \begingroup
1696     \newlinechar=`^^J
1697     \def\{^^J(babel) }%
1698     \message{\{#1}%
1699   \endgroup}
1700   \let\bbl@infowarn\bbl@warning
1701   \def\bbl@info#1{%
1702     \begingroup
1703     \newlinechar=`^^J
1704     \def\{^^J}%
1705     \wlog{#1}%
1706   \endgroup}
1707 \fi
1708 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1709 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
```

```

1710 \global\@namedef{#2}{\textbf{?#1?}}%
1711 \@nameuse{#2}%
1712 \edef\bbl@tempa{#1}%
1713 \bbl@sreplace\bbl@tempa{name}{}%
1714 \bbl@warning{% TODO.
1715   \@backslashchar#1 not set for '\language'. Please,\\%
1716   define it after the language has been loaded\\%
1717   (typically in the preamble) with:\\%
1718   \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
1719   Reported}}
1720 \def\bbl@tentative{\protect\bbl@tentative@i}
1721 \def\bbl@tentative@i#1{%
1722   \bbl@warning{%
1723     Some functions for '#1' are tentative.\\%
1724     They might not work as expected and their behavior\\%
1725     could change in the future.\\%
1726     Reported}}
1727 \def\@nolanerr#1{%
1728   \bbl@error
1729   {You haven't defined the language '#1' yet.\\%
1730     Perhaps you misspelled it or your installation\\%
1731     is not complete}%
1732   {Your command will be ignored, type <return> to proceed}}
1733 \def\@nopatterns#1{%
1734   \bbl@warning
1735   {No hyphenation patterns were preloaded for\\%
1736     the language '#1' into the format.\\%
1737     Please, configure your TeX system to add them and\\%
1738     rebuild the format. Now I will use the patterns\\%
1739     preloaded for \bbl@nulllanguage\space instead}}
1740 \let\bbl@usehooks\@gobbles
1741 \ifx\bbl@onlyswitch\@empty\endinput\fi
1742 % Here ended switch.def

Here ended switch.def.

1743 \ifx\directlua\@undefined\else
1744   \ifx\bbl@luapatterns\@undefined
1745     \input luababel.def
1746   \fi
1747 \fi
1748 <<Basic macros>>
1749 \bbl@trace{Compatibility with language.def}
1750 \ifx\bbl@languages\@undefined
1751   \ifx\directlua\@undefined
1752     \openin1 = language.def % TODO. Remove hardcoded number
1753     \ifeof1
1754       \closein1
1755       \message{I couldn't find the file language.def}
1756     \else
1757       \closein1
1758       \begingroup
1759         \def\addlanguage#1#2#3#4#5{%
1760           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1761             \global\expandafter\let\csname l@#1\endcsname
1762             \csname lang@#1\endcsname
1763           \fi}%
1764         \def\uselanguage#1{%
1765           \input language.def
1766         \endgroup

```

```

1767 \fi
1768 \fi
1769 \chardef\l@english\z@
1770 \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.

```

1771 \def\addto#1#2{%
1772 \ifx#1\undefined
1773 \def#1{#2}%
1774 \else
1775 \ifx#1\relax
1776 \def#1{#2}%
1777 \else
1778 {\toks@\expandafter{#1#2}%
1779 \xdef#1{the\toks@}}%
1780 \fi
1781 \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?

```

1782 \def\bbl@withactive#1#2{%
1783 \begingroup
1784 \lccode`~=#2\relax
1785 \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`.

```

1786 \def\bbl@redefine#1{%
1787 \edef\bbl@tempa{\bbl@stripslash#1}%
1788 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1789 \expandafter\def\csname\bbl@tempa\endcsname}
1790 \@onlypreamble\bbl@redefine

```

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

```

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

```

1796 \def\bbl@redefineroobust#1{%
1797 \edef\bbl@tempa{\bbl@stripslash#1}%
1798 \bbl@ifunset{\bbl@tempa\space}%
1799 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1800 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1801 {\bbl@exp{\let\<org@\bbl@tempa\>\<\bbl@tempa\space>}}}%
1802 \@namedef{\bbl@tempa\space}}
1803 \@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.

```

1804 \bbl@trace{Hooks}
1805 \newcommand\AddBabelHook[3][\%
1806   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{\%
1807   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}\%
1808   \expandafter\bbl@tempa\bbl@evargs,##3=\@empty
1809   \bbl@ifunset{bbl@ev@#2@#3@#1}\%
1810     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}\%
1811     {\bbl@csarg\let{ev@#2@#3@#1}\relax}\%
1812   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1813 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1814 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1815 \def\bbl@usehooks#1#2{\%
1816   \def\bbl@elth##1{\%
1817     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}\%
1818     \bbl@cs{ev@#1@}\%
1819     \ifx\language\@undefined\else % Test required for Plain (?)
1820       \def\bbl@elth##1{\%
1821         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}\%
1822         \bbl@cl{ev@#1}\%
1823       \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).

```

1824 \def\bbl@evargs{\% <- don't delete this comma
1825   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1826   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1827   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1828   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1829   beforestart=0,language=2}

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro `\bbl@e@<language>` contains `\bbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1830 \bbl@trace{Defining babelensure}
1831 \newcommand\babelensure[2][\% TODO - revise test files
1832   \AddBabelHook{babel-ensure}{afterextras}\%
1833   \ifcase\bbl@select@type
1834     \bbl@cl{e}\%
1835   \fi}\%
1836 \begingroup
1837   \let\bbl@ens@include\@empty
1838   \let\bbl@ens@exclude\@empty
1839   \def\bbl@ens@fontenc{\relax}\%
1840   \def\bbl@tempb##1{\%
1841     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}\%
1842   \edef\bbl@tempa{\bbl@tempb#1\@empty}\%
1843   \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}\%

```

```

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

```

1894 \bbl@trace{Macros for setting language files up}
1895 \def\bbl@ldfinit{%
1896   \let\bbl@screset\@empty
1897   \let\BabelStrings\bbl@opt@string
1898   \let\BabelOptions\@empty
1899   \let\BabelLanguages\relax
1900   \ifx\originalTeX\@undefined
1901     \let\originalTeX\@empty
1902   \else
1903     \originalTeX
1904   \fi}
1905 \def\LdfInit#1#2{%
1906   \chardef\atcatcode=\catcode`\@
1907   \catcode`\@=11\relax
1908   \chardef\eqcatcode=\catcode`\=
1909   \catcode`\==12\relax
1910   \expandafter\if\expandafter\@backslashchar
1911     \expandafter\@car\string#2\@nil
1912     \ifx#2\@undefined\else
1913       \ldf@quit{#1}%
1914     \fi
1915   \else
1916     \expandafter\ifx\csname#2\endcsname\relax\else
1917       \ldf@quit{#1}%
1918     \fi
1919   \fi
1920   \bbl@ldfinit}

```

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

```

1921 \def\ldf@quit#1{%
1922   \expandafter\main@language\expandafter{#1}%
1923   \catcode`\@=\atcatcode \let\atcatcode\relax
1924   \catcode`\==\eqcatcode \let\eqcatcode\relax
1925   \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.

```

1926 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1927   \bbl@afterlang
1928   \let\bbl@afterlang\relax
1929   \let\BabelModifiers\relax
1930   \let\bbl@screset\relax}%
1931 \def\ldf@finish#1{%

```

```

1932 \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1933   \loadlocalcfg{#1}%
1934 \fi
1935 \bbl@afterldf{#1}%
1936 \expandafter\main@language\expandafter{#1}%
1937 \catcode`\@=\atcatcode \let\atcatcode\relax
1938 \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 .

```

1939 \@onlypreamble\LdfInit
1940 \@onlypreamble\ldf@quit
1941 \@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.

```

1942 \def\main@language#1{%
1943   \def\bbl@main@language{#1}%
1944   \let\language\main@language % TODO. Set localename
1945   \bbl@id@assign
1946   \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`.

```

1947 \def\bbl@beforestart{%
1948   \bbl@usehooks{beforestart}{}%
1949   \global\let\bbl@beforestart\relax}
1950 \AtBeginDocument{%
1951   \@nameuse{bbl@beforestart}%
1952   \if@filesw
1953     \providecommand\babel@aux[2]{}%
1954     \immediate\write\@mainaux{%
1955       \string\providecommand\string\babel@aux[2]{}%
1956       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1957   \fi
1958   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1959   \ifbbl@single % must go after the line above.
1960     \renewcommand\selectlanguage[1]{}%
1961     \renewcommand\foreignlanguage[2]{#2}%
1962     \global\let\babel@aux\@gobbletwo % Also as flag
1963   \fi
1964   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1965 \def\select@language@x#1{%
1966   \ifcase\bbl@select@type
1967     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1968   \else
1969     \select@language{#1}%
1970   \fi}

```

9.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if \LaTeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1971 \bbl@trace{Shorhands}
1972 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1973 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1974 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1975 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1976 \begingroup
1977 \catcode`#1\active
1978 \nfss@catcodes
1979 \ifnum\catcode`#1=\active
1980 \endgroup
1981 \bbl@add\nfss@catcodes{\@makeother#1}%
1982 \else
1983 \endgroup
1984 \fi
1985 \fi}

```

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

```

1986 \def\bbl@remove@special#1{%
1987 \begingroup
1988 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1989 \else\noexpand##1\noexpand##2\fi}%
1990 \def\do{\x\do}%
1991 \def\@makeother{\x\@makeother}%
1992 \edef\x{\endgroup
1993 \def\noexpand\dospecials{\dospecials}%
1994 \expandafter\ifx\cname @sanitize\endcname\relax\else
1995 \def\noexpand\@sanitize{\@sanitize}%
1996 \fi}%
1997 \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).

```

1998 \def\bbl@active@def#1#2#3#4{%
1999 \namedef{#3#1}{%
2000 \expandafter\ifx\cname#2@sh@#1\endcname\relax
2001 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
2002 \else
2003 \bbl@afterfi\cname#2@sh@#1\endcname
2004 \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.

```

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

```

2011 \def\initiate@active@char#1{%
2012   \bbl@ifunset{active@char\string#1}%
2013   {\bbl@withactive
2014     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
2015   {}}

```

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).

```

2016 \def\@initiate@active@char#1#2#3{%
2017   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
2018   \ifx#1\@undefined
2019     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
2020   \else
2021     \bbl@csarg\let{oridef@#2}#1%
2022     \bbl@csarg\edef{oridef@#2}{%
2023       \let\noexpand#1%
2024       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
2025   \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).

```

2026 \ifx#1#3\relax
2027   \expandafter\let\csname normal@char#2\endcsname#3%
2028 \else
2029   \bbl@info{Making #2 an active character}%
2030   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2031   \@namedef{normal@char#2}{%
2032     \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
2033   \else
2034     \@namedef{normal@char#2}{#3}%
2035   \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).

```

2036 \bbl@restoreactive{#2}%
2037 \AtBeginDocument{%
2038   \catcode`#2\active
2039   \if@filesw
2040     \immediate\write\@mainaux{\catcode`\string#2\active}%
2041   \fi}%

```

```

2042 \expandafter\bb1@add@special\csname#2\endcsname
2043 \catcode`#2\active
2044 \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⟩`).

```

2045 \let\bb1@tempa\@firstoftwo
2046 \if\string^#2%
2047 \def\bb1@tempa{\noexpand\textormath}%
2048 \else
2049 \ifx\bb1@mathnormal\@undefined\else
2050 \let\bb1@tempa\bb1@mathnormal
2051 \fi
2052 \fi
2053 \expandafter\edef\csname active@char#2\endcsname{%
2054 \bb1@tempa
2055 {\noexpand\if@safe@actives
2056 \noexpand\expandafter
2057 \expandafter\noexpand\csname normal@char#2\endcsname
2058 \noexpand\else
2059 \noexpand\expandafter
2060 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2061 \noexpand\fi}%
2062 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2063 \bb1@csarg\edef{doactive#2}{%
2064 \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!).

```

2065 \bb1@csarg\edef{active@#2}{%
2066 \noexpand\active@prefix\noexpand#1%
2067 \expandafter\noexpand\csname active@char#2\endcsname}%
2068 \bb1@csarg\edef{normal@#2}{%
2069 \noexpand\active@prefix\noexpand#1%
2070 \expandafter\noexpand\csname normal@char#2\endcsname}%
2071 \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.

```

2072 \bb1@active@def#2\user@group{user@active}{language@active}%
2073 \bb1@active@def#2\language@group{language@active}{system@active}%
2074 \bb1@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.

```

2075 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2076 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2077 \expandafter\edef\csname\user@group @sh@#2@string\protect@\endcsname
2078 {\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.

```
2079 \if\string'#2%
2080 \let\prim@s\bbl@prim@s
2081 \let\active@math@prime#1%
2082 \fi
2083 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}
```

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

```
2084 <<{*More package options}>> ≡
2085 \DeclareOption{math=active}{}
2086 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2087 <</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.

```
2088 \@ifpackagewith{babel}{KeepShorthandsActive}%
2089 {\let\bbl@restoreactive\@gobble}%
2090 {\def\bbl@restoreactive#1{%
2091   \bbl@exp{%
2092     \\\AfterBabelLanguage\\CurrentOption
2093     {\catcode`#1=\the\catcode`#1\relax}%
2094     \\\AtEndOfPackage
2095     {\catcode`#1=\the\catcode`#1\relax}}}%
2096 \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.

```
2097 \def\bbl@sh@select#1#2{%
2098   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2099     \bbl@afterelse\bbl@scndcs
2100   \else
2101     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2102   \fi}
```

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

```
2103 \begingroup
2104 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2105 {\gdef\active@prefix#1{%
2106   \ifx\protect\@typeset@protect
2107   \else
2108     \ifx\protect\@unexpandable@protect
2109       \noexpand#1%
2110     \else
2111       \protect#1%
2112     \fi
```

```

2113     \expandafter\@gobble
2114     \fi}}
2115 {\gdef\active@prefix#1{%
2116     \ifincsname
2117     \string#1%
2118     \expandafter\@gobble
2119     \else
2120     \ifx\protect\@typeset@protect
2121     \else
2122     \ifx\protect\@unexpandable@protect
2123     \noexpand#1%
2124     \else
2125     \protect#1%
2126     \fi
2127     \expandafter\expandafter\expandafter\@gobble
2128     \fi
2129     \fi}}
2130 \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⟩`.

```

2131 \newif\if@safe@actives
2132 \@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.

```

2133 \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`.

```

2134 \chardef\bbl@activated\z@
2135 \def\bbl@activate#1{%
2136     \chardef\bbl@activated\@ne
2137     \bbl@withactive{\expandafter\let\expandafter}#1%
2138     \csname bbl@active@\string#1\endcsname}
2139 \def\bbl@deactivate#1{%
2140     \chardef\bbl@activated\tw@
2141     \bbl@withactive{\expandafter\let\expandafter}#1%
2142     \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2143 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2144 \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.

```

2145 \def\babel@texpdf#1#2#3#4{%
2146   \ifx\texorpdfstring\undefined
2147     \textormath{#1}{#3}%
2148   \else
2149     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2150     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2151   \fi}
2152 %
2153 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2154 \def\@decl@short#1#2#3\@nil#4{%
2155   \def\bbl@tempa{#3}%
2156   \ifx\bbl@tempa\empty
2157     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2158     \bbl@ifunset{#1@sh@\string#2@}{}%
2159     {\def\bbl@tempa{#4}%
2160      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2161      \else
2162        \bbl@info
2163          {Redefining #1 shorthand \string#2\%
2164           in language \CurrentOption}%
2165        \fi}%
2166     \@namedef{#1@sh@\string#2@}{#4}%
2167   \else
2168     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2169     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2170     {\def\bbl@tempa{#4}%
2171      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2172      \else
2173        \bbl@info
2174          {Redefining #1 shorthand \string#2\string#3\%
2175           in language \CurrentOption}%
2176        \fi}%
2177     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2178   \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.

```

2179 \def\textormath{%
2180   \ifmmode
2181     \expandafter\@secondoftwo
2182   \else
2183     \expandafter\@firstoftwo
2184   \fi}

```

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

```

2185 \def\user@group{user}
2186 \def\language@group{english} % TODO. I don't like defaults
2187 \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.

```

2188 \def\useshorthands{%
2189   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2190 \def\bbl@usesh@s#1{%
2191   \bbl@usesh@x

```

```

2192     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2193     {#1}}
2194 \def\bbl@usesesh@x#1#2{%
2195   \bbl@ifshorthand{#2}%
2196   {\def\user@group{user}%
2197     \initiate@active@char{#2}%
2198     #1%
2199     \bbl@activate{#2}}%
2200   {\bbl@error
2201     {Cannot declare a shorthand turned off (\string#2)}
2202     {Sorry, but you cannot use shorthands which have been\\%
2203       turned off in the package options}}}

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

2204 \def\user@language@group{user@\language@group}
2205 \def\bbl@set@user@generic#1#2{%
2206   \bbl@ifunset{user@generic@active#1}%
2207   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2208     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2209     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2210       \expandafter\noexpand\csname normal@char#1\endcsname}%
2211     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2212       \expandafter\noexpand\csname user@active#1\endcsname}}%
2213   \@empty}
2214 \newcommand\defineshorthand[3][user]{%
2215   \edef\bbl@tempa{\zap@space#1 \@empty}%
2216   \bbl@for\bbl@tempb\bbl@tempa{%
2217     \if*\expandafter\@car\bbl@tempb\@nil
2218       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2219       \@expandtwoargs
2220       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2221     \fi
2222     \declare@shorthand{\bbl@tempb}{#2}{#3}}}

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

2223 \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".

2224 \def\aliasshorthand#1#2{%
2225   \bbl@ifshorthand{#2}%
2226   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2227     \ifx\document\@notprerr
2228       \@notshorthand{#2}%
2229     \else
2230       \initiate@active@char{#2}%
2231       \expandafter\let\csname active@char\string#2\endcsname\expandafter\endcsname
2232       \csname active@char\string#1\endcsname
2233       \expandafter\let\csname normal@char\string#2\endcsname\expandafter\endcsname
2234       \csname normal@char\string#1\endcsname
2235       \bbl@activate{#2}%

```

```

2236     \fi
2237     \fi}%
2238     {\bbl@error
2239       {Cannot declare a shorthand turned off (\string#2)}
2240       {Sorry, but you cannot use shorthands which have been\\%
2241         turned off in the package options}}}

\@notshorthand

2242 \def\@notshorthand#1{%
2243   \bbl@error{%
2244     The character '\string #1' should be made a shorthand character;\\%
2245     add the command \string\usesshorthands\string{#1\string} to
2246     the preamble.\\%
2247     I will ignore your instruction}%
2248   {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.

2249 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2250 \DeclareRobustCommand*\shorthandoff{%
2251   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2252 \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.

2253 \def\bbl@switch@sh#1#2{%
2254   \ifx#2\@nnil\else
2255     \bbl@ifunset{\bbl@active@\string#2}%
2256     {\bbl@error
2257       {I can't switch '\string#2' on or off--not a shorthand}%
2258       {This character is not a shorthand. Maybe you made\\%
2259         a typing mistake? I will ignore your instruction.}}%
2260     {\ifcase#1%   off, on, off*
2261       \catcode`#212\relax
2262     \or
2263       \catcode`#2\active
2264       \bbl@ifunset{\bbl@shdef@\string#2}%
2265       {}%
2266       {\bbl@withactive{\expandafter\let\expandafter}#2%
2267         \csname bbl@shdef@\string#2\endcsname
2268         \bbl@csarg\let{\shdef@\string#2}\relax}%
2269       \ifcase\bbl@activated\or
2270         \bbl@activate{#2}%
2271       \else
2272         \bbl@deactivate{#2}%
2273       \fi
2274     \or
2275       \bbl@ifunset{\bbl@shdef@\string#2}%
2276       {\bbl@withactive{\bbl@csarg\let{\shdef@\string#2}}#2}%
2277       {}%
2278       \csname bbl@oricat@\string#2\endcsname
2279       \csname bbl@oridef@\string#2\endcsname
2280     \fi}%

```

```

2281 \bbl@afterfi\bbl@switch@sh#1%
2282 \fi}

```

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

```

2283 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2284 \def\bbl@putsh#1{%
2285 \bbl@ifunset{bbl@active@\string#1}%
2286 {\bbl@putsh@i#1\@empty\@nnil}%
2287 {\csname bbl@active@\string#1\endcsname}}
2288 \def\bbl@putsh@i#1#2\@nnil{%
2289 \csname\language@group @sh@\string#1@%
2290 \ifx\@empty#2\else\string#2\fi\endcsname}
2291 \ifx\bbl@opt@shorthands\@nnil\else
2292 \let\bbl@s@initiate@active@char\initiate@active@char
2293 \def\initiate@active@char#1{%
2294 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2295 \let\bbl@s@switch@sh\bbl@switch@sh
2296 \def\bbl@switch@sh#1#2{%
2297 \ifx#2\@nnil\else
2298 \bbl@afterfi
2299 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2300 \fi}
2301 \let\bbl@s@activate\bbl@activate
2302 \def\bbl@activate#1{%
2303 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2304 \let\bbl@s@deactivate\bbl@deactivate
2305 \def\bbl@deactivate#1{%
2306 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2307 \fi

```

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

```

2308 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

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

```

2309 \def\bbl@prim@s{%
2310 \prime\futurelet\@let@token\bbl@pr@m@s}
2311 \def\bbl@if@primes#1#2{%
2312 \ifx#1\@let@token
2313 \expandafter\@firstoftwo
2314 \else\ifx#2\@let@token
2315 \bbl@afterelse\expandafter\@firstoftwo
2316 \else
2317 \bbl@afterfi\expandafter\@secondoftwo
2318 \fi\fi}
2319 \begingroup
2320 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2321 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\'
2322 \lowercase{%
2323 \gdef\bbl@pr@m@s{%
2324 \bbl@if@primes"%
2325 \pr@@@s
2326 {\bbl@if@primes*\^{\pr@@@t\egroup}}}
2327 \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).

```
2328 \initiate@active@char{~}
2329 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2330 \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.

```
2331 \expandafter\def\csname OT1dqpos\endcsname{127}
2332 \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

```
2333 \ifx\f@encoding\@undefined
2334   \def\f@encoding{OT1}
2335 \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.

```
2336 \bbl@trace{Language attributes}
2337 \newcommand\languageattribute[2]{%
2338   \def\bbl@tempc{#1}%
2339   \bbl@fixname\bbl@tempc
2340   \bbl@iflanguage\bbl@tempc{%
2341     \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.

```
2342     \ifx\bbl@known@attribs\@undefined
2343       \in@false
2344     \else
2345       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2346     \fi
2347     \ifin@
2348       \bbl@warning{%
2349         You have more than once selected the attribute '##1'\%
2350         for language #1. Reported}%
2351     \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.

```
2352       \bbl@exp{%
2353         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2354       \edef\bbl@tempa{\bbl@tempc-##1}%
2355       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2356       {\csname\bbl@tempc @attr##1\endcsname}%
2357       {\@attrerr{\bbl@tempc}{##1}}%
2358     \fi}}
2359 \@onlypreamble\languageattribute
```

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

```
2360 \newcommand*{\@attrerr}[2]{%
2361   \bbl@error
2362   {The attribute #2 is unknown for language #1.}%
2363   {Your command will be ignored, type <return> to proceed}}
```

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

```
2364 \def\bbl@declare@ttribute#1#2#3{%
2365   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2366   \ifin@
2367     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2368   \fi
2369   \bbl@add@list\bbl@attributes{#1-#2}%
2370   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

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

```
2371 \def\bbl@ifattributeset#1#2#3#4{%
2372   \ifx\bbl@known@attribs\@undefined
2373     \in@false
2374   \else
2375     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2376   \fi
2377   \ifin@
2378     \bbl@afterelse#3%
2379   \else
2380     \bbl@afterfi#4%
2381   \fi}
```

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

```
2382 \def\bbl@ifknown@ttrib#1#2{%
2383   \let\bbl@tempa\@secondoftwo
2384   \bbl@loopx\bbl@tempb{#2}{%
2385     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2386   \ifin@
2387     \let\bbl@tempa\@firstoftwo
2388   \else
2389   \fi}%
2390   \bbl@tempa}
```

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

```
2391 \def\bbl@clear@ttribs{%
2392   \ifx\bbl@attributes\@undefined\else
2393     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2394       \expandafter\bbl@clear@ttrib\bbl@tempa.
2395     }%
2396     \let\bbl@attributes\@undefined
```

```

2397 \fi}
2398 \def\bbl@clear@ttrib#1-#2.{%
2399 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2400 \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`

```

2401 \bbl@trace{Macros for saving definitions}
2402 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2403 \newcount\babel@savecnt
2404 \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.

```

2405 \def\babel@save#1{%
2406 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2407 \toks@\expandafter{\originalTeX\let#1=}%
2408 \bbl@exp{%
2409 \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2410 \advance\babel@savecnt@ne}
2411 \def\babel@savevariable#1{%
2412 \toks@\expandafter{\originalTeX #1=}%
2413 \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

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

```

2414 \def\bbl@frenchspacing{%
2415 \ifnum\the\sfcode\.\=@m
2416 \let\bbl@nonfrenchspacing\relax
2417 \else
2418 \frenchspacing
2419 \let\bbl@nonfrenchspacing\nonfrenchspacing
2420 \fi}
2421 \let\bbl@nonfrenchspacing\nonfrenchspacing
2422 \let\bbl@elt\relax
2423 \edef\bbl@fs@chars{%
2424 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2425 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2426 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}

```

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

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.

```

2427 \bbl@trace{Short tags}
2428 \def\babeltags#1{%
2429   \edef\bbl@tempa{\zap@space#1 \@empty}%
2430   \def\bbl@tempb##1=##2\@{
2431     \edef\bbl@tempc{%
2432       \noexpand\newcommand
2433       \expandafter\noexpand\csname ##1\endcsname{%
2434         \noexpand\protect
2435         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2436       \noexpand\newcommand
2437       \expandafter\noexpand\csname text##1\endcsname{%
2438         \noexpand\foreignlanguage{##2}}}
2439   \bbl@tempc}%
2440   \bbl@for\bbl@tempa\bbl@tempa{%
2441     \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.

