

Babel

Version 3.51.2197
2020/11/20

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

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	9
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	16
1.12	The base option	18
1.13	ini files	19
1.14	Selecting fonts	27
1.15	Modifying a language	29
1.16	Creating a language	30
1.17	Digits and counters	33
1.18	Dates	35
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	37
1.21	Selection based on BCP 47 tags	39
1.22	Selecting scripts	40
1.23	Selecting directions	41
1.24	Language attributes	45
1.25	Hooks	45
1.26	Languages supported by babel with ldf files	46
1.27	Unicode character properties in luatex	48
1.28	Tweaking some features	48
1.29	Tips, workarounds, known issues and notes	48
1.30	Current and future work	50
1.31	Tentative and experimental code	50
2	Loading languages with language.dat	50
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	53
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	56
3.6	Support for extending macros	56
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	57
4	Changes	61
4.1	Changes in babel version 3.9	61
II	Source code	61

5	Identification and loading of required files	61
6	locale directory	62
7	Tools	62
7.1	Multiple languages	67
7.2	The Package File (\LaTeX , babel.sty)	67
7.3	base	69
7.4	Conditional loading of shorthands	71
7.5	Cross referencing macros	73
7.6	Marks	76
7.7	Preventing clashes with other packages	77
7.7.1	ifthen	77
7.7.2	varioref	77
7.7.3	hhline	78
7.7.4	hyperref	78
7.7.5	fancyhdr	78
7.8	Encoding and fonts	79
7.9	Basic bidi support	81
7.10	Local Language Configuration	86
8	The kernel of Babel (babel.def, common)	90
8.1	Tools	90
9	Multiple languages	91
9.1	Selecting the language	94
9.2	Errors	102
9.3	Hooks	105
9.4	Setting up language files	107
9.5	Shorthands	109
9.6	Language attributes	119
9.7	Support for saving macro definitions	121
9.8	Short tags	122
9.9	Hyphens	122
9.10	Multiencoding strings	124
9.11	Macros common to a number of languages	131
9.12	Making glyphs available	131
9.12.1	Quotation marks	131
9.12.2	Letters	132
9.12.3	Shorthands for quotation marks	133
9.12.4	Umlauts and tremas	134
9.13	Layout	136
9.14	Load engine specific macros	136
9.15	Creating and modifying languages	137
10	Adjusting the Babel bahavior	156
11	Loading hyphenation patterns	158
12	Font handling with fontspec	163

13	Hooks for XeTeX and LuaTeX	167
13.1	XeTeX	167
13.2	Layout	169
13.3	LuaTeX	171
13.4	Southeast Asian scripts	177
13.5	CJK line breaking	180
13.6	Automatic fonts and ids switching	181
13.7	Layout	192
13.8	Auto bidi with basic and basic-r	194
14	Data for CJK	205
15	The ‘nil’ language	205
16	Support for Plain T_EX (plain.def)	206
16.1	Not renaming hyphen.tex	206
16.2	Emulating some L _A T _E X features	207
16.3	General tools	207
16.4	Encoding related macros	211
17	Acknowledgements	214

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	9
Unknown language ‘LANG’	9
Argument of \language@active@arg” has an extra }	13
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	29
Package babel Info: The following fonts are not babel standard families	29

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 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 wiki](#). 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 (however, the package `inputenc` may be omitted with $\LaTeX \geq 2018-04-01$ if the encoding is UTF-8):

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}
% \usepackage[utf8]{inputenc} % Uncomment if LaTeX < 2018-04-01

\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 \LaTeX 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 follows. The main language is french, which is activated when the document begins. The package `inputenc` may be omitted with L^AT_EX ≥ 2018-04-01 if the encoding is UTF-8.

PDFTEX

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
\usepackage[utf8]{inputenc}  
  
\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 xetex and luatex, 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 1.21 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 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 tree-letter word is a valid name for a language (eg, `yi`). See section 1.21 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

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

```
\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

```
! 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 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 Plain.⁴

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` {*<language>*}

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.

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 `bidi` 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}`

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

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

⁴Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues have been fixed.