```

2442 \bbl@trace{Hyphens}
2443 \@onlypreamble\babelhyphenation
2444 \AtEndOfPackage{%
2445   \newcommand\babelhyphenation[2][\@empty]{%
2446     \ifx\bbl@hyphenation@relax
2447       \let\bbl@hyphenation@\@empty
2448     \fi
2449     \ifx\bbl@hyphlist\@empty\else
2450       \bbl@warning{%
2451         You must not intermingle \string\selectlanguage\space and\%
2452         \string\babelhyphenation\space or some exceptions will not\%
2453         be taken into account. Reported}%
2454     \fi
2455     \ifx\@empty#1%
2456       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2457     \else
2458       \bbl@vforeach{#1}{%
2459         \def\bbl@tempa{##1}%
2460         \bbl@fixname\bbl@tempa
2461         \bbl@iflanguage\bbl@tempa{%
2462           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2463             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2464             {}%
2465             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2466             #2}}}%
2467       \fi}

```

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

```

2468 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}

```

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

```

2469 \def\bbl@t@one{T1}
2470 \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.

```

2471 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2472 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2473 \def\bbl@hyphen{%
2474   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2475 \def\bbl@hyphen@i#1#2{%
2476   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
2477   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2478   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

```

2479 \def\bbl@usehyphen#1{%
2480   \leavevmode
2481   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2482   \nobreak\hskip\z@skip}
2483 \def\bbl@@usehyphen#1{%
2484   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2485 \def\bbl@hyphenchar{%
2486   \ifnum\hyphenchar\font=\m@ne
2487     \babellnullhyphen
2488   \else
2489     \char\hyphenchar\font
2490   \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.

```

2491 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2492 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2493 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2494 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2495 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2496 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2497 \def\bbl@hy@repeat{%
2498   \bbl@usehyphen{%
2499     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2500 \def\bbl@hy@@repeat{%
2501   \bbl@usehyphen{%
2502     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2503 \def\bbl@hy@empty{\hskip\z@skip}
2504 \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.

```

2505 \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.

```
2506 \bbl@trace{Multiencoding strings}
2507 \def\bbl@tglobal#1{\global\let#1#1}
2508 \def\bbl@recatcode#1{% TODO. Used only once?
2509   \@tempcnta="7F
2510   \def\bbl@tempa{%
2511     \ifnum\@tempcnta>"FF\else
2512       \catcode\@tempcnta=#1\relax
2513       \advance\@tempcnta\@ne
2514       \expandafter\bbl@tempa
2515     \fi}%
2516   \bbl@tempa}
```

The second one. We need to patch `\@uclclist`, but it is done once and only if `\SetCase` is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact `\@uclclist` is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually `\reserved@a`), we pass it as argument to `\bbl@uclc`. The parser is restarted inside `\lang\bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

```
\let\bbl@tolower\@empty\bbl@toupper\@empty
```

and starts over (and similarly when lowercasing).

```
2517 \ifpackagewith{babel}{nocase}%
2518   {\let\bbl@patchuclc\relax}%
2519   {\def\bbl@patchuclc{%
2520     \global\let\bbl@patchuclc\relax
2521     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2522     \gdef\bbl@uclc##1{%
2523       \let\bbl@encoded\bbl@encoded@uclc
2524       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2525       {##1}%
2526       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2527         \csname\language @bbl@uclc\endcsname}%
2528       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2529     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2530     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2531 \langle\langle *More package options\rangle\rangle \equiv
2532 \DeclareOption{nocase}{}
2533 \rangle\rangle\langle\langle /More package options\rangle\rangle
```

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

```
2534 \langle\langle *More package options\rangle\rangle \equiv
2535 \let\bbl@opt@strings\@nnil % accept strings=value
2536 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2537 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2538 \def\BabelStringsDefault{generic}
2539 \rangle\rangle\langle\langle /More package options\rangle\rangle
```

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

```

2540 \@onlypreamble\StartBabelCommands
2541 \def\StartBabelCommands{%
2542   \begingroup
2543   \bbl@recatcode{11}%
2544   <\Macros local to BabelCommands>
2545   \def\bbl@provstring##1##2{%
2546     \providecommand##1{##2}%
2547     \bbl@tglobal##1}%
2548   \global\let\bbl@scafter\@empty
2549   \let\StartBabelCommands\bbl@startcmds
2550   \ifx\BabelLanguages\relax
2551     \let\BabelLanguages\CurrentOption
2552   \fi
2553   \begingroup
2554   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2555   \StartBabelCommands}
2556 \def\bbl@startcmds{%
2557   \ifx\bbl@screset\@nnil\else
2558     \bbl@usehooks{stopcommands}{}%
2559   \fi
2560   \endgroup
2561   \begingroup
2562   \@ifstar
2563     {\ifx\bbl@opt@strings\@nnil
2564       \let\bbl@opt@strings\BabelStringsDefault
2565     \fi
2566     \bbl@startcmds@i}%
2567   \bbl@startcmds@i}
2568 \def\bbl@startcmds@i#1#2{%
2569   \edef\bbl@L{\zap@space#1 \@empty}%
2570   \edef\bbl@G{\zap@space#2 \@empty}%
2571   \bbl@startcmds@ii}
2572 \let\bbl@startcmds\StartBabelCommands

```

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

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

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

```

2573 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2574   \let\SetString\@gobbletwo
2575   \let\bbl@stringdef\@gobbletwo
2576   \let\AfterBabelCommands\@gobble
2577   \ifx\@empty#1%
2578     \def\bbl@sc@label{generic}%
2579     \def\bbl@encstring##1##2{%
2580       \ProvideTextCommandDefault##1{##2}%
2581       \bbl@tglobal##1%
2582       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2583     \let\bbl@sctest\in@true
2584   \else
2585     \let\bbl@sc@charset\space % <- zapped below

```

```

2586 \let\bbl@sc@fontenc\space % <- " "
2587 \def\bbl@tempa##1=##2\@nil{%
2588 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
2589 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2590 \def\bbl@tempa##1 ##2{% space -> comma
2591 ##1%
2592 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2593 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2594 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2595 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2596 \def\bbl@encstring##1##2{%
2597 \bbl@foreach\bbl@sc@fontenc{%
2598 \bbl@ifunset{T####1}%
2599 {}%
2600 {\ProvideTextCommand##1{####1}{##2}%
2601 \bbl@tglobal##1%
2602 \expandafter
2603 \bbl@tglobal\csname####1\string##1\endcsname}}}%
2604 \def\bbl@sctest{%
2605 \bbl@xin@{\, \bbl@opt@strings,}{, \bbl@sc@label, \bbl@sc@fontenc,}}%
2606 \fi
2607 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2608 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2609 \let\AfterBabelCommands\bbl@aftercmds
2610 \let\SetString\bbl@setstring
2611 \let\bbl@stringdef\bbl@encstring
2612 \else % ie, strings=value
2613 \bbl@sctest
2614 \ifin@
2615 \let\AfterBabelCommands\bbl@aftercmds
2616 \let\SetString\bbl@setstring
2617 \let\bbl@stringdef\bbl@provstring
2618 \fi\fi\fi
2619 \bbl@scswitch
2620 \ifx\bbl@G\@empty
2621 \def\SetString##1##2{%
2622 \bbl@error{Missing group for string \string##1}%
2623 {You must assign strings to some category, typically\\%
2624 captions or extras, but you set none}}%
2625 \fi
2626 \ifx\@empty#1%
2627 \bbl@usehooks{defaultcommands}}}%
2628 \else
2629 \@expandtwoargs
2630 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2631 \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).

```

2632 \def\bbl@forlang#1##2{%
2633 \bbl@for#1\bbl@L{%
2634 \bbl@xin@{, #1,}{, \BabelLanguages,}%
2635 \ifin@#2\relax\fi}}

```

```

2636 \def\bbl@scswitch{%
2637   \bbl@forlang\bbl@tempa{%
2638     \ifx\bbl@G\@empty\else
2639       \ifx\SetString\@gobbletwo\else
2640         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2641         \bbl@xin{,\bbl@GL,}{,\bbl@screset,}%
2642         \ifin@\else
2643           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2644           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2645         \fi
2646       \fi
2647     \fi}}
2648 \AtEndOfPackage{%
2649   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
2650   \let\bbl@scswitch\relax}
2651 \@onlypreamble\EndBabelCommands
2652 \def\EndBabelCommands{%
2653   \bbl@usehooks{stopcommands}{}%
2654   \endgroup
2655   \endgroup
2656   \bbl@scafter}
2657 \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.

```

2658 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2659   \bbl@forlang\bbl@tempa{%
2660     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2661     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2662     {\bbl@exp{%
2663       \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2664     }%
2665     \def\BabelString{#2}%
2666     \bbl@usehooks{stringprocess}{}%
2667     \expandafter\bbl@stringdef
2668     \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.

```

2669 \ifx\bbl@opt@strings\relax
2670   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2671   \bbl@patchuclc
2672   \let\bbl@encoded\relax
2673   \def\bbl@encoded@uclc#1{%
2674     \@inmathwarn#1%
2675     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2676       \expandafter\ifx\csname ?\string#1\endcsname\relax
2677         \TextSymbolUnavailable#1%
2678       \else
2679         \csname ?\string#1\endcsname
2680       \fi
2681     \else
2682       \csname\cf@encoding\string#1\endcsname

```

```

2683     \fi}
2684 \else
2685   \def\bbl@scset#1#2{\def#1{#2}}
2686 \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.

```

2687 << *Macros local to BabelCommands >> ≡
2688 \def\SetStringLoop##1##2{%
2689   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2690   \count@\z@
2691   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2692     \advance\count@\@ne
2693     \toks@\expandafter{\bbl@tempa}%
2694     \bbl@exp{%
2695       \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2696       \count@=\the\count@\relax}}}%
2697 <</Macros local to BabelCommands >>

```

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

```

2698 \def\bbl@aftercmds#1{%
2699   \toks@\expandafter{\bbl@scafter#1}%
2700   \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.

```

2701 << *Macros local to BabelCommands >> ≡
2702 \newcommand\SetCase[3][]{%
2703   \bbl@patchuclc
2704   \bbl@forlang\bbl@tempa{%
2705     \expandafter\bbl@encstring
2706     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2707     \expandafter\bbl@encstring
2708     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2709     \expandafter\bbl@encstring
2710     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2711 <</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.

```

2712 << *Macros local to BabelCommands >> ≡
2713 \newcommand\SetHyphenMap[1]{%
2714   \bbl@forlang\bbl@tempa{%
2715     \expandafter\bbl@stringdef
2716     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2717 <</Macros local to BabelCommands >>

```

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

```

2718 \newcommand\BabelLower[2]{% one to one.
2719   \ifnum\lccode#1=#2\else
2720     \babel@savevariable{\lccode#1}%
2721     \lccode#1=#2\relax
2722   \fi}
2723 \newcommand\BabelLowerMM[4]{% many-to-many
2724   \@tempcnta=#1\relax

```

```

2725 \@tempcntb=#4\relax
2726 \def\bbl@tempa{%
2727   \ifnum\@tempcnta>#2\else
2728     \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2729     \advance\@tempcnta#3\relax
2730     \advance\@tempcntb#3\relax
2731     \expandafter\bbl@tempa
2732   \fi}%
2733 \bbl@tempa}
2734 \newcommand\BabelLowerM0[4]{% many-to-one
2735   \@tempcnta=#1\relax
2736   \def\bbl@tempa{%
2737     \ifnum\@tempcnta>#2\else
2738       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2739       \advance\@tempcnta#3
2740       \expandafter\bbl@tempa
2741     \fi}%
2742   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2743 <<{*More package options}>> ≡
2744 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2745 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2746 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2747 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@}
2748 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2749 <</More package options>>

```

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

```

2750 \AtEndOfPackage{%
2751   \ifx\bbl@opt@hyphenmap\undefined
2752     \bbl@xin@{,}{\bbl@language@opts}%
2753     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2754   \fi}

```

This section 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.

```

2755 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2756   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2757 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2758   \bbl@trim@def\bbl@tempa{#2}%
2759   \bbl@xin@{.template}{\bbl@tempa}%
2760   \ifin@
2761     \bbl@ini@captions@template{#3}{#1}%
2762   \else
2763     \edef\bbl@tempd{%
2764       \expandafter\expandafter\expandafter
2765       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2766     \bbl@xin@
2767       {\expandafter\string\csname #2name\endcsname}%
2768     {\bbl@tempd}%
2769     \ifin@ % Renew caption
2770       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2771     \ifin@
2772       \bbl@exp{%
2773         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2774         {\bbl@scset\<#2name>\<#1#2name>}%
2775         {}}%

```

```

2776 \else % Old way converts to new way
2777 \bbl@ifunset{#1#2name}%
2778 {\bbl@exp{%
2779 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
2780 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2781 {\def\<#2name>{\<#1#2name>}}}%
2782 {}}}%
2783 {}%
2784 \fi
2785 \else
2786 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2787 \ifin@ % New way
2788 \bbl@exp{%
2789 \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}}%
2790 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2791 {\\\bbl@scset\<#2name>\<#1#2name>}}%
2792 {}}}%
2793 \else % Old way, but defined in the new way
2794 \bbl@exp{%
2795 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
2796 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2797 {\def\<#2name>{\<#1#2name>}}}%
2798 {}}}%
2799 \fi%
2800 \fi
2801 \@namedef{#1#2name}{#3}%
2802 \toks@\expandafter{\bbl@captionslist}%
2803 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2804 \ifin@ \else
2805 \bbl@exp{\\\bbl@add\\bbl@captionslist{\<#2name>}}%
2806 \bbl@tglobal\bbl@captionslist
2807 \fi
2808 \fi}
2809 % \def\bbl@setcaption@#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.

```

2810 \bbl@trace{Macros related to glyphs}
2811 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2812 \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2813 \setbox\z@ \hbox{\lower\dimen\z@ \box\z@}\ht\z@ \ht\tw@ \dp\z@ \dp\tw@}

```

`\save@sf@q` The macro `\save@sf@q` is used to save and reset the current space factor.

```

2814 \def\save@sf@q#1{\leavevmode
2815 \begingroup
2816 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2817 \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.

```
2818 \ProvideTextCommand{\quotedblbase}{OT1}{%
2819   \save@sf@q{\set@low@box{\textquotedblright\}%
2820     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
2821 \ProvideTextCommandDefault{\quotedblbase}{%
2822   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2823 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2824   \save@sf@q{\set@low@box{\textquoteright\}%
2825     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
2826 \ProvideTextCommandDefault{\quotesinglbase}{%
2827   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2828 \ProvideTextCommand{\guillemetleft}{OT1}{%
2829   \ifmmode
2830     \ll
2831   \else
2832     \save@sf@q{\nobreak
2833       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2834   \fi}
2835 \ProvideTextCommand{\guillemetright}{OT1}{%
2836   \ifmmode
2837     \gg
2838   \else
2839     \save@sf@q{\nobreak
2840       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2841   \fi}
2842 \ProvideTextCommand{\guillemotleft}{OT1}{%
2843   \ifmmode
2844     \ll
2845   \else
2846     \save@sf@q{\nobreak
2847       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2848   \fi}
2849 \ProvideTextCommand{\guillemotright}{OT1}{%
2850   \ifmmode
2851     \gg
2852   \else
2853     \save@sf@q{\nobreak
2854       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2855   \fi}
```

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

```
2856 \ProvideTextCommandDefault{\guillemetleft}{%
2857   \UseTextSymbol{OT1}{\guillemetleft}}
2858 \ProvideTextCommandDefault{\guillemetright}{%
2859   \UseTextSymbol{OT1}{\guillemetright}}
2860 \ProvideTextCommandDefault{\guillemotleft}{%
2861   \UseTextSymbol{OT1}{\guillemotleft}}
2862 \ProvideTextCommandDefault{\guillemotright}{%
2863   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```

2864 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2865   \ifmmode
2866     <%
2867   \else
2868     \save@sf@q{\nobreak
2869       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2870   \fi}
2871 \ProvideTextCommand{\guilsinglright}{OT1}{%
2872   \ifmmode
2873     >%
2874   \else
2875     \save@sf@q{\nobreak
2876       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2877   \fi}

```

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

```

2878 \ProvideTextCommandDefault{\guilsinglleft}{%
2879   \UseTextSymbol{OT1}{\guilsinglleft}}
2880 \ProvideTextCommandDefault{\guilsinglright}{%
2881   \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.

```

2882 \DeclareTextCommand{\ij}{OT1}{%
2883   i\kern-0.02em\bbl@allowhyphens j}
2884 \DeclareTextCommand{\IJ}{OT1}{%
2885   I\kern-0.02em\bbl@allowhyphens J}
2886 \DeclareTextCommand{\ij}{T1}{\char188}
2887 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2888 \ProvideTextCommandDefault{\ij}{%
2889   \UseTextSymbol{OT1}{\ij}}
2890 \ProvideTextCommandDefault{\IJ}{%
2891   \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).

```

2892 \def\crrtic@{\hrule height0.1ex width0.3em}
2893 \def\crttic@{\hrule height0.1ex width0.33em}
2894 \def\ddj@{%
2895   \setbox0\hbox{d}\dimen@=\ht0
2896   \advance\dimen@1ex
2897   \dimen@.45\dimen@
2898   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2899   \advance\dimen@ii.5ex
2900   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\box{\crrtic@}}}}
2901 \def\DDJ@{%
2902   \setbox0\hbox{D}\dimen@=.55\ht0
2903   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2904   \advance\dimen@ii.15ex % correction for the dash position
2905   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2906   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@

```

```

2907 \leavevmode\rlap{\raise\dimen@hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2908 %
2909 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2910 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2911 \ProvideTextCommandDefault{\dj}{%
2912 \UseTextSymbol{OT1}{\dj}}
2913 \ProvideTextCommandDefault{\DJ}{%
2914 \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.

```

2915 \DeclareTextCommand{\SS}{OT1}{SS}
2916 \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 2917 \ProvideTextCommandDefault{\glq}{%
2918 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2919 \ProvideTextCommand{\grq}{T1}{%
2920 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2921 \ProvideTextCommand{\grq}{TU}{%
2922 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2923 \ProvideTextCommand{\grq}{OT1}{%
2924 \save@sf@q{\kern-.0125em
2925 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2926 \kern.07em\relax}}
2927 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

\glqq The ‘german’ double quotes.

```

\grqq 2928 \ProvideTextCommandDefault{\glqq}{%
2929 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2930 \ProvideTextCommand{\grqq}{T1}{%
2931 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2932 \ProvideTextCommand{\grqq}{TU}{%
2933 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2934 \ProvideTextCommand{\grqq}{OT1}{%
2935 \save@sf@q{\kern-.07em
2936 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2937 \kern.07em\relax}}
2938 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}

```

\flq The ‘french’ single guillemets.

```

\frq 2939 \ProvideTextCommandDefault{\flq}{%
2940 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2941 \ProvideTextCommandDefault{\frq}{%
2942 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.
`\frqq`

```

2943 \ProvideTextCommandDefault{\flqq}{%
2944   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2945 \ProvideTextCommandDefault{\frqq}{%
2946   \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).

```

2947 \def\umlauthigh{%
2948   \def\bbl@umlauta##1{\leavevmode\bggroup%
2949     \expandafter\accent\csname\fontencoding dqpos\endcsname
2950     ##1\bbl@allowhyphens\egroup}%
2951   \let\bbl@umlaute\bbl@umlauta}
2952 \def\umlautlow{%
2953   \def\bbl@umlauta{\protect\lower@umlaut}}
2954 \def\umlautelow{%
2955   \def\bbl@umlaute{\protect\lower@umlaut}}
2956 \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.

```

2957 \expandafter\ifx\csname U@D\endcsname\relax
2958   \csname newdimen\endcsname\U@D
2959 \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.

```

2960 \def\lower@umlaut#1{%
2961   \leavevmode\bggroup
2962   \U@D 1ex%
2963   {\setbox\z@\hbox{%
2964     \expandafter\char\csname\fontencoding dqpos\endcsname}%
2965     \dimen@ -.45ex\advance\dimen@\ht\z@
2966     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2967   \expandafter\accent\csname\fontencoding dqpos\endcsname
2968   \fontdimen5\font\U@D #1%
2969   \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).

```

2970 \AtBeginDocument{%
```

```

2971 \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2972 \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2973 \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2974 \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
2975 \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2976 \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2977 \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2978 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2979 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2980 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2981 \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.

```

2982 \ifx\l@english\@undefined
2983 \chardef\l@english\z@
2984 \fi
2985 % The following is used to cancel rules in ini files (see Amharic).
2986 \ifx\l@unhyphenated\@undefined
2987 \newlanguage\l@unhyphenated
2988 \fi

```

9.13 Layout

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

```

2989 \bbl@trace{Bidi layout}
2990 \providecommand\IfBabelLayout[3]{#3}%
2991 \newcommand\BabelPatchSection[1]{%
2992   \@ifundefined{#1}{}{%
2993     \bbl@exp{\let\bbl@ss@#1>\<#1>}%
2994     \@namedef{#1}{%
2995       \ifstar\bbl@presec@#1}%
2996       {\@dblarg\bbl@presec@x{#1}}}%
2997 \def\bbl@presec@x#1[#2]#3{%
2998   \bbl@exp{%
2999     \\\select@language@x{\bbl@main@language}%
3000     \\\bbl@cs{sspre@#1}%
3001     \\\bbl@cs{ss@#1}%
3002     [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
3003     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
3004     \\\select@language@x{\languagename}}%
3005 \def\bbl@presec@#1#2{%
3006   \bbl@exp{%
3007     \\\select@language@x{\bbl@main@language}%
3008     \\\bbl@cs{sspre@#1}%
3009     \\\bbl@cs{ss@#1}*%
3010     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
3011     \\\select@language@x{\languagename}}%
3012 \IfBabelLayout{sectioning}%
3013   {\BabelPatchSection{part}%
3014     \BabelPatchSection{chapter}%
3015     \BabelPatchSection{section}%
3016     \BabelPatchSection{subsection}%
3017     \BabelPatchSection{subsubsection}%
3018     \BabelPatchSection{paragraph}%
3019     \BabelPatchSection{subparagraph}%
3020     \def\babel@toc#1{%
3021       \select@language@x{\bbl@main@language}}}%

```

```

3022 \IfBabelLayout{captions}%
3023 {\BabelPatchSection{caption}}{}

```

9.14 Load engine specific macros

```

3024 \bbl@trace{Input engine specific macros}
3025 \ifcase\bbl@engine
3026 \input txtbabel.def
3027 \or
3028 \input luababel.def
3029 \or
3030 \input xebabel.def
3031 \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.

```

3032 \bbl@trace{Creating languages and reading ini files}
3033 \newcommand\babelprovide[2][]{%
3034 \let\bbl@savelangname\language
3035 \edef\bbl@savelocaleid{\the\localeid}%
3036 % Set name and locale id
3037 \edef\language{#2}%
3038 % \global\@namedef\bbl@lcname{#2}{#2}%
3039 \bbl@id@assign
3040 \let\bbl@KVP@captions\@nil
3041 \let\bbl@KVP@date\@nil
3042 \let\bbl@KVP@import\@nil
3043 \let\bbl@KVP@main\@nil
3044 \let\bbl@KVP@script\@nil
3045 \let\bbl@KVP@language\@nil
3046 \let\bbl@KVP@hyphenrules\@nil
3047 \let\bbl@KVP@linebreaking\@nil
3048 \let\bbl@KVP@justification\@nil
3049 \let\bbl@KVP@mapfont\@nil
3050 \let\bbl@KVP@maparabic\@nil
3051 \let\bbl@KVP@mapdigits\@nil
3052 \let\bbl@KVP@intraspace\@nil
3053 \let\bbl@KVP@intrapenalty\@nil
3054 \let\bbl@KVP@onchar\@nil
3055 \let\bbl@KVP@transforms\@nil
3056 \global\let\bbl@release@transforms\@empty
3057 \let\bbl@KVP@alph\@nil
3058 \let\bbl@KVP@Alph\@nil
3059 \let\bbl@KVP@labels\@nil
3060 \bbl@csarg\let{KVP@labels*}\@nil
3061 \global\let\bbl@inidata\@empty
3062 \bbl@forkv{#1}{% TODO - error handling
3063 \in@{/}{##1}%
3064 \ifin@
3065 \bbl@renewinikey##1\@{##2}%
3066 \else
3067 \bbl@csarg\def{KVP@##1}{##2}%
3068 \fi}%
3069 % == init ==
3070 \ifx\bbl@screset\@undefined
3071 \bbl@ldfinit

```

```

3072 \fi
3073 % ==
3074 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3075 \bbl@ifunset{date#2}%
3076   {\let\bbl@lbkflag\@empty}% new
3077   {\ifx\bbl@KVP@hyphenrules\@nil\else
3078     \let\bbl@lbkflag\@empty
3079     \fi
3080     \ifx\bbl@KVP@import\@nil\else
3081       \let\bbl@lbkflag\@empty
3082     \fi}%
3083 % == import, captions ==
3084 \ifx\bbl@KVP@import\@nil\else
3085   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
3086   {\ifx\bbl@initoload\relax
3087     \begingroup
3088       \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
3089       \bbl@input@texini{#2}%
3090     \endgroup
3091   \else
3092     \xdef\bbl@KVP@import{\bbl@initoload}%
3093   \fi}%
3094 {}%
3095 \fi
3096 \ifx\bbl@KVP@captions\@nil
3097   \let\bbl@KVP@captions\bbl@KVP@import
3098 \fi
3099 % ==
3100 \ifx\bbl@KVP@transforms\@nil\else
3101   \bbl@replace\bbl@KVP@transforms{ }{,}%
3102 \fi
3103 % Load ini
3104 \bbl@ifunset{date#2}%
3105   {\bbl@provide@new{#2}}%
3106   {\bbl@ifblank{#1}%
3107     }% With \bbl@load@basic below
3108   {\bbl@provide@renew{#2}}%
3109 % Post tasks
3110 % -----
3111 % == ensure captions ==
3112 \ifx\bbl@KVP@captions\@nil\else
3113   \bbl@ifunset{\bbl@extracaps#2}%
3114     {\bbl@exp{\bbl@babelensure[exclude=\today]{#2}}}%
3115     {\toks@ \expandafter \expandafter \expandafter
3116       {\csname bbl@extracaps@#2\endcsname}%
3117       \bbl@exp{\bbl@babelensure[exclude=\today,include=\the\toks@]{#2}}%
3118     \bbl@ifunset{\bbl@ensure@language}%
3119     {\bbl@exp%
3120       \\\DeclareRobustCommand\<bbl@ensure@language>[1]{%
3121         \\\foreignlanguage{language}%
3122         {###1}}}%
3123     }%
3124   \bbl@exp%
3125     \\\bbl@tglobal\<bbl@ensure@language>%
3126     \\\bbl@tglobal\<bbl@ensure@language\space>%
3127 \fi
3128 % ==
3129 % At this point all parameters are defined if 'import'. Now we
3130 % execute some code depending on them. But what about if nothing was

```