The environment `other language` does basically the same as `\selectlanguage`, except that language change is (mostly) local to the environment. Actually, there might be some non-local changes, as this environment is roughly equivalent to:

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

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

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

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

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

`\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 discouraged and `other language*` (the starred version) is preferred, as the former does not take into account possible changes in encodings of characters like, say, ‘ done by some languages (eg. italian, french, ukraineb). To set hyphenation exceptions, use `\babelhyphenation` (see below).

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with `fontenc`.⁵ 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` of `\dag`). With `ini` files (see below), captions are ensured by default.

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things, for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionaries and breaks can be inserted easily with "-", "=", etc.

⁵With it, encoded strings may not work as expected.

The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

NOTE Note 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) 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 it is deactivated with, eg, `\string`).

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\languageshorthands` `{\langle language \rangle}`

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).⁶ 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 german with

⁶Actually, 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.

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

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

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:⁷

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

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

Basque " ' ~

Breton : ; ? !

Catalan " ' `

Czech " -

Esperanto ^

Estonian " ~

French (all varieties) : ; ? !

Galician " . ' ~ < >

Greek ~

Hungarian `

Kurmanji ^

Latin " ^ =

Slovak " ^ ' -

Spanish " . < > ' ~

Turkish : ! =

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.⁸

⁷Thanks to Enrico Gregorio

⁸This declaration serves to nothing, but it is preserved for backward compatibility.

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

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

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

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

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

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

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

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

1.11 Package options

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

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

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

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

safe=	none ref bib
	Some \LaTeX macros are redefined so that using shorthands is safe. With <code>safe=bib</code> only <code>\nocite</code> , <code>\bibcite</code> and <code>\bibitem</code> are redefined. With <code>safe=ref</code> only <code>\newlabel</code> , <code>\ref</code> and <code>\pageref</code> are redefined (as well as a few macros from <code>varioref</code> and <code>ifthen</code>). With <code>safe=none</code> no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of New 3.34 , in $\epsilon\TeX$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).
math=	active normal
	Shorthands are mainly intended for text, not for math. By setting this option with the value <code>normal</code> they are deactivated in math mode (default is <code>active</code>) and things like <code>#{a'}</code> (a closing brace after a shorthand) are not a source of trouble anymore.
config=	$\langle file \rangle$
	Load $\langle file \rangle$.cfg instead of the default config file <code>bblopts.cfg</code> (the file is loaded even with <code>noconfigs</code>).
main=	$\langle language \rangle$
	Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.
headfoot=	$\langle language \rangle$
	By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
noconfigs	Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected .cfg file. However, if the key config is set, this file is loaded.
showlanguages	Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
nocase	New 3.9l Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code>) are ignored. Use only if there are incompatibilities with other packages.
silent	New 3.9l No warnings and no <i>infos</i> are written to the log file. ⁹
strings=	generic unicode encoded $\langle label \rangle$ $\langle font encoding \rangle$
	Selects the encoding of strings in languages supporting this feature. Predefined labels are <code>generic</code> (for traditional \TeX , LICR and ASCII strings), <code>unicode</code> (for engines like <code>xetex</code> and <code>luatex</code>) and <code>encoded</code> (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 <code>\MakeUpper</code> case and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
hyphenmap=	off first select other other*

⁹You can use alternatively the package `silence`.

New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.¹⁰ 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;¹¹
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.¹²

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. 1.23.

layout=

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.23.

1.12 The base option

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

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

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

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

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

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

¹⁰Turned off in plain.

¹¹Duplicated options count as several ones.

¹²Providing `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.

```

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

```

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

1.13 ini files

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

ini files are not meant only for babel, and they have been devised as a resource for other packages. To easy interoperability between T_EX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Language Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \ldots 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, and a recent version of fontspec/loaotfload is required. In xetex babel resorts to the bidi package, which seems to work.

Hebrew Niqqud marks seem to work in both engines, but cantillation marks are 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 fine tuning the font behavior is not always possible.