```

3131 % imported? We just set the basic parameters, but still loading the
3132 % whole ini file.
3133 \bbl@load@basic{#2}%
3134 % == script, language ==
3135 % Override the values from ini or defines them
3136 \ifx\bbl@KVP@script\@nil\else
3137   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
3138 \fi
3139 \ifx\bbl@KVP@language\@nil\else
3140   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
3141 \fi
3142 % == onchar ==
3143 \ifx\bbl@KVP@onchar\@nil\else
3144   \bbl@luahyphenate
3145   \directlua{
3146     if Babel.locale_mapped == nil then
3147       Babel.locale_mapped = true
3148       Babel.linebreaking.add_before(Babel.locale_map)
3149       Babel.loc_to_scr = {}
3150       Babel.chr_to_loc = Babel.chr_to_loc or {}
3151     end}%
3152   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3153   \ifin@
3154     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
3155       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
3156     \fi
3157     \bbl@exp{\bbl@add\bbl@starthyphens
3158       {\bbl@patterns@lua{languagename}}}%
3159     % TODO - error/warning if no script
3160     \directlua{
3161       if Babel.script_blocks['\bbl@cl{sbc}'] then
3162         Babel.loc_to_scr[\the\localeid] =
3163           Babel.script_blocks['\bbl@cl{sbc}']
3164         Babel.locale_props[\the\localeid].lc = \the\localeid\space
3165         Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@languagename}\space
3166       end
3167     }%
3168   \fi
3169   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3170   \ifin@
3171     \bbl@ifunset{bbl@lsys{languagename}}{\bbl@provide@lsys{languagename}}{}%
3172     \bbl@ifunset{bbl@wdir{languagename}}{\bbl@provide@dirs{languagename}}{}%
3173     \directlua{
3174       if Babel.script_blocks['\bbl@cl{sbc}'] then
3175         Babel.loc_to_scr[\the\localeid] =
3176           Babel.script_blocks['\bbl@cl{sbc}']
3177       end}%
3178     \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
3179       \AtBeginDocument{%
3180         \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}%
3181         {\selectfont}}%
3182       \def\bbl@mapselect{%
3183         \let\bbl@mapselect\relax
3184         \edef\bbl@prefontid{\fontid\font}%
3185       \def\bbl@mapdir##1{%
3186         {\def\languagename{##1}%
3187           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3188           \bbl@switchfont
3189           \directlua{

```

```

3190         Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
3191         ['\bbl@prefontid'] = \fontid\font\space}}}%
3192     \fi
3193     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3194     \fi
3195     % TODO - catch non-valid values
3196 \fi
3197 % == mapfont ==
3198 % For bidi texts, to switch the font based on direction
3199 \ifx\bbl@KVP@mapfont\@nil\else
3200     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
3201     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
3202         mapfont. Use 'direction'.%
3203         {See the manual for details.}}}%
3204     \bbl@ifunset{\bbl@sys@\language}{\bbl@provide@sys{\language}}}%
3205     \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3206     \ifx\bbl@mapselect\@undefined % TODO. See onchar
3207         \AtBeginDocument{%
3208             \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3209             {\selectfont}}%
3210         \def\bbl@mapselect{%
3211             \let\bbl@mapselect\relax
3212             \edef\bbl@prefontid{\fontid\font}}%
3213         \def\bbl@mapdir##1{%
3214             {\def\language{##1}%
3215             \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3216             \bbl@switchfont
3217             \directlua{Babel.fontmap
3218                 [\the\csname bbl@wdir@@##1\endcsname]%
3219                 [\bbl@prefontid]=\fontid\font}}}%
3220         \fi
3221         \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3222     \fi
3223 % == Line breaking: intraspace, intrapenalty ==
3224 % For CJK, East Asian, Southeast Asian, if interspace in ini
3225 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3226     \bbl@csarg\edef{intsp#2}{\bbl@KVP@intraspace}%
3227     \fi
3228     \bbl@provide@intraspace
3229     %
3230     \ifx\bbl@KVP@justification\@nil\else
3231         \let\bbl@KVP@linebreaking\bbl@KVP@justification
3232     \fi
3233     \ifx\bbl@KVP@linebreaking\@nil\else
3234         \bbl@xin{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3235         \ifin@
3236             \bbl@csarg\xdef
3237             {\lnbrk@\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
3238         \fi
3239     \fi
3240     \bbl@xin{/e}{/\bbl@cl{\lnbrk}}%
3241     \ifin@else\bbl@xin{/k}{/\bbl@cl{\lnbrk}}\fi
3242     \ifin@\bbl@arabicjust\fi
3243 % == Line breaking: hyphenate.other.locale/.script==
3244 \ifx\bbl@lbfkflag\@empty
3245     \bbl@ifunset{\bbl@hyotl@\language}{}%
3246     {\bbl@csarg\bbl@replace{hyotl@\language}{ }{,}%
3247     \bbl@startcommands*\language}%
3248     \bbl@csarg\bbl@foreach{hyotl@\language}{%

```

```

3249 \ifcase\bbbl@engine
3250 \ifnum##1<257
3251 \SetHyphenMap{\BabelLower{##1}{##1}}%
3252 \fi
3253 \else
3254 \SetHyphenMap{\BabelLower{##1}{##1}}%
3255 \fi}%
3256 \bbbl@endcommands}%
3257 \bbbl@ifunset{\bbbl@hyots@\languagename}{}%
3258 {\bbbl@csarg\bbbl@replace{\hyots@\languagename}{ }{,}}%
3259 \bbbl@csarg\bbbl@foreach{\hyots@\languagename}{%
3260 \ifcase\bbbl@engine
3261 \ifnum##1<257
3262 \global\lccode##1=##1\relax
3263 \fi
3264 \else
3265 \global\lccode##1=##1\relax
3266 \fi}}}%
3267 \fi
3268 % == Counters: maparabic ==
3269 % Native digits, if provided in ini (TeX level, xe and lua)
3270 \ifcase\bbbl@engine\else
3271 \bbbl@ifunset{\bbbl@dgnat@\languagename}{}%
3272 {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
3273 \expandafter\expandafter\expandafter
3274 \bbbl@setdigits\csname bbl@dgnat@\languagename\endcsname
3275 \ifx\bbbl@KVP@maparabic\@nil\else
3276 \ifx\bbbl@latinarabic\@undefined
3277 \expandafter\let\expandafter\@arabic
3278 \csname bbl@counter@\languagename\endcsname
3279 \else % ie, if layout=counters, which redefines \@arabic
3280 \expandafter\let\expandafter\bbbl@latinarabic
3281 \csname bbl@counter@\languagename\endcsname
3282 \fi
3283 \fi
3284 \fi}%
3285 \fi
3286 % == Counters: mapdigits ==
3287 % Native digits (lua level).
3288 \ifodd\bbbl@engine
3289 \ifx\bbbl@KVP@mapdigits\@nil\else
3290 \bbbl@ifunset{\bbbl@dgnat@\languagename}{}%
3291 {\RequirePackage{luatexbase}}%
3292 \bbbl@activate@preotf
3293 \directlua{
3294 Babel = Babel or {} %%% -> presets in luababel
3295 Babel.digits_mapped = true
3296 Babel.digits = Babel.digits or {}
3297 Babel.digits[\the\localeid] =
3298 table.pack(string.utfvalue('\bbbl@cl{dgnat}'))
3299 if not Babel.numbers then
3300 function Babel.numbers(head)
3301 local LOCALE = luatexbase.registernumber'\bbbl@attr@locale'
3302 local GLYPH = node.id'glyph'
3303 local inmath = false
3304 for item in node.traverse(head) do
3305 if not inmath and item.id == GLYPH then
3306 local temp = node.get_attribute(item, LOCALE)
3307 if Babel.digits[temp] then

```

```

3308         local chr = item.char
3309         if chr > 47 and chr < 58 then
3310             item.char = Babel.digits[temp][chr-47]
3311         end
3312     end
3313     elseif item.id == node.id'math' then
3314         inmath = (item.subtype == 0)
3315     end
3316 end
3317 return head
3318 end
3319 end
3320 }}%
3321 \fi
3322 \fi
3323 % == Counters: alph, Alph ==
3324 % What if extras<lang> contains a \babel@save\@alph? It won't be
3325 % restored correctly when exiting the language, so we ignore
3326 % this change with the \bbl@alph@saved trick.
3327 \ifx\bbl@KVP@alph\@nil\else
3328     \toks@\expandafter\expandafter\expandafter{%
3329         \csname extras\language\endcsname}%
3330     \bbl@exp{%
3331         \def<extras\language>{%
3332             \let\\bbl@alph@saved\\@alph
3333             \the\toks@
3334             \let\\@alph\\bbl@alph@saved
3335             \\babel@save\\@alph
3336             \let\\@alph<bbl@cntr@bbl@KVP@alph @\language>}}%
3337 \fi
3338 \ifx\bbl@KVP@Alph\@nil\else
3339     \toks@\expandafter\expandafter\expandafter{%
3340         \csname extras\language\endcsname}%
3341     \bbl@exp{%
3342         \def<extras\language>{%
3343             \let\\bbl@Alph@saved\\@Alph
3344             \the\toks@
3345             \let\\@Alph\\bbl@Alph@saved
3346             \\babel@save\\@Alph
3347             \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language>}}%
3348 \fi
3349 % == require.babel in ini ==
3350 % To load or reload the babel-*.tex, if require.babel in ini
3351 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3352     \bbl@ifunset{bbl@rqtex@\language}{}%
3353     {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3354         \let\BabelBeforeIni\@gobbletwo
3355         \chardef\atcatcode=\catcode`\@
3356         \catcode`\@=11\relax
3357         \bbl@input@texini{\bbl@cs{rqtex@\language}}%
3358         \catcode`\@=\atcatcode
3359         \let\atcatcode\relax
3360     \fi}%
3361 \fi
3362 % == Release saved transforms ==
3363 \bbl@release@transforms\relax % \relax closes the last item.
3364 % == main ==
3365 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3366     \let\language\bbl@savelangname

```

```

3367 \chardef\localeid\bb1@savelocaleid\relax
3368 \fi}

```

Depending on whether or not the language exists, we define two macros.

```

3369 \def\bb1@provide@new#1{%
3370 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3371 \@namedef{extras#1}{}%
3372 \@namedef{noextras#1}{}%
3373 \bb1@startcommands*{#1}{captions}%
3374 \ifx\bb1@KVP@captions\@nil % and also if import, implicit
3375 \def\bb1@tempb##1{% elt for \bb1@captionslist
3376 \ifx##1\@empty\else
3377 \bb1@exp{%
3378 \\\SetString\\##1{%
3379 \\\bb1@nocaption{\bb1@stripslash##1}{#1\bb1@stripslash##1}}}%
3380 \expandafter\bb1@tempb
3381 \fi}%
3382 \expandafter\bb1@tempb\bb1@captionslist\@empty
3383 \else
3384 \ifx\bb1@initoload\relax
3385 \bb1@read@ini{\bb1@KVP@captions}2% % Here letters cat = 11
3386 \else
3387 \bb1@read@ini{\bb1@initoload}2% % Same
3388 \fi
3389 \fi
3390 \StartBabelCommands*{#1}{date}%
3391 \ifx\bb1@KVP@import\@nil
3392 \bb1@exp{%
3393 \\\SetString\\today{\\bb1@nocaption{today}{#1today}}}%
3394 \else
3395 \bb1@savetoday
3396 \bb1@savestate
3397 \fi
3398 \bb1@endcommands
3399 \bb1@load@basic{#1}%
3400 % == hyphenmins == (only if new)
3401 \bb1@exp{%
3402 \gdef\<#1hyphenmins>{%
3403 {\bb1@ifunset{\bb1@lfthm@#1}{2}{\bb1@cs{lfthm@#1}}}%
3404 {\bb1@ifunset{\bb1@rgthm@#1}{3}{\bb1@cs{rgthm@#1}}}%
3405 % == hyphenrules ==
3406 \bb1@provide@hyphens{#1}%
3407 % == frenchspacing == (only if new)
3408 \bb1@ifunset{\bb1@frspc@#1}{}%
3409 {\edef\bb1@tempa{\bb1@cl{frspc}}}%
3410 \edef\bb1@tempa{\expandafter\@car\bb1@tempa\@nil}%
3411 \if u\bb1@tempa % do nothing
3412 \else\if n\bb1@tempa % non french
3413 \expandafter\bb1@add\csname extras#1\endcsname{%
3414 \let\bb1@elt\bb1@fs@elt@i
3415 \bb1@fs@chars}%
3416 \else\if y\bb1@tempa % french
3417 \expandafter\bb1@add\csname extras#1\endcsname{%
3418 \let\bb1@elt\bb1@fs@elt@ii
3419 \bb1@fs@chars}%
3420 \fi\fi\fi}%
3421 %
3422 \ifx\bb1@KVP@main\@nil\else
3423 \expandafter\main@language\expandafter{#1}%

```

```

3424 \fi}
3425 % A couple of macros used above, to avoid hashes #####...
3426 \def\bbl@fs@elt@i#1#2#3{%
3427   \ifnum\sfcode`#1=#2\relax
3428     \babel@savevariable{\sfcode`#1}%
3429     \sfcode`#1=#3\relax
3430   \fi}%
3431 \def\bbl@fs@elt@ii#1#2#3{%
3432   \ifnum\sfcode`#1=#3\relax
3433     \babel@savevariable{\sfcode`#1}%
3434     \sfcode`#1=#2\relax
3435   \fi}%
3436 %
3437 \def\bbl@provide@renew#1{%
3438   \ifx\bbl@KVP@captions\@nil\else
3439     \StartBabelCommands*{#1}{captions}%
3440     \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
3441     \EndBabelCommands
3442   \fi
3443   \ifx\bbl@KVP@import\@nil\else
3444     \StartBabelCommands*{#1}{date}%
3445     \bbl@savetoday
3446     \bbl@savedate
3447     \EndBabelCommands
3448   \fi
3449   % == hyphenrules ==
3450   \ifx\bbl@lbfkflag\@empty
3451     \bbl@provide@hyphens{#1}%
3452   \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.)

```

3453 \def\bbl@load@basic#1{%
3454   \bbl@ifunset{bbl@inidata@\language}\relax
3455   {\getlocaleproperty\bbl@tempa{\language}{identification/load.level}%
3456   \ifcase\bbl@tempa
3457     \bbl@csarg\let{lname@\language}\relax
3458   \fi}%
3459   \bbl@ifunset{bbl@lname@#1}%
3460   {\def\BabelBeforeIni##1##2{%
3461     \begingroup
3462       \let\bbl@ini@captions@aux\@gobbletwo
3463       \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
3464       \bbl@read@ini{##1}1%
3465       \ifx\bbl@initoload\relax\endinput\fi
3466     \endgroup}%
3467     \begingroup           % boxed, to avoid extra spaces:
3468       \ifx\bbl@initoload\relax
3469         \bbl@input@texini{##1}%
3470       \else
3471         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3472       \fi
3473     \endgroup}%
3474   {}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3475 \def\bbl@provide@hyphens#1{%
3476   \let\bbl@tempa\relax

```

```

3477 \ifx\bb1@KVP@hyphenrules\@nil\else
3478   \bb1@replace\bb1@KVP@hyphenrules{ }{,}%
3479   \bb1@foreach\bb1@KVP@hyphenrules{%
3480     \ifx\bb1@tempa\relax % if not yet found
3481       \bb1@ifsamestring{##1}{+}%
3482       {\bb1@exp{\addlanguage\<l@##1>}}}%
3483       {}%
3484       \bb1@ifunset{l@##1}%
3485       {}%
3486       {\bb1@exp{\let\bb1@tempa\<l@##1>}}}%
3487   \fi}%
3488 \fi
3489 \ifx\bb1@tempa\relax % if no opt or no language in opt found
3490   \ifx\bb1@KVP@import\@nil
3491     \ifx\bb1@initoload\relax\else
3492       \bb1@exp{% and hyphenrules is not empty
3493         \\bb1@ifblank{\bb1@cs{hyphr@#1}}%
3494         {}%
3495         {\let\\bb1@tempa\<l@bb1@cl{hyphr}>}}}%
3496     \fi
3497   \else % if importing
3498     \bb1@exp{% and hyphenrules is not empty
3499       \\bb1@ifblank{\bb1@cs{hyphr@#1}}%
3500       {}%
3501       {\let\\bb1@tempa\<l@bb1@cl{hyphr}>}}}%
3502   \fi
3503 \fi
3504 \bb1@ifunset{bb1@tempa}% ie, relax or undefined
3505   {\bb1@ifunset{l@##1}% no hyphenrules found - fallback
3506     {\bb1@exp{\adddialect\<l@##1>\language}}%
3507     {}% so, l@<lang> is ok - nothing to do
3508     {\bb1@exp{\adddialect\<l@##1>\bb1@tempa}}}% found in opt list or ini

```

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

```

3509 \def\bb1@input@texini#1{%
3510   \bb1@bsphack
3511   \bb1@exp{%
3512     \catcode\=\14 \catcode\=0
3513     \catcode\={1 \catcode\}=2
3514     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
3515     \catcode\=\the\catcode\relax
3516     \catcode\=\the\catcode\relax
3517     \catcode\={\the\catcode\relax
3518     \catcode\}=the\catcode\relax}%
3519   \bb1@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 \bb1@read@ini.

```

3520 \def\bb1@inline#1\bb1@inline{%
3521   \@ifnextchar[\bb1@inisect{\@ifnextchar;\bb1@iniskip\bb1@inistore}#1\@@% ]
3522 \def\bb1@inisect[#1]#2\@@{\def\bb1@section{#1}}%
3523 \def\bb1@iniskip#1\@@{% if starts with ;
3524 \def\bb1@inistore#1=#2\@@{% full (default)
3525   \bb1@trim@def\bb1@tempa{#1}%
3526   \bb1@trim\toks@{#2}%
3527   \bb1@ifunset{bb1@KVP@\bb1@section/\bb1@tempa}%
3528   {\bb1@exp{%
3529     \g@addto@macro\\bb1@inidata{%

```

```

3530      \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3531    {}}%
3532 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
3533   \bbl@trim@def\bbl@tempa{#1}%
3534   \bbl@trim\toks@{#2}%
3535   \bbl@xin@{.identification.}{.\bbl@section.}%
3536   \ifin@
3537     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
3538       \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3539   \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.

```

3540 \ifx\bbl@readstream\@undefined
3541   \csname newread\endcsname\bbl@readstream
3542 \fi
3543 \def\bbl@read@ini#1#2{%
3544   \openin\bbl@readstream=babel-#1.ini
3545   \ifeof\bbl@readstream
3546     \bbl@error
3547     {There is no ini file for the requested language\\%
3548      (#1). Perhaps you misspelled it or your installation\\%
3549      is not complete.}%
3550     {Fix the name or reinstall babel.}%
3551   \else
3552     % Store ini data in \bbl@inidata
3553     \catcode\ [=12 \catcode\ ]=12 \catcode\ ~=12 \catcode\ &=12
3554     \catcode\ ;=12 \catcode\ |=12 \catcode\ %=14 \catcode\ -=12
3555     \bbl@info{Importing
3556               \ifcase#2font and identification \or basic \fi
3557               data for \language\\%
3558               from babel-#1.ini. Reported}%
3559     \ifnum#2=\z@
3560       \global\let\bbl@inidata\@empty
3561       \let\bbl@inistore\bbl@inistore@min % Remember it's local
3562     \fi
3563     \def\bbl@section{identification}%
3564     \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
3565     \bbl@inistore load.level=#2\@@
3566     \loop
3567     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3568       \endlinechar\m@ne
3569       \read\bbl@readstream to \bbl@line
3570       \endlinechar\^^M
3571       \ifx\bbl@line\@empty\else
3572         \expandafter\bbl@inline\bbl@line\bbl@inline
3573       \fi
3574     \repeat
3575     % Process stored data
3576     \bbl@csarg\xdef{lini@\language}{#1}%
3577     \let\bbl@savestrings\@empty
3578     \let\bbl@savetoday\@empty
3579     \let\bbl@savestate\@empty
3580     \def\bbl@elt##1##2##3{%
3581       \def\bbl@section{##1}%

```

```

3582 \in@{=date.}{=##1}% Find a better place
3583 \ifin@
3584 \bbl@ini@calendar{##1}%
3585 \fi
3586 \global\bbl@csarg\let\bbl@KVP@##1/##2\relax
3587 \bbl@ifunset\bbl@inikv@##1}{}%
3588 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
3589 \bbl@inidata
3590 % 'Export' data
3591 \bbl@ini@exports{#2}%
3592 \global\bbl@csarg\let\inidata@\languagename\bbl@inidata
3593 \global\let\bbl@inidata\@empty
3594 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
3595 \bbl@tglobal\bbl@ini@loaded
3596 \fi}

```

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

```

3597 \def\bbl@ini@calendar#1{%
3598 \lowercase{\def\bbl@tempa{=1=}}%
3599 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3600 \bbl@replace\bbl@tempa{=date.}{}%
3601 \in@{.licr=}{#1=}%
3602 \ifin@
3603 \ifcase\bbl@engine
3604 \bbl@replace\bbl@tempa{.licr=}{}%
3605 \else
3606 \let\bbl@tempa\relax
3607 \fi
3608 \fi
3609 \ifx\bbl@tempa\relax\else
3610 \bbl@replace\bbl@tempa{=}{}%
3611 \bbl@exp{%
3612 \def<\bbl@inikv@#1>####1####2{%
3613 \bbl@inidate####1...\relax{####2}{\bbl@tempa}}%
3614 \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).

```

3615 \def\bbl@renewinikey#1/#2\@#3{%
3616 \edef\bbl@tempa{\zap@space #1 \@empty}% section
3617 \edef\bbl@tempb{\zap@space #2 \@empty}% key
3618 \bbl@trim\toks@{#3}% value
3619 \bbl@exp{%
3620 \global\let<\bbl@KVP@\bbl@tempa/\bbl@tempb>\@empty % just a flag
3621 \g@addto@macro\bbl@inidata{%
3622 \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.

```

3623 \def\bbl@exportkey#1#2#3{%
3624 \bbl@ifunset\bbl@kv@#2}%
3625 {\bbl@csarg\gdef{#1\languagename}{#3}}%
3626 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3627 \bbl@csarg\gdef{#1\languagename}{#3}}%
3628 \else
3629 \bbl@exp{\global\let<\bbl@#1\languagename>\bbl@kv@#2}%
3630 \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.

```

3631 \def\bbl@iniwarning#1{%
3632   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3633   {\bbl@warning{%
3634     From babel-\bbl@cs{lini@language}\language\ini:\%
3635     \bbl@cs{@kv@identification.warning#1}\%
3636     Reported }}%
3637 %
3638 \let\bbl@release@transforms\@empty
3639 %
3640 \def\bbl@ini@exports#1{%
3641   % Identification always exported
3642   \bbl@iniwarning{%
3643     \ifcase\bbl@engine
3644       \bbl@iniwarning{.pdflatex}%
3645     \or
3646       \bbl@iniwarning{.lualatex}%
3647     \or
3648       \bbl@iniwarning{.xelatex}%
3649     \fi%
3650     \bbl@exportkey{elname}{identification.name.english}{}%
3651     \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3652       {\csname bbl@elname@language\endcsname}}%
3653     \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
3654     \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
3655     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3656     \bbl@exportkey{esname}{identification.script.name}{}%
3657     \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3658       {\csname bbl@esname@language\endcsname}}%
3659     \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
3660     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3661     % Also maps bcp47 -> language
3662     \ifbbl@bcptoname
3663       \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
3664     \fi
3665     % Conditional
3666     \ifnum#1>\z@      % 0 = only info, 1, 2 = basic, (re)new
3667       \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3668       \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3669       \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3670       \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3671       \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3672       \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3673       \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3674       \bbl@exportkey{intsp}{typography.intraspaces}{}%
3675       \bbl@exportkey{chrng}{characters.ranges}{}%
3676       \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3677       \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3678       \ifnum#1=\tw@    % only (re)new
3679         \bbl@exportkey{rtex}{identification.require.babel}{}%
3680         \bbl@toggle\bbl@savetoday
3681         \bbl@toggle\bbl@savestate
3682         \bbl@savestrings
3683       \fi
3684     \fi}

```

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

```

3685 \def\bbl@inikv#1#2{%      key=value
3686 \toks@{#2}%               This hides #'s from ini values
3687 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3688 \let\bbl@inikv@identification\bbl@inikv
3689 \let\bbl@inikv@typography\bbl@inikv
3690 \let\bbl@inikv@characters\bbl@inikv
3691 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3692 \def\bbl@inikv@counters#1#2{%
3693 \bbl@ifsamestring{#1}{digits}%
3694 {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3695             decimal digits}%
3696 {Use another name.}}%
3697 }%
3698 \def\bbl@tempc{#1}%
3699 \bbl@trim@def{\bbl@tempb*}{#2}%
3700 \in@{.1$}{#1$}%
3701 \ifin@
3702 \bbl@replace\bbl@tempc{.1}{}%
3703 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3704 \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3705 \fi
3706 \in@{.F.}{#1}%
3707 \ifin@else\in@{.S.}{#1}\fi
3708 \ifin@
3709 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3710 \else
3711 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3712 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3713 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3714 \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.

```

3715 \ifcase\bbl@engine
3716 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3717 \bbl@ini@captions@aux{#1}{#2}}
3718 \else
3719 \def\bbl@inikv@captions#1#2{%
3720 \bbl@ini@captions@aux{#1}{#2}}
3721 \fi

```

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

```

3722 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3723 \bbl@replace\bbl@tempa{.template}{}%
3724 \def\bbl@toreplace{#1}{}%
3725 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3726 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3727 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3728 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3729 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3730 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3731 \ifin@

```

```

3732 \nameuse{bbl@patch\bbl@tempa}%
3733 \global\bbl@csarg\let\bbl@tempa fmt@#2\bbl@toreplace
3734 \fi
3735 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3736 \ifin@
3737 \toks@{\expandafter\bbl@toreplace}%
3738 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3739 \fi}
3740 \def\bbl@ini@captions@aux#1#2{%
3741 \bbl@trim@def\bbl@tempa{#1}%
3742 \bbl@xin@{.template}{\bbl@tempa}%
3743 \ifin@
3744 \bbl@ini@captions@template{#2}\language\name
3745 \else
3746 \bbl@ifblank{#2}%
3747 {\bbl@exp{%
3748 \toks@{\bbl@nocaption{\bbl@tempa}\language\bbl@tempa name}}}%
3749 {\bbl@trim\toks@{#2}}%
3750 \bbl@exp{%
3751 \bbl@add\bbl@savestrings{%
3752 \SetString\<\bbl@tempa name>\the\toks@}}%
3753 \toks@{\expandafter\bbl@captionslist}%
3754 \bbl@exp{\in@{\<\bbl@tempa name>\the\toks@}}%
3755 \ifin@ \else
3756 \bbl@exp{%
3757 \bbl@add\<bbl@extracaps@\language\name>\<\bbl@tempa name>%
3758 \bbl@toggle\bbl@extracaps@\language\name>%
3759 \fi
3760 \fi}

```

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

```

3761 \def\bbl@list@the{%
3762 part,chapter,section,subsection,subsubsection,paragraph,%
3763 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3764 table,page,footnote,mpfootnote,mpfn}
3765 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3766 \bbl@ifunset{bbl@map@#1\language}%
3767 {\nameuse{#1}}%
3768 {\nameuse{bbl@map@#1\language}}}
3769 \def\bbl@inikv@labels#1#2{%
3770 \in@{.map}{#1}%
3771 \ifin@
3772 \ifx\bbl@KVP@labels\@nil\else
3773 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3774 \ifin@
3775 \def\bbl@tempc{#1}%
3776 \bbl@replace\bbl@tempc{.map}{}%
3777 \in@{,#2,}{,arabic,roman,Roman,alpha,Alpha,fnsymbol,}%
3778 \bbl@exp{%
3779 \gdef\<bbl@map@\bbl@tempc @\language>%
3780 {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3781 \bbl@foreach\bbl@list@the{%
3782 \bbl@ifunset{the##1}{}%
3783 {\bbl@exp{\let\bbl@tempd\<the##1>}}%
3784 \bbl@exp{%
3785 \bbl@sreplace\<the##1>%
3786 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3787 \bbl@sreplace\<the##1>%
3788 {\<\@empty @\bbl@tempc>\<c##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%

```

```

3789         \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3790         \toks@ \expandafter\expandafter\expandafter{%
3791             \csname the##1\endcsname}%
3792         \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3793         \fi}}%
3794     \fi
3795 \fi
3796 %
3797 \else
3798 %
3799 % The following code is still under study. You can test it and make
3800 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3801 % language dependent.
3802 \in@{enumerate.}{#1}%
3803 \ifin@
3804     \def\bbl@tempa{#1}%
3805     \bbl@replace\bbl@tempa{enumerate.}{}%
3806     \def\bbl@toreplace{#2}%
3807     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3808     \bbl@replace\bbl@toreplace{{}}{\csname the}%
3809     \bbl@replace\bbl@toreplace{}}{\endcsname{}}%
3810     \toks@ \expandafter{\bbl@toreplace}%
3811     \bbl@exp{%
3812         \\bbl@add\<extras\language>{%
3813             \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3814             \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3815             \\bbl@tglobal\<extras\language>}%
3816     \fi
3817 \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.

```

3818 \def\bbl@chapttype{chapter}
3819 \ifx\@makechapterhead\undefined
3820     \let\bbl@patchchapter\relax
3821 \else\ifx\thechapter\undefined
3822     \let\bbl@patchchapter\relax
3823 \else\ifx\ps@headings\undefined
3824     \let\bbl@patchchapter\relax
3825 \else
3826     \def\bbl@patchchapter{%
3827         \global\let\bbl@patchchapter\relax
3828         \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3829         \bbl@tglobal\appendix
3830         \bbl@sreplace\ps@headings
3831             {\@chapapp\ thechapter}%
3832             {\bbl@chapterformat}%
3833         \bbl@tglobal\ps@headings
3834         \bbl@sreplace\chaptermark
3835             {\@chapapp\ thechapter}%
3836             {\bbl@chapterformat}%
3837         \bbl@tglobal\chaptermark
3838         \bbl@sreplace\@makechapterhead
3839             {\@chapapp\space\thechapter}%
3840             {\bbl@chapterformat}%
3841         \bbl@tglobal\@makechapterhead
3842         \gdef\bbl@chapterformat{%

```

```

3843 \bbl@ifunset{\bbl@chapttype fmt@language}%
3844 {\chapapp\space\thechapter}
3845 {\@nameuse{\bbl@chapttype fmt@language}}}}
3846 \let\bbl@patchappendix\bbl@patchchapter
3847 \fi\fi\fi
3848 \ifx\@part\undefined
3849 \let\bbl@patchpart\relax
3850 \else
3851 \def\bbl@patchpart{%
3852 \global\let\bbl@patchpart\relax
3853 \bbl@sreplace\@part
3854 {\partname\nobreakspace\thepart}%
3855 {\bbl@partformat}%
3856 \bbl@tglobal\@part
3857 \gdef\bbl@partformat{%
3858 \bbl@ifunset{\bbl@partfmt@language}%
3859 {\partname\nobreakspace\thepart}
3860 {\@nameuse{\bbl@partfmt@language}}}}
3861 \fi

```

Date. TODO. Document

```

3862 % Arguments are _not_ protected.
3863 \let\bbl@calendar\@empty
3864 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3865 \def\bbl@localedate#1#2#3#4{%
3866 \begingroup
3867 \ifx\@empty#1\@empty\else
3868 \let\bbl@ld@calendar\@empty
3869 \let\bbl@ld@variant\@empty
3870 \edef\bbl@tempa{\zap@space#1 \@empty}%
3871 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld##1}{##2}}%
3872 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3873 \edef\bbl@calendar{%
3874 \bbl@ld@calendar
3875 \ifx\bbl@ld@variant\@empty\else
3876 .\bbl@ld@variant
3877 \fi}%
3878 \bbl@replace\bbl@calendar{gregorian}{}%
3879 \fi
3880 \bbl@cased
3881 {\@nameuse{\bbl@date@language @\bbl@calendar}{#2}{#3}{#4}}%
3882 \endgroup}
3883 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3884 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3885 \bbl@trim@def\bbl@tempa{#1.#2}%
3886 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3887 {\bbl@trim@def\bbl@tempa{#3}%
3888 \bbl@trim\toks@{#5}%
3889 \@temptokena\expandafter{\bbl@savestate}%
3890 \bbl@exp{% Reverse order - in ini last wins
3891 \def\\bbl@savestate{%
3892 \\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}%
3893 \the\@temptokena}}%
3894 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3895 {\lowercase{\def\bbl@tempb{#6}}%
3896 \bbl@trim@def\bbl@toreplace{#5}%
3897 \bbl@TG@date
3898 \bbl@ifunset{\bbl@date@language @}%
3899 {\global\bbl@csarg\let{date@language @}\bbl@toreplace

```

```

3900      % TODO. Move to a better place.
3901      \bbl@exp{%
3902          \gdef<\language name date>{\protect<\language name date >}%
3903          \gdef<\language name date >####1####2####3{%
3904              \bbl@usedategroupttrue
3905              \<bbl@ensure@language name>{%
3906                  \\\localedate{####1}{####2}{####3}}}%
3907              \bbl@add\bbl@savetoday{%
3908                  \\\SetString\\today{%
3909                      \<\language name date>%
3910                      {\the\year}{\the\month}{\the\day}}}%
3911              }%
3912          \ifx\bbl@tempb\@empty\else
3913              \global\bbl@csarg\let{date@language name @bbl@tempb}\bbl@toreplace
3914              \fi}%
3915          {}%

```

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.

```

3916 \let\bbl@calendar\@empty
3917 \newcommand\BabelDateSpace{\nobreakspace}
3918 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3919 \newcommand\BabelDated[1]{\number#1}
3920 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3921 \newcommand\BabelDateM[1]{\number#1}
3922 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3923 \newcommand\BabelDateMMMM[1]{%
3924     \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3925 \newcommand\BabelDatey[1]{\number#1}%
3926 \newcommand\BabelDateyy[1]{%
3927     \ifnum#1<10 0\number#1 %
3928     \else\ifnum#1<100 \number#1 %
3929     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3930     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3931     \else
3932         \bbl@error
3933         {Currently two-digit years are restricted to the\
3934         range 0-9999.}%
3935         {There is little you can do. Sorry.}%
3936     \fi\fi\fi\fi}%
3937 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3938 \def\bbl@replace@finish@iii#1{%
3939     \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
3940 \def\bbl@TG@date{%
3941     \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3942     \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot}}%
3943     \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3944     \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3945     \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3946     \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3947     \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3948     \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3949     \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3950     \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3951     \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3952     \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3953     \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3954 % Note after \bbl@replace \toks@ contains the resulting string.

```

```

3955% TODO - Using this implicit behavior doesn't seem a good idea.
3956 \bbl@replace@finish@iii\bbl@toreplace}
3957 \def\bbl@datecncr{\expandafter\bbl@xdatecncr\expandafter}
3958 \def\bbl@xdatecncr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3959 \let\bbl@release@transforms\@empty
3960 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3961 \bbl@transforms\babelprehyphenation}
3962 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3963 \bbl@transforms\babelposthyphenation}
3964 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3965 \begingroup
3966 \catcode`\%=12
3967 \catcode`\&=14
3968 \gdef\bbl@transforms#1#2#3{&%
3969 \ifx\bbl@KVP@transforms\@nil\else
3970 \directlua{
3971 str = [=[#2]=]
3972 str = str:gsub('%.%d+%.%d+$', '')
3973 tex.print([[ \def\string\babeltempa{]] .. str .. [ ]]])
3974 }&%
3975 \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&%
3976 \ifin@
3977 \in@{.0$}{#2$}&%
3978 \ifin@
3979 \g@addto@macro\bbl@release@transforms{&%
3980 \relax\bbl@transforms@aux#1{\language name}{#3}}&%
3981 \else
3982 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3983 \fi
3984 \fi
3985 \fi}
3986 \endgroup

```

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

```

3987 \def\bbl@provide@lsys#1{%
3988 \bbl@ifunset{bbl@lname@#1}%
3989 {\bbl@load@info{#1}}%
3990 }%
3991 \bbl@csarg\let{lsys@#1}\@empty
3992 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3993 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3994 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3995 \bbl@ifunset{bbl@lname@#1}{%
3996 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3997 \ifcase\bbl@engine\or\or
3998 \bbl@ifunset{bbl@prehc@#1}{}%
3999 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
4000 }%
4001 {\ifx\bbl@xenohyph\@undefined
4002 \let\bbl@xenohyph\bbl@xenohyph@d
4003 \ifx\AtBeginDocument\@notprerr
4004 \expandafter\@secondoftwo % to execute right now
4005 \fi
4006 \AtBeginDocument{%
4007 \expandafter\bbl@add
4008 \csname selectfont \endcsname{\bbl@xenohyph}%

```

```

4009         \expandafter\selectlanguage\expandafter{\language}%
4010         \expandafter\bbbl@tglobal\csname selectfont \endcsname}%
4011     \fi}}%
4012 \fi
4013 \bbbl@csarg\bbbl@tglobal{lsys@#1}}
4014 \def\bbbl@xeno-hyph@d{%
4015     \bbbl@ifset{\bbbl@prehc@\language}%
4016     {\ifnum\hyphenchar\font=\defaultshyphenchar
4017         \iffontchar\font\bbbl@cl{prehc}\relax
4018         \hyphenchar\font\bbbl@cl{prehc}\relax
4019         \else\iffontchar\font"200B
4020             \hyphenchar\font"200B
4021         \else
4022             \bbbl@warning
4023             {Neither 0 nor ZERO WIDTH SPACE are available\\%
4024              in the current font, and therefore the hyphen\\%
4025              will be printed. Try changing the fontspec's\\%
4026              'HyphenChar' to another value, but be aware\\%
4027              this setting is not safe (see the manual)}%
4028             \hyphenchar\font\defaultshyphenchar
4029         \fi\fi
4030     \fi}%
4031     {\hyphenchar\font\defaultshyphenchar}}
4032 % \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).

```

4033 \def\bbbl@load@info#1{%
4034     \def\BabelBeforeIni##1##2{%
4035         \begingroup
4036         \bbbl@read@ini{##1}0%
4037         \endinput          % babel- .tex may contain onlypreamble's
4038         \endgroup}%        boxed, to avoid extra spaces:
4039     {\bbbl@input@texini{#1}}}

```

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

```

4040 \def\bbbl@setdigits#1#2#3#4#5{%
4041     \bbbl@exp{%
4042         \def\<\language digits>####1{%          ie, \langdigits
4043             \<\bbbl@digits@\language>####1\\\@nil}%
4044             \let\<\bbbl@cntr@digits@\language>\<\language digits>%
4045             \def\<\language counter>####1{%      ie, \langcounter
4046                 \expandafter\<\bbbl@counter@\language>%
4047                 \csname c@####1\endcsname}%
4048             \def\<\bbbl@counter@\language>####1{% ie, \bbbl@counter@lang
4049                 \expandafter\<\bbbl@digits@\language>%
4050                 \number####1\\\@nil}}}%
4051     \def\bbbl@tempa##1##2##3##4##5{%
4052         \bbbl@exp{%      Wow, quite a lot of hashes! :- (
4053             \def\<\bbbl@digits@\language>#####1{%
4054                 \ifx#####1\\\@nil                % ie, \bbbl@digits@lang
4055                 \else
4056                     \ifx0#####1#1%
4057                     \else\ifx1#####1#2%
4058                     \else\ifx2#####1#3%

```

[illegible]

```

4071 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
4072   \ifx\\#1%           % \\ before, in case #1 is multiletter
4073     \bbl@exp{%
4074       \def\\ \bbl@tempa####1{%
4075         \<ifcase>####1\space\the\toks@\<else>\\ \ctrerr\<fi>}}%
4076   \else
4077     \toks@\expandafter{\the\toks@\or #1}%
4078     \expandafter\bbl@buildifcase
4079   \fi}

```

```

4080 \newcommand\locallenumerical[2]{\bbl@cs{cntr@#1@\languageame}{#2}}
4081 \def\bbl@localecntr#1#2{\locallenumerical{#2}{#1}}
4082 \newcommand\localecounter[2]{%
4083   \expandafter\bbl@localecntr
4084   \expandafter{\number\csname c@#2\endcsname}{#1}}
4085 \def\bbl@alphnumerical#1#2{%
4086   \expandafter\bbl@alphnumerical@i\number#2 76543210\@@{#1}}
4087 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
4088   \ifcase\car#8\enil\or    % Currenty <10000, but prepared for bigger
4089     \bbl@alphnumerical@ii{#9}000000#1\or
4090     \bbl@alphnumerical@ii{#9}00000#1#2\or
4091     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
4092     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
4093     \bbl@alphnum@invalid{>9999}%
4094   \fi}
4095 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
4096   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languageame}%
4097   {\bbl@cs{cntr@#1.4@\languageame}{#5}
4098     \bbl@cs{cntr@#1.3@\languageame}{#6}
4099     \bbl@cs{cntr@#1.2@\languageame}{#7}
4100     \bbl@cs{cntr@#1.1@\languageame}{#8}
4101     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4102       \bbl@ifunset{bbl@cntr@#1.S.321@\languageame}{}%
4103       {\bbl@cs{cntr@#1.S.321@\languageame}{}%
4104       \fi}%
4105   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languageame}}}}
4106 \def\bbl@alphnum@invalid#1{%
4107   \bbl@error{Alphabetic numeral too large (#1)}%
4108   {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.

```

4109 \newcommand\localeinfo[1]{%
4110   \bbl@ifunset{\bbl@csname bbl@info@#1\endcsname @\language}%
4111   {\bbl@error{I've found no info for the current locale.\%
4112             The corresponding ini file has not been loaded\%
4113             Perhaps it doesn't exist}%
4114   {See the manual for details.}}%
4115   {\bbl@cs{\csname bbl@info@#1\endcsname @\language}}}%
4116 \@namedef{\bbl@info@name.locale}{lcname}
4117 \@namedef{\bbl@info@tag.ini}{lini}
4118 \@namedef{\bbl@info@name.english}{elname}
4119 \@namedef{\bbl@info@name.opentype}{lname}
4120 \@namedef{\bbl@info@tag.bcp47}{tbc}
4121 \@namedef{\bbl@info@language.tag.bcp47}{lbc}
4122 \@namedef{\bbl@info@tag.opentype}{lotf}
4123 \@namedef{\bbl@info@script.name}{esname}
4124 \@namedef{\bbl@info@script.name.opentype}{sname}
4125 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
4126 \@namedef{\bbl@info@script.tag.opentype}{sotf}
4127 \let\bbl@ensureinfo\@gobble
4128 \newcommand\BabelEnsureInfo{%
4129   \ifx\InputIfFileExists\undefined\else
4130     \def\bbl@ensureinfo##1{%
4131       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}}%
4132   \fi
4133   \bbl@foreach\bbl@loaded{%
4134     \def\language{##1}%
4135     \bbl@ensureinfo{##1}}}%

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

4136 \newcommand\getlocaleproperty{%
4137   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4138 \def\bbl@getproperty@s#1#2#3{%
4139   \let#1\relax
4140   \def\bbl@elt##1##2##3{%
4141     \bbl@ifsamestring{##1/##2}{##3}%
4142     {\providecommand#1{##3}%
4143     \def\bbl@elt####1####2####3{}}}%
4144   {}}%
4145   \bbl@cs{inidata@#2}}%
4146 \def\bbl@getproperty@x#1#2#3{%
4147   \bbl@getproperty@s{#1}{#2}{#3}%
4148   \ifx#1\relax
4149     \bbl@error
4150     {Unknown key for locale '#2':\%
4151     #3\%
4152     \string#1 will be set to \relax}%
4153     {Perhaps you misspelled it.}%
4154   \fi}
4155 \let\bbl@ini@loaded\@empty
4156 \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.

```

4157 \newcommand\babeladjust[1]{% TODO. Error handling.
4158   \bbl@forkv{#1}{%
4159     \bbl@ifunset{bbl@ADJ@##1@##2}%
4160     {\bbl@cs{ADJ@##1}{##2}}%
4161     {\bbl@cs{ADJ@##1@##2}}}
4162 %
4163 \def\bbl@adjust@lua#1#2{%
4164   \ifvmode
4165     \ifnum\currentgrouplevel=\z@
4166       \directlua{ Babel.#2 }%
4167       \expandafter\expandafter\expandafter\@gobble
4168     \fi
4169   \fi
4170   {\bbl@error % The error is gobbled if everything went ok.
4171     {Currently, #1 related features can be adjusted only\\%
4172       in the main vertical list.}%
4173     {Maybe things change in the future, but this is what it is.}}}
4174 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4175   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4176 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4177   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4178 \@namedef{bbl@ADJ@bidi.text@on}{%
4179   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4180 \@namedef{bbl@ADJ@bidi.text@off}{%
4181   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4182 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4183   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4184 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4185   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4186 %
4187 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4188   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4189 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4190   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4191 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4192   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4193 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4194   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4195 \@namedef{bbl@ADJ@justify.arabic@on}{%
4196   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4197 \@namedef{bbl@ADJ@justify.arabic@off}{%
4198   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4199 %
4200 \def\bbl@adjust@layout#1{%
4201   \ifvmode
4202     #1%
4203     \expandafter\@gobble
4204   \fi
4205   {\bbl@error % The error is gobbled if everything went ok.
4206     {Currently, layout related features can be adjusted only\\%
4207       in vertical mode.}%
4208     {Maybe things change in the future, but this is what it is.}}}
4209 \@namedef{bbl@ADJ@layout.tabular@on}{%
4210   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
4211 \@namedef{bbl@ADJ@layout.tabular@off}{%
4212   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4213 \@namedef{bbl@ADJ@layout.lists@on}{%
4214   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4215 \@namedef{bbl@ADJ@layout.lists@off}{%

```

```

4216 \bbl@adjust@layout{\let\list\bbl@OL@list}}
4217 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4218 \bbl@activateposthyphen}
4219 %
4220 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4221 \bbl@bcpallowedtrue}
4222 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4223 \bbl@bcpallowedfalse}
4224 \@namedef{bbl@ADJ@autoload.bcp47.prefix}{#1}%
4225 \def\bbl@bcp@prefix{#1}}
4226 \def\bbl@bcp@prefix{bcp47-}
4227 \@namedef{bbl@ADJ@autoload.options}{#1}%
4228 \def\bbl@autoload@options{#1}}
4229 \let\bbl@autoload@bcptoptions\@empty
4230 \@namedef{bbl@ADJ@autoload.bcp47.options}{#1}%
4231 \def\bbl@autoload@bcptoptions{#1}}
4232 \newif\ifbbl@bcptoname
4233 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4234 \bbl@bcptonametrue}
4235 \BabelEnsureInfo}
4236 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4237 \bbl@bcptonamefalse}
4238 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4239 \directlua{ Babel.ignore_pre_char = function(node)
4240     return (node.lang == \the\csname l@nohyphenation\endcsname)
4241     end }}
4242 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4243 \directlua{ Babel.ignore_pre_char = function(node)
4244     return false
4245     end }}
4246 % TODO: use babel name, override
4247 %
4248 % As the final task, load the code for lua.
4249 %
4250 \ifx\directlua\@undefined\else
4251 \ifx\bbl@luapatterns\@undefined
4252 \input luababel.def
4253 \fi
4254 \fi
4255 </core>

A proxy file for switch.def
4256 <*kernel>
4257 \let\bbl@onlyswitch\@empty
4258 \input babel.def
4259 \let\bbl@onlyswitch\@undefined
4260 </kernel>
4261 <*patterns>

```

11 Loading hyphenation patterns

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

To make sure that \LaTeX 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns. Then everything is restored to the old situation and the format is dumped.

```

4262 <<Make sure ProvidesFile is defined>>
4263 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4264 \xdef\bbl@format{\jobname}
4265 \def\bbl@version{\<<version>>}
4266 \def\bbl@date{\<<date>>}
4267 \ifx\AtBeginDocument\@undefined
4268   \def\@empty{}
4269   \let\orig@dump\dump
4270   \def\dump{%
4271     \ifx\@ztryfc\@undefined
4272     \else
4273       \toks0=\expandafter{\@preamblecmds}%
4274       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4275       \def\@begindocumenthook{}%
4276     \fi
4277     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4278 \fi
4279 <<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.

```

4280 \def\process@line#1#2 #3 #4 {%
4281   \ifx=#1%
4282     \process@synonym{#2}%
4283   \else
4284     \process@language{#1#2}{#3}{#4}%
4285   \fi
4286   \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.

```

4287 \toks@{}
4288 \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 `hyphenmins` parameters for the synonym.

```

4289 \def\process@synonym#1{%
4290   \ifnum\last@language=\m@ne
4291     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4292   \else
4293     \expandafter\chardef\csname l@#1\endcsname\last@language
4294     \wlog{\string\l@#1=\string\language\the\last@language}%
4295     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4296       \csname\language\hyphenmins\endcsname
4297     \let\bbl@elt\relax
4298     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4299   \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`. T_EX 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` or `\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.

```

4300 \def\process@language#1#2#3{%
4301   \expandafter\addlanguage\csname l@#1\endcsname
4302   \expandafter\language\csname l@#1\endcsname
4303   \edef\language#1{%
4304     \bbl@hook@everylanguage{#1}%
4305     % > luatex
4306     \bbl@get@enc#1::\@@@
4307     \begingroup
4308       \lefthyphenmin\m@ne
4309       \bbl@hook@loadpatterns{#2}%
4310       % > luatex
4311       \ifnum\lefthyphenmin=\m@ne
4312       \else
4313         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4314           \the\lefthyphenmin\the\righthyphenmin}%
4315       \fi
4316     \endgroup
4317     \def\bbl@tempa{#3}%
4318     \ifx\bbl@tempa\@empty\else
4319       \bbl@hook@loadexceptions{#3}%
4320       % > luatex
4321     \fi
4322     \let\bbl@elt\relax
4323     \edef\bbl@languages{%
4324       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4325     \ifnum\the\language=\z@
4326       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4327         \set@hyphenmins\tw@\thr@@\relax
4328       \else
4329         \expandafter\expandafter\expandafter\set@hyphenmins
4330         \csname #1hyphenmins\endcsname
4331       \fi
4332       \the\toks@
4333       \toks@{}%
4334     \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.

```
4335 \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.

```
4336 \def\bbl@hook@everylanguage#1{}
4337 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4338 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4339 \def\bbl@hook@loadkernel#1{%
4340   \def\addlanguage{\csname newlanguage\endcsname}%
4341   \def\adddialect##1##2{%
4342     \global\chardef##1##2\relax
4343     \wlog{\string##1 = a dialect from \string\language##2}}%
4344   \def\iflanguage#1{%
4345     \expandafter\ifx\csname l@##1\endcsname\relax
4346       \@nolanerr{##1}%
4347     \else
4348       \ifnum\csname l@##1\endcsname=\language
4349         \expandafter\expandafter\expandafter\@firstoftwo
4350       \else
4351         \expandafter\expandafter\expandafter\@secondoftwo
4352       \fi
4353     \fi}%
4354   \def\providehyphenmins##1##2{%
4355     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4356       \@namedef{##1hyphenmins}{##2}%
4357     \fi}%
4358   \def\set@hyphenmins##1##2{%
4359     \lefthyphenmin##1\relax
4360     \righthyphenmin##2\relax}%
4361   \def\selectlanguage{%
4362     \errhelp{Selecting a language requires a package supporting it}%
4363     \errmessage{Not loaded}}%
4364   \let\foreignlanguage\selectlanguage
4365   \let\otherlanguage\selectlanguage
4366   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4367   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4368     \def\setlocale{%
4369       \errhelp{Find an armchair, sit down and wait}%
4370       \errmessage{Not yet available}}%
4371     \let\uselocale\setlocale
4372     \let\locale\setlocale
4373     \let\selectlocale\setlocale
4374     \let\localename\setlocale
4375     \let\textlocale\setlocale
4376     \let\textlanguage\setlocale
4377     \let\languagetext\setlocale}
4378   \begingroup
4379   \def\AddBabelHook#1#2{%
4380     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4381       \def\next{\toks1}%
4382     \else
4383       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4384     \fi
4385     \next}
4386   \ifx\directlua\@undefined
```

```

4387 \ifx\XeTeXinputencoding\@undefined\else
4388 \input xebabel.def
4389 \fi
4390 \else
4391 \input luababel.def
4392 \fi
4393 \openin1 = babel-\bbl@format.cfg
4394 \ifeof1
4395 \else
4396 \input babel-\bbl@format.cfg\relax
4397 \fi
4398 \closein1
4399 \endgroup
4400 \bbl@hook@loadkernel{switch.def}

```

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

```

4401 \openin1 = language.dat

```

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

```

4402 \def\language{english}%
4403 \ifeof1
4404 \message{I couldn't find the file language.dat,\space
4405         I will try the file hyphen.tex}
4406 \input hyphen.tex\relax
4407 \chardef\l@english\z@
4408 \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 .

```

4409 \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.

```

4410 \loop
4411 \endlinechar\m@ne
4412 \read1 to \bbl@line
4413 \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.

```

4414 \if T\ifeof1F\fi T\relax
4415 \ifx\bbl@line\@empty\else
4416 \edef\bbl@line{\bbl@line\space\space\space}%
4417 \expandafter\process@line\bbl@line\relax
4418 \fi
4419 \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.

```

4420 \begingroup
4421 \def\bbl@elt#1#2#3#4{%
4422 \global\language=#2\relax
4423 \gdef\language{#1}%
4424 \def\bbl@elt##1##2##3##4{}}%

```

```

4425 \bbl@languages
4426 \endgroup
4427 \fi
4428 \closein1

```

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

```

4429 \if/\the\toks@/\else
4430 \errhelp{language.dat loads no language, only synonyms}
4431 \errmessage{Orphan language synonym}
4432 \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.

```

4433 \let\bbl@line\@undefined
4434 \let\process@line\@undefined
4435 \let\process@synonym\@undefined
4436 \let\process@language\@undefined
4437 \let\bbl@get@enc\@undefined
4438 \let\bbl@hyph@enc\@undefined
4439 \let\bbl@tempa\@undefined
4440 \let\bbl@hook@loadkernel\@undefined
4441 \let\bbl@hook@everylanguage\@undefined
4442 \let\bbl@hook@loadpatterns\@undefined
4443 \let\bbl@hook@loadexceptions\@undefined
4444 \</patterns>

```

Here the code for `iniTeX` ends.

12 Font handling with fontspec

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

```

4445 <<(*More package options)>> ≡
4446 \chardef\bbl@bidimode\z@
4447 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4448 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4449 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4450 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4451 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4452 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4453 <</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.

```

4454 <<(*Font selection)>> ≡
4455 \bbl@trace{Font handling with fontspec}
4456 \ifx\ExplSyntaxOn\@undefined\else
4457 \ExplSyntaxOn
4458 \catcode`\ =10
4459 \def\bbl@loadfontspec{%
4460 \usepackage{fontspec}%
4461 \expandafter
4462 \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4463 Font '\l_fontspec_fontname_tl' is using the\%

```

```

4464     default features for language '##1'.\\%
4465     That's usually fine, because many languages\\%
4466     require no specific features, but if the output is\\%
4467     not as expected, consider selecting another font.}
4468 \expandafter
4469 \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4470     Font '\l_fontspec_fontname_tl' is using the\\%
4471     default features for script '##2'.\\%
4472     That's not always wrong, but if the output is\\%
4473     not as expected, consider selecting another font.}}
4474 \ExplSyntaxOff
4475 \fi
4476 \@onlypreamble\babelfont
4477 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4478     \bbl@foreach{#1}{%
4479         \expandafter\ifx\csname date##1\endcsname\relax
4480             \IfFileExists{babel-##1.tex}%
4481                 {\babelprovide{##1}}%
4482                 {}%
4483         \fi}%
4484 \edef\bbl@tempa{#1}%
4485 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4486 \ifx\fontspec\undefined
4487     \bbl@loadfontspec
4488 \fi
4489 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4490 \bbl@bblfont}
4491 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4492     \bbl@ifunset{\bbl@tempb family}%
4493     {\bbl@providefam{\bbl@tempb}}%
4494     {\bbl@exp{%
4495         \\bbl@sreplace\<\bbl@tempb family >%
4496         {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4497 % For the default font, just in case:
4498 \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}}%
4499 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4500 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4501     \bbl@exp{%
4502         \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4503         \\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4504         \<\bbl@tempb default>\<\bbl@tempb family>}}%
4505     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4506         \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:

```

4507 \def\bbl@providefam#1{%
4508     \bbl@exp{%
4509         \\newcommand\<#1default>{}% Just define it
4510         \\bbl@add@list\\bbl@font@fams{#1}%
4511         \\DeclareRobustCommand\<#1family>{%
4512             \\not@math@alphabet\<#1family>\relax
4513             \\fontfamily\<#1default>\\selectfont}%
4514         \\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.

```

4515 \def\bbl@nostdfont#1{%
4516     \bbl@ifunset{\bbl@WFF@f@family}%
4517     {\bbl@csarg\gdef{WFF@f@family}}}% Flag, to avoid dupl warns

```

```

4518 \bbl@infowarn{The current font is not a babel standard family:\%
4519 #1%
4520 \fontname\font\%
4521 There is nothing intrinsically wrong with this warning, and\%
4522 you can ignore it altogether if you do not need these\%
4523 families. But if they are used in the document, you should be\%
4524 aware 'babel' will no set Script and Language for them, so\%
4525 you may consider defining a new family with \string\babelfont.\%
4526 See the manual for further details about \string\babelfont.\%
4527 Reported}}
4528 {}}%
4529 \gdef\bbl@switchfont{%
4530 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}}%
4531 \bbl@exp{% eg Arabic -> arabic
4532 \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}}%
4533 \bbl@foreach\bbl@font@fams{%
4534 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4535 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4536 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4537 {}% 123=F - nothing!
4538 {\bbl@exp{% 3=T - from generic
4539 \global\let<\bbl@##1dflt@\language>%
4540 \<\bbl@##1dflt@>}}}%
4541 {\bbl@exp{% 2=T - from script
4542 \global\let<\bbl@##1dflt@\language>%
4543 \<\bbl@##1dflt@*\bbl@tempa>}}}%
4544 {}}% 1=T - language, already defined
4545 \def\bbl@tempa{\bbl@nostdfont}}%
4546 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4547 \bbl@ifunset{\bbl@##1dflt@\language}%
4548 {\bbl@cs{famrst@##1}%
4549 \global\bbl@csarg\let{famrst@##1}\relax}%
4550 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4551 \\\bbl@add\\originalTeX{%
4552 \\\bbl@font@rst{\bbl@cl{##1dflt}}}%
4553 \<##1default>\<##1family>{##1}}}%
4554 \\\bbl@font@set<\bbl@##1dflt@\language>% the main part!
4555 \<##1default>\<##1family>}}}%
4556 \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.

```

4557 \ifx\f@family\undefined\else % if latex
4558 \ifcase\bbl@engine % if pdftex
4559 \let\bbl@ckeckstdfonts\relax
4560 \else
4561 \def\bbl@ckeckstdfonts{%
4562 \begingroup
4563 \global\let\bbl@ckeckstdfonts\relax
4564 \let\bbl@tempa\@empty
4565 \bbl@foreach\bbl@font@fams{%
4566 \bbl@ifunset{\bbl@##1dflt@}%
4567 {\nameuse{##1family}%
4568 \bbl@csarg\gdef{WFF@f@family}}}% Flag
4569 \bbl@exp{\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4570 \space\space\fontname\font\\}%
4571 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4572 \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4573 {}}%

```

```

4574 \ifx\bbl@tempa\@empty\else
4575 \bbl@infowarn{The following font families will use the default\\%
4576 settings for all or some languages:\\%
4577 \bbl@tempa
4578 There is nothing intrinsically wrong with it, but\\%
4579 'babel' will no set Script and Language, which could\\%
4580 be relevant in some languages. If your document uses\\%
4581 these families, consider redefining them with \string\babelfont.\\%
4582 Reported}%
4583 \fi
4584 \endgroup}
4585 \fi
4586 \fi

```

Now the macros defining the font with fontspec.

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

```

4587 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4588 \bbl@xin@{<>}{#1}%
4589 \ifin@
4590 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4591 \fi
4592 \bbl@exp{% 'Unprotected' macros return prev values
4593 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4594 \\bbl@ifsamestring{#2}{\f@family}%
4595 {\#3%
4596 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4597 \let\\bbl@tempa\relax}%
4598 {}}}
4599 % TODO - next should be global?, but even local does its job. I'm
4600 % still not sure -- must investigate:
4601 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4602 \let\bbl@tempe\bbl@mapselect
4603 \let\bbl@mapselect\relax
4604 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4605 \let#4\@empty % Make sure \renewfontfamily is valid
4606 \bbl@exp{%
4607 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4608 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4609 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4610 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4611 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4612 \\renewfontfamily\\#4%
4613 [\bbl@cs{lsys@\languagename},#2]{#3}% ie \bbl@exp{.}{#3}
4614 \begingroup
4615 #4%
4616 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4617 \endgroup
4618 \let#4\bbl@temp@fam
4619 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4620 \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.

```

4621 \def\bbl@font@rst#1#2#3#4{%
4622 \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.

```

4623 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for `\babelFSfeatures`. The reason is explained in the user guide, but essentially – that was not the way to go :-).

```

4624 \newcommand\babelFSstore[2][{}%
4625   \bbl@ifblank{#1}%
4626   {\bbl@csarg\def\sname@#2}{Latin}}%
4627   {\bbl@csarg\def\sname@#2}{#1}}%
4628   \bbl@provide@dirs{#2}%
4629   \bbl@csarg\ifnum{wdir@#2}>\z@
4630     \let\bbl@beforeforeign\leavevmode
4631     \EnableBabelHook{babel-bidi}%
4632   \fi
4633   \bbl@foreach{#2}{%
4634     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4635     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4636     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4637 \def\bbl@FSstore#1#2#3#4{%
4638   \bbl@csarg\edef{#2default#1}{#3}%
4639   \expandafter\addto\csname extras#1\endcsname{%
4640     \let#4#3%
4641     \ifx#3\f@family
4642       \edef#3{\csname bbl@#2default#1\endcsname}%
4643       \fontfamily{#3}\selectfont
4644     \else
4645       \edef#3{\csname bbl@#2default#1\endcsname}%
4646       \fi}%
4647   \expandafter\addto\csname noextras#1\endcsname{%
4648     \ifx#3\f@family
4649       \fontfamily{#4}\selectfont
4650     \fi
4651     \let#3#4}}
4652 \let\bbl@langfeatures\empty
4653 \def\babelFSfeatures{% make sure \fontspec is redefined once
4654   \let\bbl@ori@fontspec\fontspec
4655   \renewcommand\fontspec[1][{}%
4656     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4657   \let\babelFSfeatures\bbl@FSfeatures
4658   \babelFSfeatures}
4659 \def\bbl@FSfeatures#1#2{%
4660   \expandafter\addto\csname extras#1\endcsname{%
4661     \babel@save\bbl@langfeatures
4662     \edef\bbl@langfeatures{#2,}}
4663   <</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.

```

4664 <<{*Footnote changes}>> ≡
4665 \bbl@trace{Bidi footnotes}
4666 \ifnum\bbl@bidimode>\z@
4667   \def\bbl@footnote#1#2#3{%
4668     \@ifnextchar[%
4669       {\bbl@footnote@o{#1}{#2}{#3}}%
4670       {\bbl@footnote@x{#1}{#2}{#3}}}

```

```

4671 \long\def\bbl@footnote@x#1#2#3#4{%
4672   \bgroup
4673   \select@language@x{\bbl@main@language}%
4674   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4675   \egroup}
4676 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4677   \bgroup
4678   \select@language@x{\bbl@main@language}%
4679   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4680   \egroup}
4681 \def\bbl@footnotetext#1#2#3{%
4682   \@ifnextchar[%
4683     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4684     {\bbl@footnotetext@x{#1}{#2}{#3}}%
4685   \long\def\bbl@footnotetext@x#1#2#3#4{%
4686     \bgroup
4687     \select@language@x{\bbl@main@language}%
4688     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4689     \egroup}
4690   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4691     \bgroup
4692     \select@language@x{\bbl@main@language}%
4693     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4694     \egroup}
4695   \def\BabelFootnote#1#2#3#4{%
4696     \ifx\bbl@fn@footnote\@undefined
4697       \let\bbl@fn@footnote\footnote
4698     \fi
4699     \ifx\bbl@fn@footnotetext\@undefined
4700       \let\bbl@fn@footnotetext\footnotetext
4701     \fi
4702     \bbl@ifblank{#2}%
4703     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4704      \@namedef{\bbl@stripslash#1text}%
4705       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4706     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4707      \@namedef{\bbl@stripslash#1text}%
4708       {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4709   \fi
4710 <</Footnote changes>>

```

Now, the code.

```

4711 <*\xetex>
4712 \def\BabelStringsDefault{unicode}
4713 \let\xebbl@stop\relax
4714 \AddBabelHook{xetex}{encodedcommands}{%
4715   \def\bbl@tempa{#1}%
4716   \ifx\bbl@tempa\@empty
4717     \XeTeXinputencoding"bytes"%
4718   \else
4719     \XeTeXinputencoding"#1"%
4720   \fi
4721   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4722 \AddBabelHook{xetex}{stopcommands}{%
4723   \xebbl@stop
4724   \let\xebbl@stop\relax}
4725 \def\bbl@intraspace#1 #2 #3\@@{%
4726   \bbl@csarg\gdef{\xeisp{\languagename}%
4727     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}

```

```

4728 \def\bbl@intrapenalty#1\@@{%
4729   \bbl@csarg\gdef{xeipn@\language}%
4730   {\XeTeXlinebreakpenalty #1\relax}}
4731 \def\bbl@provide@intraspace{%
4732   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4733   \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4734   \ifin@
4735     \bbl@ifunset{\bbl@intsp@\language}{}%
4736     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4737       \ifx\bbl@KVP@intraspace\@nil
4738         \bbl@exp{%
4739           \bbl@intraspace\bbl@cl{intsp}\@@}%
4740       \fi
4741       \ifx\bbl@KVP@intrapenalty\@nil
4742         \bbl@intrapenalty0\@@
4743       \fi
4744     \fi
4745     \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4746       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4747     \fi
4748     \ifx\bbl@KVP@intrapenalty\@nil\else
4749       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4750     \fi
4751     \bbl@exp{%
4752       \bbl@add\<extras\language>%
4753       \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4754       \<bbl@xeisp@\language>%
4755       \<bbl@xeipn@\language>%
4756       \bbl@toglobal\<extras\language>%
4757       \bbl@add\<noextras\language>%
4758       \XeTeXlinebreaklocale "en"%
4759       \bbl@toglobal\<noextras\language>%
4760     \ifx\bbl@ispace\@undefined
4761       \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4762     \ifx\AtBeginDocument\@notprerr
4763       \expandafter\@secondoftwo % to execute right now
4764     \fi
4765     \AtBeginDocument{%
4766       \expandafter\bbl@add
4767       \csname selectfont \endcsname{\bbl@ispace}%
4768       \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4769     \fi}%
4770   \fi}
4771 \ifx\DisableBabelHook\@undefined\endinput\fi
4772 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4773 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4774 \DisableBabelHook{babel-fontspec}
4775 <<Font selection>>
4776 \input txtbabel.def
4777 </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 `pdftex` and `xetex`.

```

4778 (*texxet)
4779 \providecommand\bbl@provide@intraspace{}
4780 \bbl@trace{Redefinitions for bidi layout}
4781 \def\bbl@sspre@caption{%
4782   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir\bbl@main@language}}}}
4783 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4784 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4785 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4786 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4787   \def\hangfrom#1{%
4788     \setbox\@tempboxa\hbox{#1}%
4789     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4790     \noindent\box\@tempboxa}
4791 \def\raggedright{%
4792   \let\@centercr
4793   \bbl@startskip\z@skip
4794   \@rightskip\@flushglue
4795   \bbl@endskip\@rightskip
4796   \parindent\z@
4797   \parfillskip\bbl@startskip}
4798 \def\raggedleft{%
4799   \let\@centercr
4800   \bbl@startskip\@flushglue
4801   \bbl@endskip\z@skip
4802   \parindent\z@
4803   \parfillskip\bbl@endskip}
4804 \fi
4805 \IfBabelLayout{lists}
4806   {\bbl@sreplace\list
4807     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4808     \def\bbl@listleftmargin{%
4809       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4810     \ifcase\bbl@engine
4811       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4812       \def\p@enumiii{\p@enumii}\theenumii{}
4813     \fi
4814     \bbl@sreplace\@verbatim
4815       {\leftskip\@totalleftmargin}%
4816       {\bbl@startskip\textwidth
4817         \advance\bbl@startskip-\linewidth}%
4818     \bbl@sreplace\@verbatim
4819       {\rightskip\z@skip}%
4820       {\bbl@endskip\z@skip}}%
4821   {}
4822 \IfBabelLayout{contents}
4823   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4824     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4825   {}
4826 \IfBabelLayout{columns}
4827   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4828     \def\bbl@outputbox#1{%
4829       \hb@xt@\textwidth{%
4830         \hskip\columnwidth
4831         \hfil
4832         {\normalcolor\vrule \@width\columnseprule}%
4833         \hfil
4834         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%

```

```

4835      \hskip-\textwidth
4836      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4837      \hskip\columnsep
4838      \hskip\columnwidth}}}%
4839  {}
4840 <<Footnote changes>>
4841 \IfBabelLayout{footnotes}%
4842  {\BabelFootnote\footnote\language\language{}{}}%
4843  \BabelFootnote\localfootnote\language\language{}{}}%
4844  \BabelFootnote\mainfootnote{}{}}{}
4845  {}

```

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.

```

4846 \IfBabelLayout{counters}%
4847  {\let\bbl@latinarabic=\@arabic
4848   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4849  \let\bbl@asciroman=\@roman
4850  \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4851  \let\bbl@asciiRoman=\@Roman
4852  \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4853 </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).

```

4854 <*\luatex>

```

```

4855 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4856 \bbl@trace{Read language.dat}
4857 \ifx\bbl@readstream\@undefined
4858   \csname newread\endcsname\bbl@readstream
4859 \fi
4860 \beginingroup
4861   \toks@{}
4862   \count@ \z@ % 0=start, 1=0th, 2=normal
4863   \def\bbl@process@line#1#2 #3 #4 {%
4864     \ifx=#1%
4865       \bbl@process@synonym{#2}%
4866     \else
4867       \bbl@process@language{#1#2}{#3}{#4}%
4868     \fi
4869     \ignorespaces}
4870   \def\bbl@manylang{%
4871     \ifnum\bbl@last>\@ne
4872       \bbl@info{Non-standard hyphenation setup}%
4873     \fi
4874     \let\bbl@manylang\relax}
4875   \def\bbl@process@language#1#2#3{%
4876     \ifcase\count@
4877       \@ifundefined{zth#1}{\count@ \tw@}{\count@ \@ne}%
4878     \or
4879       \count@ \tw@
4880     \fi
4881     \ifnum\count@=\tw@
4882       \expandafter\addlanguage\csname l@#1\endcsname
4883       \language\allocationnumber
4884       \chardef\bbl@last\allocationnumber
4885       \bbl@manylang
4886       \let\bbl@elt\relax
4887       \xdef\bbl@languages{%
4888         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4889     \fi
4890     \the\toks@
4891     \toks@{}}
4892   \def\bbl@process@synonym@aux#1#2{%
4893     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4894     \let\bbl@elt\relax
4895     \xdef\bbl@languages{%
4896       \bbl@languages\bbl@elt{#1}{#2}{}}}%
4897   \def\bbl@process@synonym#1{%
4898     \ifcase\count@
4899       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4900     \or
4901       \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4902     \else
4903       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4904     \fi}
4905   \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4906     \chardef\l@english\z@
4907     \chardef\l@USenglish\z@
4908     \chardef\bbl@last\z@
4909     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4910     \gdef\bbl@languages{%
4911       \bbl@elt{english}{0}{hyphen.tex}}%
4912     \bbl@elt{USenglish}{0}{}
4913   \else

```

```

4914 \global\let\bbl@languages@format\bbl@languages
4915 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4916 \ifnum#2>\z@\else
4917 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4918 \fi}%
4919 \xdef\bbl@languages{\bbl@languages}%
4920 \fi
4921 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4922 \bbl@languages
4923 \openin\bbl@readstream=language.dat
4924 \ifeof\bbl@readstream
4925 \bbl@warning{I couldn't find language.dat. No additional\\%
4926 patterns loaded. Reported}%
4927 \else
4928 \loop
4929 \endlinechar\m@ne
4930 \read\bbl@readstream to \bbl@line
4931 \endlinechar\^^M
4932 \if T\ifeof\bbl@readstream F\fi T\relax
4933 \ifx\bbl@line\empty\else
4934 \edef\bbl@line{\bbl@line\space\space\space}%
4935 \expandafter\bbl@process@line\bbl@line\relax
4936 \fi
4937 \repeat
4938 \fi
4939 \endgroup
4940 \bbl@trace{Macros for reading patterns files}
4941 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4942 \ifx\babelcatcodetablenum\undefined
4943 \ifx\newcatcodetable\undefined
4944 \def\babelcatcodetablenum{5211}
4945 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4946 \else
4947 \newcatcodetable\babelcatcodetablenum
4948 \newcatcodetable\bbl@pattcodes
4949 \fi
4950 \else
4951 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4952 \fi
4953 \def\bbl@luapatterns#1#2{%
4954 \bbl@get@enc#1::\@@
4955 \setbox\z@\hbox\bgroup
4956 \begingroup
4957 \savecatcodetable\babelcatcodetablenum\relax
4958 \initcatcodetable\bbl@pattcodes\relax
4959 \catcodetable\bbl@pattcodes\relax
4960 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4961 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4962 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4963 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4964 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4965 \catcode`\'=12 \catcode`\`=12 \catcode`\`=12
4966 \input #1\relax
4967 \catcodetable\babelcatcodetablenum\relax
4968 \endgroup
4969 \def\bbl@tempa{#2}%
4970 \ifx\bbl@tempa\empty\else
4971 \input #2\relax
4972 \fi

```

```

4973 \egroup}%
4974 \def\bbl@patterns@lua#1{%
4975 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4976 \csname l@#1\endcsname
4977 \edef\bbl@tempa{#1}%
4978 \else
4979 \csname l@#1:\f@encoding\endcsname
4980 \edef\bbl@tempa{#1:\f@encoding}%
4981 \fi\relax
4982 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4983 \@ifundefined{bbl@hyphendata@the\language}%
4984 {\def\bbl@elt##1##2##3##4{%
4985 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4986 \def\bbl@tempb{##3}%
4987 \ifx\bbl@tempb@empty\else % if not a synonymous
4988 \def\bbl@tempc{##3}{##4}%
4989 \fi
4990 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4991 \fi}%
4992 \bbl@languages
4993 \@ifundefined{bbl@hyphendata@the\language}%
4994 {\bbl@info{No hyphenation patterns were set for\%
4995 language '\bbl@tempa'. Reported}}%
4996 {\expandafter\expandafter\expandafter\bbl@luapatterns
4997 \csname bbl@hyphendata@the\language\endcsname}}}%
4998 \endinput\fi
4999 % Here ends \ifx\AddBabelHook\@undefined
5000 % A few lines are only read by hyphen.cfg
5001 \ifx\DisableBabelHook\@undefined
5002 \AddBabelHook{luatex}{everylanguage}{%
5003 \def\process@language##1##2##3{%
5004 \def\process@line####1####2 ####3 ####4 {}}%
5005 \AddBabelHook{luatex}{loadpatterns}{%
5006 \input #1\relax
5007 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5008 {{#1}}}%
5009 \AddBabelHook{luatex}{loadexceptions}{%
5010 \input #1\relax
5011 \def\bbl@tempb##1##2{{##1}{##2}}%
5012 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5013 {\expandafter\expandafter\expandafter\bbl@tempb
5014 \csname bbl@hyphendata@the\language\endcsname}}
5015 \endinput\fi
5016 % Here stops reading code for hyphen.cfg
5017 % The following is read the 2nd time it's loaded
5018 \begingroup % TODO - to a lua file
5019 \catcode`\%=12
5020 \catcode`\'=12
5021 \catcode`\%=12
5022 \catcode`\:=12
5023 \directlua{
5024 Babel = Babel or {}
5025 function Babel.bytes(line)
5026 return line:gsub(".",
5027 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5028 end
5029 function Babel.begin_process_input()
5030 if luatexbase and luatexbase.add_to_callback then
5031 luatexbase.add_to_callback('process_input_buffer',

```

```

5032                                     Babel.bytes,'Babel.bytes')
5033     else
5034         Babel.callback = callback.find('process_input_buffer')
5035         callback.register('process_input_buffer',Babel.bytes)
5036     end
5037 end
5038 function Babel.end_process_input ()
5039     if luatexbase and luatexbase.remove_from_callback then
5040         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5041     else
5042         callback.register('process_input_buffer',Babel.callback)
5043     end
5044 end
5045 function Babel.addpatterns(pp, lg)
5046     local lg = lang.new(lg)
5047     local pats = lang.patterns(lg) or ''
5048     lang.clear_patterns(lg)
5049     for p in pp:gmatch('[^%s]+') do
5050         ss = ''
5051         for i in string.utfcharacters(p:gsub('%d', '')) do
5052             ss = ss .. '%d?' .. i
5053         end
5054         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5055         ss = ss:gsub('%..%%d%?$', '%%.')
5056         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5057         if n == 0 then
5058             tex.sprint(
5059                 [[\string\csname\space bbl@info\endcsname{New pattern: }
5060                 .. p .. [{}]])
5061             pats = pats .. ' ' .. p
5062         else
5063             tex.sprint(
5064                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }
5065                 .. p .. [{}]])
5066         end
5067     end
5068     lang.patterns(lg, pats)
5069 end
5070 }
5071 \endgroup
5072 \ifx\newattribute\@undefined\else
5073     \newattribute\bbl@attr@locale
5074     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5075     \AddBabelHook{luatex}{beforeextras}{%
5076         \setattribute\bbl@attr@locale\localeid}
5077 \fi
5078 \def\BabelStringsDefault{unicode}
5079 \let\luabbl@stop\relax
5080 \AddBabelHook{luatex}{encodedcommands}{%
5081     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5082     \ifx\bbl@tempa\bbl@tempb\else
5083         \directlua{Babel.begin_process_input()}%
5084         \def\luabbl@stop{%
5085             \directlua{Babel.end_process_input()}}%
5086     \fi}%
5087 \AddBabelHook{luatex}{stopcommands}{%
5088     \luabbl@stop
5089     \let\luabbl@stop\relax}
5090 \AddBabelHook{luatex}{patterns}{%

```

```

5091 \@ifundefined{bbl@hyphendata@the\language}%
5092 {\def\bbl@elt##1##2##3##4{%
5093   \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5094   \def\bbl@tempb{##3}%
5095   \ifx\bbl@tempb\@empty\else % if not a synonymous
5096     \def\bbl@tempc{##3}{##4}}%
5097   \fi
5098   \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5099   \fi}%
5100 \bbl@languages
5101 \@ifundefined{bbl@hyphendata@the\language}%
5102 {\bbl@info{No hyphenation patterns were set for\%
5103   language '#2'. Reported}}%
5104 {\expandafter\expandafter\expandafter\bbl@luapatterns
5105   \csname bbl@hyphendata@the\language\endcsname}}}%
5106 \@ifundefined{bbl@patterns@}{}%
5107 \begingroup
5108   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5109   \ifin\else
5110     \ifx\bbl@patterns@\@empty\else
5111       \directlua{ Babel.addpatterns(
5112         [[\bbl@patterns@]], \number\language) }%
5113       \fi
5114       \@ifundefined{bbl@patterns@#1}%
5115         \@empty
5116         {\directlua{ Babel.addpatterns(
5117           [[\space\csname bbl@patterns@#1\endcsname]],
5118           \number\language) }}%
5119       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5120     \fi
5121   \endgroup}%
5122 \bbl@exp{%
5123   \bbl@ifunset{bbl@prehc@\languagename}}}%
5124   {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5125   {\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.

```

5126 \@onlypreamble\babelpatterns
5127 \AtEndOfPackage{%
5128   \newcommand\babelpatterns[2][\@empty]{%
5129     \ifx\bbl@patterns@\relax
5130       \let\bbl@patterns@\@empty
5131     \fi
5132     \ifx\bbl@pttnlist@\@empty\else
5133       \bbl@warning{%
5134         You must not intermingle \string\selectlanguage\space and\%
5135         \string\babelpatterns\space or some patterns will not\%
5136         be taken into account. Reported}%
5137       \fi
5138       \ifx\@empty#1%
5139         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5140       \else
5141         \edef\bbl@tempb{\zap@space#1 \@empty}%
5142         \bbl@for\bbl@tempa\bbl@tempb{%
5143           \bbl@fixname\bbl@tempa
5144           \bbl@iflanguage\bbl@tempa{%
5145             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%

```

```

5146         \@ifundefined{bbl@patterns@bbl@tempa}%
5147         \@empty
5148         {\csname bbl@patterns@bbl@tempa\endcsname\space}%
5149         #2}}}%
5150     \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.

```

5151% TODO - to a lua file
5152\directlua{
5153  Babel = Babel or {}
5154  Babel.linebreaking = Babel.linebreaking or {}
5155  Babel.linebreaking.before = {}
5156  Babel.linebreaking.after = {}
5157  Babel.locale = {} % Free to use, indexed by \localeid
5158  function Babel.linebreaking.add_before(func)
5159    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5160    table.insert(Babel.linebreaking.before, func)
5161  end
5162  function Babel.linebreaking.add_after(func)
5163    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5164    table.insert(Babel.linebreaking.after, func)
5165  end
5166}
5167\def\bbl@intraspace#1 #2 #3\@@{%
5168  \directlua{
5169    Babel = Babel or {}
5170    Babel.intraspaces = Babel.intraspaces or {}
5171    Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5172      {b = #1, p = #2, m = #3}
5173    Babel.locale_props[\the\localeid].intraspace = %
5174      {b = #1, p = #2, m = #3}
5175  }}
5176\def\bbl@intrapenalty#1\@@{%
5177  \directlua{
5178    Babel = Babel or {}
5179    Babel.intrapenalties = Babel.intrapenalties or {}
5180    Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5181    Babel.locale_props[\the\localeid].intrapenalty = #1
5182  }}
5183\begingroup
5184\catcode`\%=12
5185\catcode`\^=14
5186\catcode`\'=12
5187\catcode`\~=12
5188\gdef\bbl@seaintraspace{^
5189  \let\bbl@seaintraspace\relax
5190  \directlua{
5191    Babel = Babel or {}
5192    Babel.sea_enabled = true
5193    Babel.sea_ranges = Babel.sea_ranges or {}
5194    function Babel.set_chranges (script, chrng)
5195      local c = 0
5196      for s, e in string.gmatch(chrng..' ', '(-)%%.(-)%s') do

```

```

5197         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5198         c = c + 1
5199     end
5200 end
5201 function Babel.sea_disc_to_space (head)
5202     local sea_ranges = Babel.sea_ranges
5203     local last_char = nil
5204     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5205     for item in node.traverse(head) do
5206         local i = item.id
5207         if i == node.id'glyph' then
5208             last_char = item
5209         elseif i == 7 and item.subtype == 3 and last_char
5210             and last_char.char > 0x0C99 then
5211             quad = font.getfont(last_char.font).size
5212             for lg, rg in pairs(sea_ranges) do
5213                 if last_char.char > rg[1] and last_char.char < rg[2] then
5214                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5215                     local intraspace = Babel.intraspaces[lg]
5216                     local intrapenalty = Babel.intrapenalties[lg]
5217                     local n
5218                     if intrapenalty ~= 0 then
5219                         n = node.new(14, 0)      ^% penalty
5220                         n.penalty = intrapenalty
5221                         node.insert_before(head, item, n)
5222                     end
5223                     n = node.new(12, 13)      ^% (glue, spaceskip)
5224                     node.setglue(n, intraspace.b * quad,
5225                                 intraspace.p * quad,
5226                                 intraspace.m * quad)
5227                     node.insert_before(head, item, n)
5228                     node.remove(head, item)
5229                 end
5230             end
5231         end
5232     end
5233 end
5234 }^^
5235 \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.

```

5236 \catcode`\%=14
5237 \gdef\bbl@cjkintraspaces{%
5238     \let\bbl@cjkintraspaces\relax
5239     \directlua{
5240         Babel = Babel or {}
5241         require('babel-data-cjk.lua')
5242         Babel.cjk_enabled = true
5243         function Babel.cjk_linebreak(head)
5244             local GLYPH = node.id'glyph'
5245             local last_char = nil

```

```

5246     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5247     local last_class = nil
5248     local last_lang = nil
5249
5250     for item in node.traverse(head) do
5251         if item.id == GLYPH then
5252
5253             local lang = item.lang
5254
5255             local LOCALE = node.get_attribute(item,
5256                 luatexbase.registernumber'bb1@attr@locale')
5257             local props = Babel.locale_props[LOCALE]
5258
5259             local class = Babel.cjk_class[item.char].c
5260
5261             if class == 'cp' then class = 'cl' end %]) as CL
5262             if class == 'id' then class = 'I' end
5263
5264             local br = 0
5265             if class and last_class and Babel.cjk_breaks[last_class][class] then
5266                 br = Babel.cjk_breaks[last_class][class]
5267             end
5268
5269             if br == 1 and props.linebreak == 'c' and
5270                 lang ~= \the\l@nohyphenation\space and
5271                 last_lang ~= \the\l@nohyphenation then
5272                 local intrapenalty = props.intrapenalty
5273                 if intrapenalty ~= 0 then
5274                     local n = node.new(14, 0)      % penalty
5275                     n.penalty = intrapenalty
5276                     node.insert_before(head, item, n)
5277                 end
5278                 local intraspace = props.intraspace
5279                 local n = node.new(12, 13)      % (glue, spaceskip)
5280                 node.setglue(n, intraspace.b * quad,
5281                     intraspace.p * quad,
5282                     intraspace.m * quad)
5283                 node.insert_before(head, item, n)
5284             end
5285
5286             if font.getfont(item.font) then
5287                 quad = font.getfont(item.font).size
5288             end
5289             last_class = class
5290             last_lang = lang
5291         else % if penalty, glue or anything else
5292             last_class = nil
5293         end
5294     end
5295     lang.hyphenate(head)
5296 end
5297 }%
5298 \bb1@luahyphenate}
5299 \gdef\bb1@luahyphenate{%
5300 \let\bb1@luahyphenate\relax
5301 \directlua{
5302     luatexbase.add_to_callback('hyphenate',
5303     function (head, tail)
5304         if Babel.linebreaking.before then

```

```

5305     for k, func in ipairs(Babel.linebreaking.before) do
5306         func(head)
5307     end
5308 end
5309 if Babel.cjk_enabled then
5310     Babel.cjk_linebreak(head)
5311 end
5312 lang.hyphenate(head)
5313 if Babel.linebreaking.after then
5314     for k, func in ipairs(Babel.linebreaking.after) do
5315         func(head)
5316     end
5317 end
5318 if Babel.sea_enabled then
5319     Babel.sea_disc_to_space(head)
5320 end
5321 end,
5322 'Babel.hyphenate')
5323 }
5324 }
5325 \endgroup
5326 \def\bbl@provide@intraspace{%
5327     \bbl@ifunset{\bbl@intsp@{language}}{}%
5328     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5329         \bbl@xin@{/c}{\bbl@cl{lbrk}}}%
5330     \ifin@           % cjk
5331         \bbl@cjk@intraspace
5332         \directlua{
5333             Babel = Babel or {}
5334             Babel.locale_props = Babel.locale_props or {}
5335             Babel.locale_props[\the\localeid].linebreak = 'c'
5336         }%
5337         \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@cl{}%
5338         \ifx\bbl@KVP@intrapenalty\@nil
5339             \bbl@intrapenalty0\@
5340         \fi
5341     \else           % sea
5342         \bbl@sea@intraspace
5343         \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@cl{}%
5344         \directlua{
5345             Babel = Babel or {}
5346             Babel.sea_ranges = Babel.sea_ranges or {}
5347             Babel.set_chranges('\bbl@cl{sbc}',
5348                 '\bbl@cl{chrng}')
5349         }%
5350         \ifx\bbl@KVP@intrapenalty\@nil
5351             \bbl@intrapenalty0\@
5352         \fi
5353     \fi
5354 \fi
5355 \ifx\bbl@KVP@intrapenalty\@nil\else
5356     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5357 \fi}}

```

13.6 Arabic justification

```

5358 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5359 \def\bblar@chars{%
5360     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%

```

```

5361 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5362 0640,0641,0642,0643,0644,0645,0646,0647,0649}
5363 \def\bblar@elongated{%
5364 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5365 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5366 0649,064A}
5367 \begingroup
5368 \catcode`\_ =11 \catcode`\:=11
5369 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5370 \endgroup
5371 \gdef\bbl@arabicjust{%
5372 \let\bbl@arabicjust\relax
5373 \newattribute\bblar@kashida
5374 \bblar@kashida=\z@
5375 \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@parsejalt}}%
5376 \directlua{
5377   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5378   Babel.arabic.elong_map[\the\localeid] = {}
5379   luatexbase.add_to_callback('post_linebreak_filter',
5380     Babel.arabic.justify, 'Babel.arabic.justify')
5381 %   luatexbase.add_to_callback('hpack_filter',
5382 %     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5383 }%
5384 % Save both node lists to make replacement. TODO. Save also widths to
5385 % make computations
5386 \def\bblar@fetchjalt#1#2#3#4{%
5387 \bbl@exp{\bbl@foreach{#1}}{%
5388 \bbl@ifunset\bblar@JE##1}%
5389 {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5390 {\setbox\z@\hbox{^^^200d\char"\@nameuse\bblar@JE##1#2}}%
5391 \directlua{%
5392   local last = nil
5393   for item in node.traverse(tex.box[0].head) do
5394     if item.id == node.id'glyph' and item.char > 0x600 and
5395       not (item.char == 0x200D) then
5396       last = item
5397     end
5398   end
5399   Babel.arabic.#3['##1#4'] = last.char
5400 }}
5401 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5402 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5403 % positioning?
5404 \gdef\bbl@parsejalt{%
5405 \ifx\addfontfeature\@undefined\else
5406 \bbl@xin{/e}{/\bbl@c1{lbrk}}%
5407 \ifin@
5408 \directlua{%
5409   if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5410     Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5411     tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5412   end
5413 }%
5414 \fi
5415 \fi}
5416 \gdef\bbl@parsejalti{%
5417 \begingroup
5418 \let\bbl@parsejalt\relax % To avoid infinite loop
5419 \edef\bbl@tempb{\fontid\font}%

```

```

5420 \bblar@nofswarn
5421 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5422 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5423 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5424 \addfontfeature{RawFeature+=jalt}%
5425 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5426 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5427 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5428 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5429 \directlua{%
5430     for k, v in pairs(Babel.arabic.from) do
5431         if Babel.arabic.dest[k] and
5432             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5433             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5434                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5435         end
5436     end
5437 }%
5438 \endgroup}
5439 %
5440 \begingroup
5441 \catcode`#=11
5442 \catcode`-=11
5443 \directlua{
5444
5445 Babel.arabic = Babel.arabic or {}
5446 Babel.arabic.from = {}
5447 Babel.arabic.dest = {}
5448 Babel.arabic.justify_factor = 0.95
5449 Babel.arabic.justify_enabled = true
5450
5451 function Babel.arabic.justify(head)
5452   if not Babel.arabic.justify_enabled then return head end
5453   for line in node.traverse_id(node.id'hlist', head) do
5454     Babel.arabic.justify_hlist(head, line)
5455   end
5456   return head
5457 end
5458
5459 function Babel.arabic.justify_hbox(head, gc, size, pack)
5460   if Babel.arabic.justify_enabled and pack == 'exactly' then
5461     Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5462   end
5463   return head
5464 end
5465
5466 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5467   local d, new
5468   local k_list, k_item, pos_inline
5469   local width, width_new, full, k_curr, wt_pos, goal, shift
5470   local subst_done = false
5471   local elong_map = Babel.arabic.elong_map
5472   local last_line
5473   local GLYPH = node.id'glyph'
5474   local KASHIDA = luatexbase.registernumber'bblar@kashida'
5475   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5476
5477   if line == nil then
5478     line = {}

```

```

5479     line.glue_sign = 1
5480     line.glue_order = 0
5481     line.head = head
5482     line.shift = 0
5483     line.width = size
5484 end
5485
5486 % Exclude last line. todo. But-- it discards one-word lines, too!
5487 % ? Look for glue = 12:15
5488 if (line.glue_sign == 1 and line.glue_order == 0) then
5489     elongs = {}      % Stores elongated candidates of each line
5490     k_list = {}      % And all letters with kashida
5491     pos_inline = 0   % Not yet used
5492
5493     for n in node.traverse_id(GLYPH, line.head) do
5494         pos_inline = pos_inline + 1 % To find where it is. Not used.
5495
5496         % Elongated glyphs
5497         if elong_map then
5498             local locale = node.get_attribute(n, LOCALE)
5499             if elong_map[locale] and elong_map[locale][n.font] and
5500                 elong_map[locale][n.font][n.char] then
5501                 table.insert(elongs, {node = n, locale = locale} )
5502                 node.set_attribute(n.prev, KASHIDA, 0)
5503             end
5504         end
5505
5506         % Tatwil
5507         if Babel.kashida_wts then
5508             local k_wt = node.get_attribute(n, KASHIDA)
5509             if k_wt > 0 then % todo. parameter for multi inserts
5510                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5511             end
5512         end
5513     end % of node.traverse_id
5514
5515     if #elongs == 0 and #k_list == 0 then goto next_line end
5516     full = line.width
5517     shift = line.shift
5518     goal = full * Babel.arabic.justify_factor % A bit crude
5519     width = node.dimensions(line.head)      % The 'natural' width
5520
5521     % == Elongated ==
5522     % Original idea taken from 'chickenize'
5523     while (#elongs > 0 and width < goal) do
5524         subst_done = true
5525         local x = #elongs
5526         local curr = elongs[x].node
5527         local oldchar = curr.char
5528         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5529         width = node.dimensions(line.head) % Check if the line is too wide
5530         % Substitute back if the line would be too wide and break:
5531         if width > goal then
5532             curr.char = oldchar
5533             break
5534         end
5535         % If continue, pop the just substituted node from the list:
5536         table.remove(elongs, x)
5537     end

```

```

5538     end
5539
5540     % == Tatwil ==
5541     if #k_list == 0 then goto next_line end
5542
5543     width = node.dimensions(line.head)    % The 'natural' width
5544     k_curr = #k_list
5545     wt_pos = 1
5546
5547     while width < goal do
5548         subst_done = true
5549         k_item = k_list[k_curr].node
5550         if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5551             d = node.copy(k_item)
5552             d.char = 0x0640
5553             line.head, new = node.insert_after(line.head, k_item, d)
5554             width_new = node.dimensions(line.head)
5555             if width > goal or width == width_new then
5556                 node.remove(line.head, new) % Better compute before
5557                 break
5558             end
5559             width = width_new
5560         end
5561         if k_curr == 1 then
5562             k_curr = #k_list
5563             wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5564         else
5565             k_curr = k_curr - 1
5566         end
5567     end
5568
5569     ::next_line::
5570
5571     % Must take into account marks and ins, see luatex manual.
5572     % Have to be executed only if there are changes. Investigate
5573     % what's going on exactly.
5574     if subst_done and not gc then
5575         d = node.hpack(line.head, full, 'exactly')
5576         d.shift = shift
5577         node.insert_before(head, line, d)
5578         node.remove(head, line)
5579     end
5580 end % if process line
5581 end
5582 }
5583 \endgroup
5584 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5585 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5586 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5587 \DisableBabelHook{babel-fontspec}
5588 <<Font selection>>

```

13.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an

intermediate table built on the fly for optimization). This locale is then used to get the \language and the \localeid as stored in locale_props, as well as the font (as requested). In the latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5589% TODO - to a lua file
5590 \directlua{
5591 Babel.script_blocks = {
5592   ['dflt'] = {},
5593   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5594              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5595   ['Armn'] = {{0x0530, 0x058F}},
5596   ['Beng'] = {{0x0980, 0x09FF}},
5597   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5598   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5599   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5600              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5601   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5602   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5603              {0xAB00, 0xAB2F}},
5604   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5605   % Don't follow strictly Unicode, which places some Coptic letters in
5606   % the 'Greek and Coptic' block
5607   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5608   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5609              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5610              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5611              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5612              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5613              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5614   ['Hebr'] = {{0x0590, 0x05FF}},
5615   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5616              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5617   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5618   ['Knda'] = {{0x0C80, 0x0CFF}},
5619   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5620              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5621              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5622   ['Lao0'] = {{0x0E80, 0x0EFF}},
5623   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5624              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5625              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5626   ['Mahj'] = {{0x11150, 0x1117F}},
5627   ['Mlym'] = {{0x0D00, 0x0D7F}},
5628   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5629   ['Orya'] = {{0x0B00, 0x0B7F}},
5630   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5631   ['Syr1'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5632   ['Taml'] = {{0x0B80, 0x0BFF}},
5633   ['Telu'] = {{0x0C00, 0x0C7F}},
5634   ['Tfng'] = {{0x2D30, 0x2D7F}},
5635   ['Thai'] = {{0x0E00, 0x0E7F}},
5636   ['Tibt'] = {{0x0F00, 0x0FFF}},
5637   ['Vaii'] = {{0xA500, 0xA63F}},
5638   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5639 }
5640
5641 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyr1
5642 Babel.script_blocks.Hant = Babel.script_blocks.Hans

```

```

5643 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5644
5645 function Babel.locale_map(head)
5646   if not Babel.locale_mapped then return head end
5647
5648   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5649   local GLYPH = node.id('glyph')
5650   local inmath = false
5651   local toloc_save
5652   for item in node.traverse(head) do
5653     local toloc
5654     if not inmath and item.id == GLYPH then
5655       % Optimization: build a table with the chars found
5656       if Babel.chr_to_loc[item.char] then
5657         toloc = Babel.chr_to_loc[item.char]
5658       else
5659         for lc, maps in pairs(Babel.loc_to_scr) do
5660           for _, rg in pairs(maps) do
5661             if item.char >= rg[1] and item.char <= rg[2] then
5662               Babel.chr_to_loc[item.char] = lc
5663               toloc = lc
5664               break
5665             end
5666           end
5667         end
5668       end
5669       % Now, take action, but treat composite chars in a different
5670       % fashion, because they 'inherit' the previous locale. Not yet
5671       % optimized.
5672       if not toloc and
5673         (item.char >= 0x0300 and item.char <= 0x036F) or
5674         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5675         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5676         toloc = toloc_save
5677       end
5678       if toloc and toloc > -1 then
5679         if Babel.locale_props[toloc].lg then
5680           item.lang = Babel.locale_props[toloc].lg
5681           node.set_attribute(item, LOCALE, toloc)
5682         end
5683         if Babel.locale_props[toloc]['/'..item.font] then
5684           item.font = Babel.locale_props[toloc]['/'..item.font]
5685         end
5686         toloc_save = toloc
5687       end
5688     elseif not inmath and item.id == 7 then
5689       item.replace = item.replace and Babel.locale_map(item.replace)
5690       item.pre      = item.pre and Babel.locale_map(item.pre)
5691       item.post     = item.post and Babel.locale_map(item.post)
5692     elseif item.id == node.id'math' then
5693       inmath = (item.subtype == 0)
5694     end
5695   end
5696   return head
5697 end
5698 }

```

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

```

5699 \newcommand\babelcharproperty[1]{%
5700   \count@=#1\relax
5701   \ifvmode
5702     \expandafter\babel@chprop
5703   \else
5704     \babel@error{\string\babelcharproperty\space can be used only in\%
5705               vertical mode (preamble or between paragraphs)}%
5706               {See the manual for futher info}%
5707   \fi}
5708 \newcommand\babel@chprop[3][\the\count@]{%
5709   \@tempcnta=#1\relax
5710   \babel@ifunset{\babel@chprop@#2}%
5711   {\babel@error{No property named '#2'. Allowed values are\%
5712             direction (bc), mirror (bmg), and linebreak (lb)}%
5713             {See the manual for futher info}}%
5714   {%
5715   \loop
5716     \babel@cs{\babel@chprop@#2}{#3}%
5717   \ifnum\count@<\@tempcnta
5718     \advance\count@\@ne
5719   \repeat}
5720 \def\babel@chprop@direction#1{%
5721   \directlua{
5722     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5723     Babel.characters[\the\count@]['d'] = '#1'
5724   }}
5725 \let\babel@chprop@bc\babel@chprop@direction
5726 \def\babel@chprop@mirror#1{%
5727   \directlua{
5728     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5729     Babel.characters[\the\count@]['m'] = '\number#1'
5730   }}
5731 \let\babel@chprop@bmg\babel@chprop@mirror
5732 \def\babel@chprop@linebreak#1{%
5733   \directlua{
5734     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5735     Babel.cjk_characters[\the\count@]['c'] = '#1'
5736   }}
5737 \let\babel@chprop@lb\babel@chprop@linebreak
5738 \def\babel@chprop@locale#1{%
5739   \directlua{
5740     Babel.chr_to_loc = Babel.chr_to_loc or {}
5741     Babel.chr_to_loc[\the\count@] =
5742       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
5743   }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionary, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

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

```

5744 \begingroup % TODO - to a lua file
5745 \catcode`\~ = 12
5746 \catcode`\# = 12
5747 \catcode`\% = 12
5748 \catcode`\& = 14
5749 \directlua{
5750   Babel.linebreaking.replacements = {}
5751   Babel.linebreaking.replacements[0] = {}  %% pre
5752   Babel.linebreaking.replacements[1] = {}  %% post
5753
5754   %% Discretionaries contain strings as nodes
5755   function Babel.str_to_nodes(fn, matches, base)
5756     local n, head, last
5757     if fn == nil then return nil end
5758     for s in string.utfvalues(fn(matches)) do
5759       if base.id == 7 then
5760         base = base.replace
5761       end
5762       n = node.copy(base)
5763       n.char = s
5764       if not head then
5765         head = n
5766       else
5767         last.next = n
5768       end
5769       last = n
5770     end
5771     return head
5772   end
5773
5774   Babel.fetch_subtext = {}
5775
5776   Babel.ignore_pre_char = function(node)
5777     return (node.lang == \the\l@nohyphenation)
5778   end
5779
5780   %% Merging both functions doesn't seem feasible, because there are too
5781   %% many differences.
5782   Babel.fetch_subtext[0] = function(head)
5783     local word_string = ''
5784     local word_nodes = {}
5785     local lang
5786     local item = head
5787     local inmath = false
5788
5789     while item do
5790
5791       if item.id == 11 then
5792         inmath = (item.subtype == 0)
5793       end
5794
5795       if inmath then
5796         %% pass
5797       elseif item.id == 29 then
5798         local locale = node.get_attribute(item, Babel.attr_locale)
5799
5800         if lang == locale or lang == nil then
5801           lang = lang or locale
5802         end
5803       end
5804       item = item.next
5805     end
5806
5807     return word_string
5808   end
5809 }

```

```

5803         if Babel.ignore_pre_char(item) then
5804             word_string = word_string .. Babel.us_char
5805         else
5806             word_string = word_string .. unicode.utf8.char(item.char)
5807         end
5808         word_nodes[#word_nodes+1] = item
5809     else
5810         break
5811     end
5812
5813     elseif item.id == 12 and item.subtype == 13 then
5814         word_string = word_string .. ' '
5815         word_nodes[#word_nodes+1] = item
5816
5817         %% Ignore leading unrecognized nodes, too.
5818         elseif word_string ~= '' then
5819             word_string = word_string .. Babel.us_char
5820             word_nodes[#word_nodes+1] = item    %% Will be ignored
5821         end
5822
5823         item = item.next
5824     end
5825
5826     %% Here and above we remove some trailing chars but not the
5827     %% corresponding nodes. But they aren't accessed.
5828     if word_string:sub(-1) == ' ' then
5829         word_string = word_string:sub(1,-2)
5830     end
5831     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5832     return word_string, word_nodes, item, lang
5833 end
5834
5835 Babel.fetch_subtext[1] = function(head)
5836     local word_string = ''
5837     local word_nodes = {}
5838     local lang
5839     local item = head
5840     local inmath = false
5841
5842     while item do
5843
5844         if item.id == 11 then
5845             inmath = (item.subtype == 0)
5846         end
5847
5848         if inmath then
5849             %% pass
5850
5851         elseif item.id == 29 then
5852             if item.lang == lang or lang == nil then
5853                 if (item.char ~= 124) and (item.char ~= 61) then %% not =, not |
5854                     lang = lang or item.lang
5855                     word_string = word_string .. unicode.utf8.char(item.char)
5856                     word_nodes[#word_nodes+1] = item
5857                 end
5858             else
5859                 break
5860             end
5861

```

```

5862     elseif item.id == 7 and item.subtype == 2 then
5863         word_string = word_string .. '='
5864         word_nodes[#word_nodes+1] = item
5865
5866     elseif item.id == 7 and item.subtype == 3 then
5867         word_string = word_string .. '|'
5868         word_nodes[#word_nodes+1] = item
5869
5870     && (1) Go to next word if nothing was found, and (2) implicitly
5871     && remove leading USs.
5872     elseif word_string == '' then
5873         && pass
5874
5875     && This is the responsible for splitting by words.
5876     elseif (item.id == 12 and item.subtype == 13) then
5877         break
5878
5879     else
5880         word_string = word_string .. Babel.us_char
5881         word_nodes[#word_nodes+1] = item && Will be ignored
5882     end
5883
5884     item = item.next
5885 end
5886
5887 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5888 return word_string, word_nodes, item, lang
5889 end
5890
5891 function Babel.pre_hyphenate_replace(head)
5892     Babel.hyphenate_replace(head, 0)
5893 end
5894
5895 function Babel.post_hyphenate_replace(head)
5896     Babel.hyphenate_replace(head, 1)
5897 end
5898
5899 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5900     local ss = ''
5901     for pp = 1, 40 do
5902         if wn[pp] then
5903             if wn[pp].id == 29 then
5904                 ss = ss .. unicode.utf8.char(wn[pp].char)
5905             else
5906                 ss = ss .. '{' .. wn[pp].id .. '}'
5907             end
5908         end
5909     end
5910     print('nod', ss)
5911     print('lst_m',
5912         string.rep(' ', unicode.utf8.len(
5913             string.sub(w, 1, last_match))-1) .. '>')
5914     print('str', w)
5915     print('sc', string.rep(' ', sc-1) .. '^')
5916     if first == last then
5917         print('f=l', string.rep(' ', first-1) .. '!')
5918     else
5919         print('f/l', string.rep(' ', first-1) .. '[' ..
5920             string.rep(' ', last-first-1) .. ']')

```

```

5921     end
5922 end
5923
5924 Babel.us_char = string.char(31)
5925
5926 function Babel.hyphenate_replace(head, mode)
5927     local u = unicode.utf8
5928     local lbkr = Babel.linebreaking.replacements[mode]
5929
5930     local word_head = head
5931
5932     while true do    %% for each subtext block
5933
5934         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5935
5936         if Babel.debug then
5937             print()
5938             print((mode == 0) and '@@@<' or '@@@>', w)
5939         end
5940
5941         if nw == nil and w == '' then break end
5942
5943         if not lang then goto next end
5944         if not lbkr[lang] then goto next end
5945
5946         %% For each saved (pre|post)hyphenation. TODO. Reconsider how
5947         %% loops are nested.
5948         for k=1, #lbkr[lang] do
5949             local p = lbkr[lang][k].pattern
5950             local r = lbkr[lang][k].replace
5951
5952             if Babel.debug then
5953                 print('*****', p, mode)
5954             end
5955
5956             %% This variable is set in some cases below to the first *byte*
5957             %% after the match, either as found by u.match (faster) or the
5958             %% computed position based on sc if w has changed.
5959             local last_match = 0
5960             local step = 0
5961
5962             %% For every match.
5963             while true do
5964                 if Babel.debug then
5965                     print('====')
5966                 end
5967                 local new    %% used when inserting and removing nodes
5968
5969                 local matches = { u.match(w, p, last_match) }
5970
5971                 if #matches < 2 then break end
5972
5973                 %% Get and remove empty captures (with ()'s, which return a
5974                 %% number with the position), and keep actual captures
5975                 %% (from (...)), if any, in matches.
5976                 local first = table.remove(matches, 1)
5977                 local last  = table.remove(matches, #matches)
5978                 %% Non re-fetched substrings may contain \31, which separates
5979                 %% subsubstrings.

```

```

5980         if string.find(w:sub(first, last-1), Babel.us_char) then break end
5981
5982         local save_last = last && with A()BC()D, points to D
5983
5984         && Fix offsets, from bytes to unicode. Explained above.
5985         first = u.len(w:sub(1, first-1)) + 1
5986         last = u.len(w:sub(1, last-1)) && now last points to C
5987
5988         && This loop stores in n small table the nodes
5989         && corresponding to the pattern. Used by 'data' to provide a
5990         && predictable behavior with 'insert' (now w_nodes is modified on
5991         && the fly), and also access to 'remove'd nodes.
5992         local sc = first-1 && Used below, too
5993         local data_nodes = {}
5994
5995         for q = 1, last-first+1 do
5996             data_nodes[q] = w_nodes[sc+q]
5997         end
5998
5999         && This loop traverses the matched substring and takes the
6000         && corresponding action stored in the replacement list.
6001         && sc = the position in substr nodes / string
6002         && rc = the replacement table index
6003         local rc = 0
6004
6005         while rc < last-first+1 do && for each replacement
6006             if Babel.debug then
6007                 print('.....', rc + 1)
6008             end
6009             sc = sc + 1
6010             rc = rc + 1
6011
6012             if Babel.debug then
6013                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6014                 local ss = ''
6015                 for itt in node.traverse(head) do
6016                     if itt.id == 29 then
6017                         ss = ss .. unicode.utf8.char(itt.char)
6018                     else
6019                         ss = ss .. '{' .. itt.id .. '}'
6020                     end
6021                 end
6022                 print('*****', ss)
6023             end
6024
6025             local crep = r[rc]
6026             local item = w_nodes[sc]
6027             local item_base = item
6028             local placeholder = Babel.us_char
6029             local d
6030
6031             if crep and crep.data then
6032                 item_base = data_nodes[crep.data]
6033             end
6034
6035             if crep then
6036                 step = crep.step or 0
6037             end
6038

```

```

6039
6040     if crep and next(crep) == nil then &% = {}
6041         last_match = save_last    &% Optimization
6042         goto next
6043
6044     elseif crep == nil or crep.remove then
6045         node.remove(head, item)
6046         table.remove(w_nodes, sc)
6047         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6048         sc = sc - 1    &% Nothing has been inserted.
6049         last_match = utf8.offset(w, sc+1+step)
6050         goto next
6051
6052     elseif crep and crep.kashida then &% Experimental
6053         node.set_attribute(item,
6054             luatexbase.registernumber'bblar@kashida',
6055             crep.kashida)
6056         last_match = utf8.offset(w, sc+1+step)
6057         goto next
6058
6059     elseif crep and crep.string then
6060         local str = crep.string(matches)
6061         if str == '' then &% Gather with nil
6062             node.remove(head, item)
6063             table.remove(w_nodes, sc)
6064             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6065             sc = sc - 1    &% Nothing has been inserted.
6066         else
6067             local loop_first = true
6068             for s in string.utfvalues(str) do
6069                 d = node.copy(item_base)
6070                 d.char = s
6071                 if loop_first then
6072                     loop_first = false
6073                     head, new = node.insert_before(head, item, d)
6074                     if sc == 1 then
6075                         word_head = head
6076                     end
6077                     w_nodes[sc] = d
6078                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6079                 else
6080                     sc = sc + 1
6081                     head, new = node.insert_before(head, item, d)
6082                     table.insert(w_nodes, sc, new)
6083                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6084                 end
6085                 if Babel.debug then
6086                     print('.....', 'str')
6087                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6088                 end
6089             end &% for
6090             node.remove(head, item)
6091         end &% if ''
6092         last_match = utf8.offset(w, sc+1+step)
6093         goto next
6094
6095     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6096         d = node.new(7, 0)    &% (disc, discretionary)
6097         d.pre    = Babel.str_to_nodes(crep.pre, matches, item_base)

```

```

6098     d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6099     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6100     d.attr = item_base.attr
6101     if crep.pre == nil then  &% TeXbook p96
6102         d.penalty = crep.penalty or tex.hyphenpenalty
6103     else
6104         d.penalty = crep.penalty or tex.exhyphenpenalty
6105     end
6106     placeholder = '|'
6107     head, new = node.insert_before(head, item, d)
6108
6109 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6110     &% ERROR
6111
6112 elseif crep and crep.penalty then
6113     d = node.new(14, 0)  &% (penalty, userpenalty)
6114     d.attr = item_base.attr
6115     d.penalty = crep.penalty
6116     head, new = node.insert_before(head, item, d)
6117
6118 elseif crep and crep.space then
6119     &% 655360 = 10 pt = 10 * 65536 sp
6120     d = node.new(12, 13)  &% (glue, spaceskip)
6121     local quad = font.getfont(item_base.font).size or 655360
6122     node.setglue(d, crep.space[1] * quad,
6123                  crep.space[2] * quad,
6124                  crep.space[3] * quad)
6125     if mode == 0 then
6126         placeholder = ' '
6127     end
6128     head, new = node.insert_before(head, item, d)
6129
6130 elseif crep and crep.spacefactor then
6131     d = node.new(12, 13)  &% (glue, spaceskip)
6132     local base_font = font.getfont(item_base.font)
6133     node.setglue(d,
6134                  crep.spacefactor[1] * base_font.parameters['space'],
6135                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6136                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6137     if mode == 0 then
6138         placeholder = ' '
6139     end
6140     head, new = node.insert_before(head, item, d)
6141
6142 elseif mode == 0 and crep and crep.space then
6143     &% ERROR
6144
6145 end  &% ie replacement cases
6146
6147 &% Shared by disc, space and penalty.
6148 if sc == 1 then
6149     word_head = head
6150 end
6151 if crep.insert then
6152     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6153     table.insert(w_nodes, sc, new)
6154     last = last + 1
6155 else
6156     w_nodes[sc] = d

```

```

6157         node.remove(head, item)
6158         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6159     end
6160
6161     last_match = utf8.offset(w, sc+1+step)
6162
6163     ::next::
6164
6165     end && for each replacement
6166
6167     if Babel.debug then
6168         print('.....', '/')
6169         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6170     end
6171
6172     end && for match
6173
6174     end && for patterns
6175
6176     ::next::
6177     word_head = nw
6178     end && for substring
6179     return head
6180 end
6181
6182 && This table stores capture maps, numbered consecutively
6183 Babel.capture_maps = {}
6184
6185 && The following functions belong to the next macro
6186 function Babel.capture_func(key, cap)
6187     local ret = "[" .. cap:gsub('{{[0-9]}}', "].m[%1]..[" .. "]"
6188     local cnt
6189     local u = unicode.utf8
6190     ret, cnt = ret:gsub('{{[0-9]}|([^\]|+)|(.-)}', Babel.capture_func_map)
6191     if cnt == 0 then
6192         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
6193             function (n)
6194                 return u.char(tonumber(n, 16))
6195             end)
6196     end
6197     ret = ret:gsub("%[%[%]]%.", '')
6198     ret = ret:gsub("%.%[%[%]]%", '')
6199     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6200 end
6201
6202 function Babel.capt_map(from, mapno)
6203     return Babel.capture_maps[mapno][from] or from
6204 end
6205
6206 && Handle the {n|abc|ABC} syntax in captures
6207 function Babel.capture_func_map(capno, from, to)
6208     local u = unicode.utf8
6209     from = u.gsub(from, '{{(%x%x%x%x+)}}',
6210         function (n)
6211             return u.char(tonumber(n, 16))
6212         end)
6213     to = u.gsub(to, '{{(%x%x%x%x+)}}',
6214         function (n)
6215             return u.char(tonumber(n, 16))

```

```

6216         end)
6217     local froms = {}
6218     for s in string.utfcharacters(from) do
6219         table.insert(froms, s)
6220     end
6221     local cnt = 1
6222     table.insert(Babel.capture_maps, {})
6223     local mlen = table.getn(Babel.capture_maps)
6224     for s in string.utfcharacters(to) do
6225         Babel.capture_maps[mlen][froms[cnt]] = s
6226         cnt = cnt + 1
6227     end
6228     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6229         (mlen) .. ").." .. "["
6230 end
6231
6232 &% Create/Extend reversed sorted list of kashida weights:
6233 function Babel.capture_kashida(key, wt)
6234     wt = tonumber(wt)
6235     if Babel.kashida_wts then
6236         for p, q in ipairs(Babel.kashida_wts) do
6237             if wt == q then
6238                 break
6239             elseif wt > q then
6240                 table.insert(Babel.kashida_wts, p, wt)
6241                 break
6242             elseif table.getn(Babel.kashida_wts) == p then
6243                 table.insert(Babel.kashida_wts, wt)
6244             end
6245         end
6246     else
6247         Babel.kashida_wts = { wt }
6248     end
6249     return 'kashida = ' .. wt
6250 end
6251 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, `pre={1}{1}` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

6252 \catcode`\#=6
6253 \gdef\babelposthyphenation#1#2#3{&%
6254     \bbl@activateposthyphen
6255     \beginingroup
6256     \def\babeltempa{\bbl@add@list\babeltempb}&%
6257     \let\babeltempb\@empty
6258     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6259     \bbl@replace\bbl@tempa{,}{ ,}&%
6260     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6261         \bbl@ifsamestring{##1}{remove}&%
6262         {\bbl@add@list\babeltempb{nil}}&%
6263         {\directlua{
6264             local rep = [= [##1]=]

```

```

6265         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6266         rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
6267         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6268         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6269         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6270         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6271         tex.print([[\\string\\babeltempa{[]] .. rep .. [[]}]]])
6272     }&&%
6273 \\directlua{
6274     local lbkr = Babel.linebreaking.replacements[1]
6275     local u = unicode.utf8
6276     local id = \\the\\csname l@#1\\endcsname
6277     &% Convert pattern:
6278     local patt = string.gsub(==[#2]==, '%s', '')
6279     if not u.find(patt, '()', nil, true) then
6280         patt = '()' .. patt .. '()'
6281     end
6282     patt = string.gsub(patt, '%(%)%\\', '^()')
6283     patt = string.gsub(patt, '%$(%)', '()$')
6284     patt = u.gsub(patt, '{(.)}',
6285         function (n)
6286             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6287         end)
6288     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6289         function (n)
6290             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6291         end)
6292     lbkr[id] = lbkr[id] or {}
6293     table.insert(lbkr[id], { pattern = patt, replace = { \\babeltempb } })
6294 }&&%
6295 \\endgroup}
6296 % TODO. Copypaste pattern.
6297 \\gdef\\babelprehyphenation#1#2#3{&%
6298     \\bbl@activateprehyphen
6299     \\begin{group}
6300     \\def\\babeltempa{\\bbl@add@list\\babeltempb}&%
6301     \\let\\babeltempb\\empty
6302     \\def\\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6303     \\bbl@replace\\bbl@tempa{,}{ ,}&%
6304     \\expandafter\\bbl@foreach\\expandafter{\\bbl@tempa}{&%
6305         \\bbl@ifsamestring{##1}{remove}&%
6306         {\\bbl@add@list\\babeltempb{nil}}&%
6307     }\\directlua{
6308         local rep = [=[#1]=]
6309         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6310         rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
6311         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6312         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6313             'space = { ' .. '%2, %3, %4' .. ' }')
6314         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6315             'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6316         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6317         tex.print([[\\string\\babeltempa{[]] .. rep .. [[]}]]])
6318     }&&%
6319 \\directlua{
6320     local lbkr = Babel.linebreaking.replacements[0]
6321     local u = unicode.utf8
6322     local id = \\the\\csname bbl@id@@#1\\endcsname
6323     &% Convert pattern:

```

```

6324     local patt = string.gsub(#[#2]==, '%s', '')
6325     local patt = string.gsub(patt, '|', ' ')
6326     if not u.find(patt, '()', nil, true) then
6327         patt = '()' .. patt .. '()'
6328     end
6329     &% patt = string.gsub(patt, '%(%)%', '^()')
6330     &% patt = string.gsub(patt, '([%^%])%$%(%)', '%1()$')
6331     patt = u.gsub(patt, '{(.)}',
6332         function (n)
6333             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6334         end)
6335     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6336         function (n)
6337             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6338         end)
6339     lbkr[id] = lbkr[id] or {}
6340     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6341 }&%
6342 \endgroup}
6343 \endgroup
6344 \def\bbl@activateposthyphen{%
6345 \let\bbl@activateposthyphen\relax
6346 \directlua{
6347     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6348 }}
6349 \def\bbl@activateprehyphen{%
6350 \let\bbl@activateprehyphen\relax
6351 \directlua{
6352     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6353 }}

```

13.9 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

6354 \bbl@trace{Redefinitions for bidi layout}
6355 \ifx\@eqnnum\undefined\else
6356 \ifx\bbl@attr@dir\undefined\else
6357 \edef\@eqnnum{%
6358     \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6359     \unexpanded\expandafter{\@eqnnum}}
6360 \fi
6361 \fi
6362 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
6363 \ifnum\bbl@bidimode>\z@
6364 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6365     \bbl@exp{%
6366         \mathdir\the\bodydir

```

```

6367      #1%                Once entered in math, set boxes to restore values
6368      \<ifmmode>%
6369      \everyvbox{%
6370      \the\everyvbox
6371      \bodydir\the\bodydir
6372      \mathdir\the\mathdir
6373      \everyhbox{\the\everyhbox}%
6374      \everyvbox{\the\everyvbox}}%
6375      \everyhbox{%
6376      \the\everyhbox
6377      \bodydir\the\bodydir
6378      \mathdir\the\mathdir
6379      \everyhbox{\the\everyhbox}%
6380      \everyvbox{\the\everyvbox}}%
6381      \<fi>}}%
6382      \def\@hangfrom#1{%
6383      \setbox\@tempboxa\hbox{{#1}}%
6384      \hangindent\wd\@tempboxa
6385      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6386      \shapemode\@ne
6387      \fi
6388      \noindent\box\@tempboxa}
6389      \fi
6390      \IfBabelLayout{tabular}
6391      {\let\bbl@OL@tabular\@tabular
6392      \bbl@replace\@tabular{$$}{\bbl@nextfake$}%
6393      \let\bbl@NL@tabular\@tabular
6394      \AtBeginDocument{%
6395      \ifx\bbl@NL@tabular\@tabular\else
6396      \bbl@replace\@tabular{$$}{\bbl@nextfake$}%
6397      \let\bbl@NL@tabular\@tabular
6398      \fi}}
6399      {}
6400      \IfBabelLayout{lists}
6401      {\let\bbl@OL@list\list
6402      \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6403      \let\bbl@NL@list\list
6404      \def\bbl@listparshape#1#2#3{%
6405      \parshape #1 #2 #3 %
6406      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6407      \shapemode\tw@
6408      \fi}}
6409      {}
6410      \IfBabelLayout{graphics}
6411      {\let\bbl@pictresetdir\relax
6412      \def\bbl@pictsetdir#1{%
6413      \ifcase\bbl@thetextdir
6414      \let\bbl@pictresetdir\relax
6415      \else
6416      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6417      \or\textdir TLT
6418      \else\bodydir TLT \textdir TLT
6419      \fi
6420      % \(\text|par)dir required in pgf:
6421      \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6422      \fi}%
6423      \ifx\AddToHook\undefined\else
6424      \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6425      \directlua{

```

```

6426     Babel.get_picture_dir = true
6427     Babel.picture_has_bidi = 0
6428     function Babel.picture_dir (head)
6429         if not Babel.get_picture_dir then return head end
6430         for item in node.traverse(head) do
6431             if item.id == node.id'glyph' then
6432                 local itemchar = item.char
6433                 % TODO. Copypaste pattern from Babel.bidi (-r)
6434                 local chardata = Babel.characters[itemchar]
6435                 local dir = chardata and chardata.d or nil
6436                 if not dir then
6437                     for nn, et in ipairs(Babel.ranges) do
6438                         if itemchar < et[1] then
6439                             break
6440                         elseif itemchar <= et[2] then
6441                             dir = et[3]
6442                             break
6443                         end
6444                     end
6445                 end
6446                 if dir and (dir == 'al' or dir == 'r') then
6447                     Babel.picture_has_bidi = 1
6448                 end
6449             end
6450         end
6451         return head
6452     end
6453     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6454         "Babel.picture_dir")
6455 }%
6456 \AtBeginDocument{%
6457     \long\def\put(#1,#2)#3{%
6458         \@killglue
6459         % Try:
6460         \ifx\bbl@pictresetdir\relax
6461             \def\bbl@tempc{0}%
6462         \else
6463             \directlua{
6464                 Babel.get_picture_dir = true
6465                 Babel.picture_has_bidi = 0
6466             }%
6467             \setbox\z@\hb@xt@\z@{%
6468                 \@defaultunitsset\@tempdimc{#1}\unitlength
6469                 \kern\@tempdimc
6470                 #3\hss}%
6471             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6472         \fi
6473         % Do:
6474         \@defaultunitsset\@tempdimc{#2}\unitlength
6475         \raise\@tempdimc\hb@xt@\z@{%
6476             \@defaultunitsset\@tempdimc{#1}\unitlength
6477             \kern\@tempdimc
6478             {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6479         \ignorespaces}%
6480         \MakeRobust\put}%
6481 \fi
6482 \AtBeginDocument
6483     {\ifx\tikz@atbegin@node\undefined\else
6484         \ifx\AddToHook\undefined\else % TODO. Still tentative.

```

```

6485      \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6486      \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6487      \fi
6488      \let\bbl@OL@pgfpicture\pgfpicture
6489      \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
6490      {\bbl@pictsetdir\z@\pgfpicturetrue}%
6491      \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6492      \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6493      \bbl@sreplace\tikz{\begin@group}%
6494      {\begin@group\bbl@pictsetdir\tw@}%
6495      \fi
6496      \ifx\AddToHook\undefined\else
6497      \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6498      \fi
6499      }}
6500  {}

```

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.

```

6501 \IfBabelLayout{counters}%
6502  {\let\bbl@OL@@textsuperscript\textsuperscript
6503   \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6504   \let\bbl@latinarabic=\@arabic
6505   \let\bbl@OL@@arabic\@arabic
6506   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6507   \@ifpackagewith{babel}{bidi=default}%
6508   {\let\bbl@asciroman=\@roman
6509    \let\bbl@OL@@roman\@roman
6510    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6511    \let\bbl@asciiRoman=\@Roman
6512    \let\bbl@OL@@roman\@Roman
6513    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6514    \let\bbl@OL@labelenumii\labelenumii
6515    \def\labelenumii{}\theenumii}%
6516    \let\bbl@OL@p@enumiii\p@enumiii
6517    \def\p@enumiii{\p@enumii}\theenumii{}\{}\{}\}
6518  }{Footnote changes}
6519 \IfBabelLayout{footnotes}%
6520  {\let\bbl@OL@footnote\footnote
6521   \BabelFootnote\footnote\languagename{}\}%
6522   \BabelFootnote\localfootnote\languagename{}\}%
6523   \BabelFootnote\mainfootnote{}\%\}%
6524  {}

```

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.

```

6525 \IfBabelLayout{extras}%
6526  {\let\bbl@OL@underline\underline
6527   \bbl@sreplace\underline{\$@\underline}{\bbl@nextfake$@\underline}%
6528   \let\bbl@OL@LaTeX2e\LaTeX2e
6529   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6530    \if b\expandafter\@car\@f@series\@nil\boldmath\fi
6531    \babelsublr{%
6532      \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6533  {}
6534 </luatex>

```

13.10 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

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

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

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

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

```
6535 (*basic-r)
6536 Babel = Babel or {}
6537
6538 Babel.bidi_enabled = true
6539
6540 require('babel-data-bidi.lua')
6541
6542 local characters = Babel.characters
6543 local ranges = Babel.ranges
6544
6545 local DIR = node.id("dir")
6546
6547 local function dir_mark(head, from, to, outer)
6548   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6549   local d = node.new(DIR)
6550   d.dir = '+' .. dir
6551   node.insert_before(head, from, d)
6552   d = node.new(DIR)
6553   d.dir = '-' .. dir
6554   node.insert_after(head, to, d)
6555 end
```

```

6556
6557 function Babel.bidi(head, ispar)
6558   local first_n, last_n      -- first and last char with nums
6559   local last_es              -- an auxiliary 'last' used with nums
6560   local first_d, last_d      -- first and last char in L/R block
6561   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):

```

6562   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6563   local strong_lr = (strong == 'l') and 'l' or 'r'
6564   local outer = strong
6565
6566   local new_dir = false
6567   local first_dir = false
6568   local inmath = false
6569
6570   local last_lr
6571
6572   local type_n = ''
6573
6574   for item in node.traverse(head) do
6575
6576     -- three cases: glyph, dir, otherwise
6577     if item.id == node.id'glyph'
6578       or (item.id == 7 and item.subtype == 2) then
6579
6580       local itemchar
6581       if item.id == 7 and item.subtype == 2 then
6582         itemchar = item.replace.char
6583       else
6584         itemchar = item.char
6585       end
6586       local chardata = characters[itemchar]
6587       dir = chardata and chardata.d or nil
6588       if not dir then
6589         for nn, et in ipairs(ranges) do
6590           if itemchar < et[1] then
6591             break
6592           elseif itemchar <= et[2] then
6593             dir = et[3]
6594             break
6595           end
6596         end
6597       end
6598       dir = dir or 'l'
6599       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).

```

6600   if new_dir then
6601     attr_dir = 0
6602     for at in node.traverse(item.attr) do
6603       if at.number == luatexbase.registernumber'bbl@attr@dir' then
6604         attr_dir = at.value % 3

```

```

6605         end
6606     end
6607     if attr_dir == 1 then
6608         strong = 'r'
6609     elseif attr_dir == 2 then
6610         strong = 'al'
6611     else
6612         strong = 'l'
6613     end
6614     strong_lr = (strong == 'l') and 'l' or 'r'
6615     outer = strong_lr
6616     new_dir = false
6617 end
6618
6619 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

6620     dir_real = dir          -- We need dir_real to set strong below
6621     if dir == 'al' then dir = 'r' end -- W3

```

By W2, there are no <en> <et> <es> if strong == <al>, only <an>. Therefore, there are not <et en> nor <en et>, W5 can be ignored, and W6 applied:

```

6622     if strong == 'al' then
6623         if dir == 'en' then dir = 'an' end          -- W2
6624         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6625         strong_lr = 'r'                             -- W3
6626     end

```

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

```

6627     elseif item.id == node.id'dir' and not inmath then
6628         new_dir = true
6629         dir = nil
6630     elseif item.id == node.id'math' then
6631         inmath = (item.subtype == 0)
6632     else
6633         dir = nil          -- Not a char
6634     end

```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```

6635     if dir == 'en' or dir == 'an' or dir == 'et' then
6636         if dir ~= 'et' then
6637             type_n = dir
6638         end
6639         first_n = first_n or item
6640         last_n = last_es or item
6641         last_es = nil
6642     elseif dir == 'es' and last_n then -- W3+W6
6643         last_es = item
6644     elseif dir == 'cs' then          -- it's right - do nothing
6645     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6646         if strong_lr == 'r' and type_n ~= '' then
6647             dir_mark(head, first_n, last_n, 'r')
6648         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6649             dir_mark(head, first_n, last_n, 'r')
6650             dir_mark(head, first_d, last_d, outer)

```

```

6651     first_d, last_d = nil, nil
6652     elseif strong_lr == 'l' and type_n ~= '' then
6653         last_d = last_n
6654     end
6655     type_n = ''
6656     first_n, last_n = nil, nil
6657 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:

```

6658     if dir == 'l' or dir == 'r' then
6659         if dir ~= outer then
6660             first_d = first_d or item
6661             last_d = item
6662         elseif first_d and dir ~= strong_lr then
6663             dir_mark(head, first_d, last_d, outer)
6664             first_d, last_d = nil, nil
6665         end
6666     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.

```

6667     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6668         item.char = characters[item.char] and
6669             characters[item.char].m or item.char
6670     elseif (dir or new_dir) and last_lr ~= item then
6671         local mir = outer .. strong_lr .. (dir or outer)
6672         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6673             for ch in node.traverse(node.next(last_lr)) do
6674                 if ch == item then break end
6675                 if ch.id == node.id'glyph' and characters[ch.char] then
6676                     ch.char = characters[ch.char].m or ch.char
6677                 end
6678             end
6679         end
6680     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).

```

6681     if dir == 'l' or dir == 'r' then
6682         last_lr = item
6683         strong = dir_real -- Don't search back - best save now
6684         strong_lr = (strong == 'l') and 'l' or 'r'
6685     elseif new_dir then
6686         last_lr = nil
6687     end
6688 end

```

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

```

6689     if last_lr and outer == 'r' then
6690         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6691             if characters[ch.char] then
6692                 ch.char = characters[ch.char].m or ch.char
6693             end

```

```

6694     end
6695 end
6696 if first_n then
6697     dir_mark(head, first_n, last_n, outer)
6698 end
6699 if first_d then
6700     dir_mark(head, first_d, last_d, outer)
6701 end

```

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

```

6702 return node.prev(head) or head
6703 end
6704 </basic-r>

```

And here the Lua code for bidi=basic:

```

6705 <(*basic>
6706 Babel = Babel or {}
6707
6708 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6709
6710 Babel.fontmap = Babel.fontmap or {}
6711 Babel.fontmap[0] = {}      -- l
6712 Babel.fontmap[1] = {}      -- r
6713 Babel.fontmap[2] = {}      -- al/an
6714
6715 Babel.bidi_enabled = true
6716 Babel.mirroring_enabled = true
6717
6718 require('babel-data-bidi.lua')
6719
6720 local characters = Babel.characters
6721 local ranges = Babel.ranges
6722
6723 local DIR = node.id('dir')
6724 local GLYPH = node.id('glyph')
6725
6726 local function insert_implicit(head, state, outer)
6727     local new_state = state
6728     if state.sim and state.eim and state.sim ~= state.eim then
6729         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6730         local d = node.new(DIR)
6731         d.dir = '+' .. dir
6732         node.insert_before(head, state.sim, d)
6733         local d = node.new(DIR)
6734         d.dir = '-' .. dir
6735         node.insert_after(head, state.eim, d)
6736     end
6737     new_state.sim, new_state.eim = nil, nil
6738     return head, new_state
6739 end
6740
6741 local function insert_numeric(head, state)
6742     local new
6743     local new_state = state
6744     if state.san and state.ean and state.san ~= state.ean then
6745         local d = node.new(DIR)
6746         d.dir = '+TLT'
6747         _, new = node.insert_before(head, state.san, d)

```

```

6748     if state.san == state.sim then state.sim = new end
6749     local d = node.new(DIR)
6750     d.dir = '-TLT'
6751     _, new = node.insert_after(head, state.ean, d)
6752     if state.ean == state.eim then state.eim = new end
6753 end
6754 new_state.san, new_state.ean = nil, nil
6755 return head, new_state
6756 end
6757
6758 -- TODO - \hbox with an explicit dir can lead to wrong results
6759 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6760 -- was s made to improve the situation, but the problem is the 3-dir
6761 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6762 -- well.
6763
6764 function Babel.bidi(head, ispar, hdir)
6765     local d -- d is used mainly for computations in a loop
6766     local prev_d = ''
6767     local new_d = false
6768
6769     local nodes = {}
6770     local outer_first = nil
6771     local inmath = false
6772
6773     local glue_d = nil
6774     local glue_i = nil
6775
6776     local has_en = false
6777     local first_et = nil
6778
6779     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6780
6781     local save_outer
6782     local temp = node.get_attribute(head, ATDIR)
6783     if temp then
6784         temp = temp % 3
6785         save_outer = (temp == 0 and 'l') or
6786                     (temp == 1 and 'r') or
6787                     (temp == 2 and 'al')
6788     elseif ispar then -- Or error? Shouldn't happen
6789         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6790     else -- Or error? Shouldn't happen
6791         save_outer = ('TRT' == hdir) and 'r' or 'l'
6792     end
6793     -- when the callback is called, we are just _after_ the box,
6794     -- and the textdir is that of the surrounding text
6795     -- if not ispar and hdir ~= tex.textdir then
6796     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6797     -- end
6798     local outer = save_outer
6799     local last = outer
6800     -- 'al' is only taken into account in the first, current loop
6801     if save_outer == 'al' then save_outer = 'r' end
6802
6803     local fontmap = Babel.fontmap
6804
6805     for item in node.traverse(head) do
6806

```

```

6807 -- In what follows, #node is the last (previous) node, because the
6808 -- current one is not added until we start processing the neutrals.
6809
6810 -- three cases: glyph, dir, otherwise
6811 if item.id == GLYPH
6812     or (item.id == 7 and item.subtype == 2) then
6813
6814     local d_font = nil
6815     local item_r
6816     if item.id == 7 and item.subtype == 2 then
6817         item_r = item.replace -- automatic discs have just 1 glyph
6818     else
6819         item_r = item
6820     end
6821     local chardata = characters[item_r.char]
6822     d = chardata and chardata.d or nil
6823     if not d or d == 'nsm' then
6824         for nn, et in ipairs(ranges) do
6825             if item_r.char < et[1] then
6826                 break
6827             elseif item_r.char <= et[2] then
6828                 if not d then d = et[3]
6829                 elseif d == 'nsm' then d_font = et[3]
6830             end
6831             break
6832         end
6833     end
6834 end
6835 d = d or 'l'
6836
6837 -- A short 'pause' in bidi for mapfont
6838 d_font = d_font or d
6839 d_font = (d_font == 'l' and 0) or
6840         (d_font == 'nsm' and 0) or
6841         (d_font == 'r' and 1) or
6842         (d_font == 'al' and 2) or
6843         (d_font == 'an' and 2) or nil
6844 if d_font and fontmap and fontmap[d_font][item_r.font] then
6845     item_r.font = fontmap[d_font][item_r.font]
6846 end
6847
6848 if new_d then
6849     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6850     if inmath then
6851         attr_d = 0
6852     else
6853         attr_d = node.get_attribute(item, ATDIR)
6854         attr_d = attr_d % 3
6855     end
6856     if attr_d == 1 then
6857         outer_first = 'r'
6858         last = 'r'
6859     elseif attr_d == 2 then
6860         outer_first = 'r'
6861         last = 'al'
6862     else
6863         outer_first = 'l'
6864         last = 'l'
6865     end

```

```

6866         outer = last
6867         has_en = false
6868         first_et = nil
6869         new_d = false
6870     end
6871
6872     if glue_d then
6873         if (d == 'l' and 'l' or 'r') ~= glue_d then
6874             table.insert(nodes, {glue_i, 'on', nil})
6875         end
6876         glue_d = nil
6877         glue_i = nil
6878     end
6879
6880     elseif item.id == DIR then
6881         d = nil
6882         new_d = true
6883
6884     elseif item.id == node.id'glue' and item.subtype == 13 then
6885         glue_d = d
6886         glue_i = item
6887         d = nil
6888
6889     elseif item.id == node.id'math' then
6890         inmath = (item.subtype == 0)
6891
6892     else
6893         d = nil
6894     end
6895
6896     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6897     if last == 'al' and d == 'en' then
6898         d = 'an'          -- W3
6899     elseif last == 'al' and (d == 'et' or d == 'es') then
6900         d = 'on'          -- W6
6901     end
6902
6903     -- EN + CS/ES + EN      -- W4
6904     if d == 'en' and #nodes >= 2 then
6905         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6906             and nodes[#nodes-1][2] == 'en' then
6907             nodes[#nodes][2] = 'en'
6908         end
6909     end
6910
6911     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6912     if d == 'an' and #nodes >= 2 then
6913         if (nodes[#nodes][2] == 'cs')
6914             and nodes[#nodes-1][2] == 'an' then
6915             nodes[#nodes][2] = 'an'
6916         end
6917     end
6918
6919     -- ET/EN                -- W5 + W7->1 / W6->on
6920     if d == 'et' then
6921         first_et = first_et or (#nodes + 1)
6922     elseif d == 'en' then
6923         has_en = true
6924         first_et = first_et or (#nodes + 1)

```

```

6925 elseif first_et then      -- d may be nil here !
6926   if has_en then
6927     if last == 'l' then
6928       temp = 'l'      -- W7
6929     else
6930       temp = 'en'     -- W5
6931     end
6932   else
6933     temp = 'on'       -- W6
6934   end
6935   for e = first_et, #nodes do
6936     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6937   end
6938   first_et = nil
6939   has_en = false
6940 end
6941
6942 -- Force mathdir in math if ON (currently works as expected only
6943 -- with 'l')
6944 if inmath and d == 'on' then
6945   d = ('TRT' == tex.mathdir) and 'r' or 'l'
6946 end
6947
6948 if d then
6949   if d == 'al' then
6950     d = 'r'
6951     last = 'al'
6952   elseif d == 'l' or d == 'r' then
6953     last = d
6954   end
6955   prev_d = d
6956   table.insert(nodes, {item, d, outer_first})
6957 end
6958
6959 outer_first = nil
6960
6961 end
6962
6963 -- TODO -- repeated here in case EN/ET is the last node. Find a
6964 -- better way of doing things:
6965 if first_et then      -- dir may be nil here !
6966   if has_en then
6967     if last == 'l' then
6968       temp = 'l'      -- W7
6969     else
6970       temp = 'en'     -- W5
6971     end
6972   else
6973     temp = 'on'       -- W6
6974   end
6975   for e = first_et, #nodes do
6976     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6977   end
6978 end
6979
6980 -- dummy node, to close things
6981 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6982
6983 ----- NEUTRAL -----

```

```

6984
6985 outer = save_outer
6986 last = outer
6987
6988 local first_on = nil
6989
6990 for q = 1, #nodes do
6991     local item
6992
6993     local outer_first = nodes[q][3]
6994     outer = outer_first or outer
6995     last = outer_first or last
6996
6997     local d = nodes[q][2]
6998     if d == 'an' or d == 'en' then d = 'r' end
6999     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7000
7001     if d == 'on' then
7002         first_on = first_on or q
7003     elseif first_on then
7004         if last == d then
7005             temp = d
7006         else
7007             temp = outer
7008         end
7009         for r = first_on, q - 1 do
7010             nodes[r][2] = temp
7011             item = nodes[r][1] -- MIRRORING
7012             if Babel.mirroring_enabled and item.id == GLYPH
7013                 and temp == 'r' and characters[item.char] then
7014                 local font_mode = font.fonts[item.font].properties.mode
7015                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7016                     item.char = characters[item.char].m or item.char
7017                 end
7018             end
7019         end
7020         first_on = nil
7021     end
7022
7023     if d == 'r' or d == 'l' then last = d end
7024 end
7025
7026 ----- IMPLICIT, REORDER -----
7027
7028 outer = save_outer
7029 last = outer
7030
7031 local state = {}
7032 state.has_r = false
7033
7034 for q = 1, #nodes do
7035
7036     local item = nodes[q][1]
7037
7038     outer = nodes[q][3] or outer
7039
7040     local d = nodes[q][2]
7041
7042     if d == 'nsm' then d = last end -- W1

```

```

7043   if d == 'en' then d = 'an' end
7044   local isdir = (d == 'r' or d == 'l')
7045
7046   if outer == 'l' and d == 'an' then
7047     state.san = state.san or item
7048     state.ean = item
7049   elseif state.san then
7050     head, state = insert_numeric(head, state)
7051   end
7052
7053   if outer == 'l' then
7054     if d == 'an' or d == 'r' then      -- im -> implicit
7055       if d == 'r' then state.has_r = true end
7056       state.sim = state.sim or item
7057       state.eim = item
7058     elseif d == 'l' and state.sim and state.has_r then
7059       head, state = insert_implicit(head, state, outer)
7060     elseif d == 'l' then
7061       state.sim, state.eim, state.has_r = nil, nil, false
7062     end
7063   else
7064     if d == 'an' or d == 'l' then
7065       if nodes[q][3] then -- nil except after an explicit dir
7066         state.sim = item -- so we move sim 'inside' the group
7067       else
7068         state.sim = state.sim or item
7069       end
7070       state.eim = item
7071     elseif d == 'r' and state.sim then
7072       head, state = insert_implicit(head, state, outer)
7073     elseif d == 'r' then
7074       state.sim, state.eim = nil, nil
7075     end
7076   end
7077
7078   if isdir then
7079     last = d      -- Don't search back - best save now
7080   elseif d == 'on' and state.san then
7081     state.san = state.san or item
7082     state.ean = item
7083   end
7084
7085 end
7086
7087 return node.prev(head) or head
7088 end
7089 </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.

```
7090 ⟨*nil⟩
7091 \ProvidesLanguage{nil}[\⟨date⟩ \⟨version⟩ Nil language]
7092 \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.

```
7093 \ifx\l@nil\undefined
7094   \newlanguage\l@nil
7095   \@namedef{bbl@hyphendata@the\l@nil}{{}}{}% Remove warning
7096   \let\bbl@elt\relax
7097   \edef\bbl@languages{% Add it to the list of languages
7098     \bbl@languages\bbl@elt{nil}{the\l@nil}{{}}
7099 \fi
```

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

```
7100 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7101 \let\captionnil\@empty
7102 \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.

```
7103 \ldf@finish{nil}
7104 ⟨/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 `locallyhyphen.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`.

```

7105 <*bplain | bplain>
7106 \catcode`\{=1 % left brace is begin-group character
7107 \catcode`\}=2 % right brace is end-group character
7108 \catcode`\#=6 % hash mark is macro parameter character

```

```

7109 \openin 0 hyphen.cfg
7110 \ifeof0
7111 \else
7112   \let\input

```

```

7113 \def\input #1 {%
7114   \let\input\@
7115   \a hyphen.cfg
7116   \let\@undefined
7117 }
7118 \fi
7119 </bplain | bplain>

```

```

7120 <bplain>\a plain.tex
7121 <blplain>\a lplain.tex

```

```

7122 \bplain\def\fmtname{babel-plain}
7123 \blplain\def\fmtname{babel-lplain}

```

16.2 Emulating some L^AT_EX features

```

7124 \langle \langle *Emulate LaTeX \rangle \rangle \equiv
7125 % == Code for plain ==
7126 \def \@empty {}
7127 \def \loadlocalcfg #1 {%
7128   \openin 0 #1.cfg
7129   \ifeof 0
7130     \closein 0
7131   \else
7132     \closein 0
7133     {\immediate\write16{*****}%}
7134     \immediate\write16{* Local config file #1.cfg used}%
7135     \immediate\write16{*}%
7136   }
7137   \input #1.cfg\relax
7138 \fi
7139 \@endofldef}

```

16.3 General tools

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

```

7140 \long\def\@firstofone#1{#1}
7141 \long\def\@firstoftwo#1#2{#1}
7142 \long\def\@secondoftwo#1#2{#2}
7143 \def\@nnil{\@nil}
7144 \def\@gobbletwo#1#2{}
7145 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7146 \def\@star@or@long#1{%
7147   \@ifstar
7148   {\let\l@ngrel@x\relax#1}%
7149   {\let\l@ngrel@x\long#1}}
7150 \let\l@ngrel@x\relax
7151 \def\@car#1#2\@nil{#1}
7152 \def\@cdr#1#2\@nil{#2}
7153 \let\@typeset@protect\relax
7154 \let\protected@edef\edef
7155 \long\def\@gobble#1{}
7156 \edef\@backslashchar{\expandafter\@gobble\string\}
7157 \def\strip@prefix#1>{}
7158 \def\g@addto@macro#1#2{%
7159   \toks@\expandafter{#1#2}%
7160   \xdef#1{\the\toks@}}
7161 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7162 \def\@nameuse#1{\csname #1\endcsname}
7163 \def\@ifundefined#1{%
7164   \expandafter\ifx\csname#1\endcsname\relax
7165     \expandafter\@firstoftwo
7166     \else
7167       \expandafter\@secondoftwo
7168     \fi}
7169 \def\@expandtwoargs#1#2#3{%
7170   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7171 \def\zap@space#1 #2{%
7172   #1%
7173   \ifx#2\@empty\else\expandafter\zap@space\fi
7174   #2}
7175 \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}`.

```

7176 \ifx\@preamblecmds\undefined
7177   \def\@preamblecmds{}
7178 \fi
7179 \def\@onlypreamble#1{%
7180   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7181     \@preamblecmds\do#1}}
7182 \@onlypreamble\onlypreamble

```

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

```

7183 \def\begindocument{%
7184   \@begindocumenthook
7185   \global\let\@begindocumenthook\undefined
7186   \def\do##1{\global\let##1\undefined}%
7187   \@preamblecmds
7188   \global\let\do\noexpand}
7189 \ifx\@begindocumenthook\undefined
7190   \def\@begindocumenthook{}

```

```

7191 \fi
7192 \@onlypreamble\@begindocumenthook
7193 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

We also have to mimick LATEX's \AtEndOfPackage. Our replacement macro is much simpler; it stores
its argument in \@endofldf.

7194 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7195 \@onlypreamble\AtEndOfPackage
7196 \def\@endofldf{}
7197 \@onlypreamble\@endofldf
7198 \let\bbl@afterlang\@empty
7199 \chardef\bbl@opt@hyphenmap\z@

```

L^AT_EX 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.

```

7200 \catcode`\&=\z@
7201 \ifx&\if@files\@undefined
7202   \expandafter\let\csname if@files\expandafter\endcsname
7203     \csname iffalse\endcsname
7204 \fi
7205 \catcode`\&=4

```

Mimick L^AT_EX's commands to define control sequences.

```

7206 \def\newcommand{\@star@or@long\new@command}
7207 \def\new@command#1{%
7208   \@testopt{\@newcommand#1}0}
7209 \def\@newcommand#1[#2]{%
7210   \@ifnextchar [{\@xargdef#1[#2]}%
7211     {\@argdef#1[#2]}}
7212 \long\def\@argdef#1[#2]#3{%
7213   \@yargdef#1\@ne{#2}{#3}}
7214 \long\def\@xargdef#1[#2][#3]#4{%
7215   \expandafter\def\expandafter#1\expandafter{%
7216     \expandafter\@protected@testopt\expandafter #1%
7217     \csname\string#1\expandafter\endcsname{#3}}%
7218   \expandafter\@yargdef \csname\string#1\endcsname
7219   \tw@{#2}{#4}}
7220 \long\def\@yargdef#1#2#3{%
7221   \@tempcnta#3\relax
7222   \advance \@tempcnta \@ne
7223   \let\@hash@\relax
7224   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7225   \@tempcntb #2%
7226   \@whilenum\@tempcntb <\@tempcnta
7227   \do{%
7228     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7229     \advance\@tempcntb \@ne}%
7230   \let\@hash@###
7231   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7232 \def\providecommand{\@star@or@long\provide@command}
7233 \def\provide@command#1{%
7234   \begingroup
7235   \escapechar\m@ne\xdef\@tempa{\string#1}%
7236   \endgroup
7237   \expandafter\ifundefined\@tempa
7238     {\def\reserved@a{\new@command#1}}%
7239     {\let\reserved@a\relax
7240      \def\reserved@a{\new@command\reserved@a}}%
7241   \reserved@a}%

```

```

7242 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7243 \def\declare@robustcommand#1{%
7244   \edef\reserved@a{\string#1}%
7245   \def\reserved@b{#1}%
7246   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7247   \edef#1{%
7248     \ifx\reserved@a\reserved@b
7249       \noexpand\x@protect
7250       \noexpand#1%
7251     \fi
7252     \noexpand\protect
7253     \expandafter\noexpand\csname
7254       \expandafter\@gobble\string#1 \endcsname
7255   }%
7256   \expandafter\new@command\csname
7257     \expandafter\@gobble\string#1 \endcsname
7258 }
7259 \def\x@protect#1{%
7260   \ifx\protect\@typeset@protect\else
7261     \@x@protect#1%
7262   \fi
7263 }
7264 \catcode`\&=\z@ % Trick to hide conditionals
7265 \def\@x@protect#1&fi##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`.

```

7266 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7267 \catcode`\&=4
7268 \ifx\in@\@undefined
7269   \def\in@#1#2{%
7270     \def\in@@##1#1##2##3\in@@{%
7271       \ifx\in@##2\in@false\else\in@true\fi}%
7272     \in@@#2#1\in@\in@@}
7273 \else
7274   \let\bbl@tempa\@empty
7275 \fi
7276 \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).

```

7277 \def\ifpackagewith#1#2#3#4{#3}

```

The \LaTeX macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```

7278 \def\ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\text{\LaTeX}_{2\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

7279 \ifx\@tempcnta\@undefined
7280   \csname newcount\endcsname\@tempcnta\relax
7281 \fi
7282 \ifx\@tempcntb\@undefined
7283   \csname newcount\endcsname\@tempcntb\relax
7284 \fi

```

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

```

7285 \ifx\bye\@undefined
7286   \advance\count10 by -2\relax
7287 \fi
7288 \ifx\@ifnextchar\@undefined
7289   \def\@ifnextchar#1#2#3{%
7290     \let\reserved@d=#1%
7291     \def\reserved@a{#2}\def\reserved@b{#3}%
7292     \futurelet\@let@token\@ifnch}
7293 \def\@ifnch{%
7294   \ifx\@let@token\@sptoken
7295     \let\reserved@c\@xifnch
7296   \else
7297     \ifx\@let@token\reserved@d
7298       \let\reserved@c\reserved@a
7299     \else
7300       \let\reserved@c\reserved@b
7301     \fi
7302   \fi
7303   \reserved@c}
7304 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
7305 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7306 \fi
7307 \def\@testopt#1#2{%
7308   \@ifnextchar[#{1}{#1[#2]}}
7309 \def\@protected@testopt#1{%
7310   \ifx\protect\@typeset@protect
7311     \expandafter\@testopt
7312   \else
7313     \@x@protect#1%
7314   \fi}
7315 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7316   #2\relax}\fi}
7317 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7318   \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

7319 \def\DeclareTextCommand{%
7320   \@dec@text@cmd\providecommand
7321 }
7322 \def\ProvideTextCommand{%
7323   \@dec@text@cmd\providecommand
7324 }
7325 \def\DeclareTextSymbol#1#2#3{%
7326   \@dec@text@cmd\chardef#1{#2}#3\relax
7327 }
7328 \def\@dec@text@cmd#1#2#3{%
7329   \expandafter\def\expandafter#2%
7330     \expandafter{%
7331       \csname#3-cmd\expandafter\endcsname
7332       \expandafter#2%
7333       \csname#3\string#2\endcsname
7334     }%
7335 %   \let\@ifdefinable\@rc@ifdefinable
7336   \expandafter#1\csname#3\string#2\endcsname

```

```

7337 }
7338 \def\@current@cmd#1{%
7339   \ifx\protect\@typeset@protect\else
7340     \noexpand#1\expandafter\@gobble
7341   \fi
7342 }
7343 \def\@changed@cmd#1#2{%
7344   \ifx\protect\@typeset@protect
7345     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7346       \expandafter\ifx\csname ?\string#1\endcsname\relax
7347         \expandafter\def\csname ?\string#1\endcsname{%
7348           \@changed@x@err{#1}%
7349         }%
7350       \fi
7351     \global\expandafter\let
7352       \csname\cf@encoding\string#1\expandafter\endcsname
7353       \csname ?\string#1\endcsname
7354   \fi
7355   \csname\cf@encoding\string#1%
7356     \expandafter\endcsname
7357 \else
7358   \noexpand#1%
7359 \fi
7360 }
7361 \def\@changed@x@err#1{%
7362   \errhelp{Your command will be ignored, type <return> to proceed}%
7363   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}%
7364 \def\DeclareTextCommandDefault#1{%
7365   \DeclareTextCommand#1?%
7366 }
7367 \def\ProvideTextCommandDefault#1{%
7368   \ProvideTextCommand#1?%
7369 }
7370 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7371 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7372 \def\DeclareTextAccent#1#2#3{%
7373   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7374 }
7375 \def\DeclareTextCompositeCommand#1#2#3#4{%
7376   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7377   \edef\reserved@b{\string##1}%
7378   \edef\reserved@c{%
7379     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7380   \ifx\reserved@b\reserved@c
7381     \expandafter\expandafter\expandafter\ifx
7382       \expandafter\@car\reserved@a\relax\relax\@nil
7383     \@text@composite
7384   \else
7385     \edef\reserved@b##1{%
7386       \def\expandafter\noexpand
7387         \csname#2\string#1\endcsname###1{%
7388           \noexpand\@text@composite
7389             \expandafter\noexpand\csname#2\string#1\endcsname
7390             ###1\noexpand\empty\noexpand\@text@composite
7391             {##1}%
7392         }%
7393     }%
7394     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7395   \fi

```

```

7396 \expandafter\def\csname\expandafter\string\csname
7397 #2\endcsname\string#1-\string#3\endcsname{#4}
7398 \else
7399 \errhelp{Your command will be ignored, type <return> to proceed}%
7400 \errmessage{\string\DeclareTextCompositeCommand\space used on
7401 inappropriate command \protect#1}
7402 \fi
7403 }
7404 \def\@text@composite#1#2#3\@text@composite{%
7405 \expandafter\@text@composite@x
7406 \csname\string#1-\string#2\endcsname
7407 }
7408 \def\@text@composite@x#1#2{%
7409 \ifx#1\relax
7410 #2%
7411 \else
7412 #1%
7413 \fi
7414 }
7415 %
7416 \def\@strip@args#1:#2-#3\@strip@args{#2}
7417 \def\DeclareTextComposite#1#2#3#4{%
7418 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7419 \bgroup
7420 \lccode\@=#4%
7421 \lowercase{%
7422 \egroup
7423 \reserved@a @%
7424 }%
7425 }
7426 %
7427 \def\UseTextSymbol#1#2{#2}
7428 \def\UseTextAccent#1#2#3{}
7429 \def\@use@text@encoding#1{}
7430 \def\DeclareTextSymbolDefault#1#2{%
7431 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7432 }
7433 \def\DeclareTextAccentDefault#1#2{%
7434 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7435 }
7436 \def\cf@encoding{OT1}

```

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

```

7437 \DeclareTextAccent{"}{OT1}{127}
7438 \DeclareTextAccent{'}{OT1}{19}
7439 \DeclareTextAccent{^}{OT1}{94}
7440 \DeclareTextAccent`}{OT1}{18}
7441 \DeclareTextAccent{~}{OT1}{126}

```

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

```

7442 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7443 \DeclareTextSymbol{\textquotedblright}{OT1}{93}
7444 \DeclareTextSymbol{\textquoteleft}{OT1}{96}
7445 \DeclareTextSymbol{\textquoteright}{OT1}{97}
7446 \DeclareTextSymbol{\i}{OT1}{16}
7447 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain $\text{T}_\text{E}\text{X}$ doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```

7448 \ifx\scriptsize\@undefined
7449   \let\scriptsize\sevenrm
7450 \fi
7451 % End of code for plain
7452 <</Emulate LaTeX>>

A proxy file:
7453 <plain>
7454 \input babel.def
7455 </plain>

```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \TeX 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, *\TeX , 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 \TeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).