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{lṇ lṃ lṁ lṅ lṇ lṅ} % 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}	cs	Czech ^{ul}
agq	Aghem	cu	Church Slavic
ak	Akan	cu-Cyrs	Church Slavic
am	Amharic ^{ul}	cu-Glag	Church Slavic
ar	Arabic ^{ul}	cy	Welsh ^{ul}
ar-DZ	Arabic ^{ul}	da	Danish ^{ul}
ar-MA	Arabic ^{ul}	dav	Taita
ar-SY	Arabic ^{ul}	de-AT	German ^{ul}
as	Assamese	de-CH	German ^{ul}
asa	Asu	de	German ^{ul}
ast	Asturian ^{ul}	dje	Zarma
az-Cyrl	Azerbaijani	dsb	Lower Sorbian ^{ul}
az-Latn	Azerbaijani	dua	Duala
az	Azerbaijani ^{ul}	dyo	Jola-Fonyi
bas	Basaa	dz	Dzongkha
be	Belarusian ^{ul}	ebu	Embu
bem	Bemba	ee	Ewe
bez	Bena	el	Greek ^{ul}
bg	Bulgarian ^{ul}	el-polyton	Polytonic Greek ^{ul}
bm	Bambara	en-AU	English ^{ul}
bn	Bangla ^{ul}	en-CA	English ^{ul}
bo	Tibetan ^u	en-GB	English ^{ul}
brx	Bodo	en-NZ	English ^{ul}
bs-Cyrl	Bosnian	en-US	English ^{ul}
bs-Latn	Bosnian ^{ul}	en	English ^{ul}
bs	Bosnian ^{ul}	eo	Esperanto ^{ul}
ca	Catalan ^{ul}	es-MX	Spanish ^{ul}
ce	Chechen	es	Spanish ^{ul}
cgg	Chiga	et	Estonian ^{ul}
chr	Cherokee	eu	Basque ^{ul}
ckb	Central Kurdish	ewo	Ewondo
cop	Coptic	fa	Persian ^{ul}

ff	Fulah	ksb	Shambala
fi	Finnish ^{ul}	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French ^{ul}	ky	Kyrgyz
fr-BE	French ^{ul}	lag	Langi
fr-CA	French ^{ul}	lb	Luxembourgish
fr-CH	French ^{ul}	lg	Ganda
fr-LU	French ^{ul}	lkt	Lakota
fur	Friulian ^{ul}	ln	Lingala
fy	Western Frisian	lo	Lao ^{ul}
ga	Irish ^{ul}	lrc	Northern Luri
gd	Scottish Gaelic ^{ul}	lt	Lithuanian ^{ul}
gl	Galician ^{ul}	lu	Luba-Katanga
grc	Ancient Greek ^{ul}	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian ^{ul}
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa ^l	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ^l
hy	Armenian ^u	ms-SG	Malay ^l
ia	Interlingua ^{ul}	ms	Malay ^{ul}
id	Indonesian ^{ul}	mt	Maltese
ig	Igbo	mua	Mundang
ii	Sichuan Yi	my	Burmese
is	Icelandic ^{ul}	mzn	Mazanderani
it	Italian ^{ul}	naq	Nama
ja	Japanese	nb	Norwegian Bokmål ^{ul}
jgo	Ngomba	nd	North Ndebele
jmc	Machame	ne	Nepali
ka	Georgian ^{ul}	nl	Dutch ^{ul}
kab	Kabyle	nmg	Kwasio
kam	Kamba	nn	Norwegian Nynorsk ^{ul}
kde	Makonde	nnh	Ngiemboon
kea	Kabuverdianu	nus	Nuer
khq	Koyra Chiini	nyn	Nyankole
ki	Kikuyu	om	Oromo
kk	Kazakh	or	Odia
kkj	Kako	os	Ossetic
kl	Kalaallisut	pa-Arab	Punjabi
klj	Kalenjin	pa-Guru	Punjabi
km	Khmer	pa	Punjabi
kn	Kannada ^{ul}	pl	Polish ^{ul}
ko	Korean	pms	Piedmontese ^{ul}
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese ^{ul}

pt-PT	Portuguese ^{ul}	sr	Serbian ^{ul}
pt	Portuguese ^{ul}	sv	Swedish ^{ul}
qu	Quechua	sw	Swahili
rm	Romansh ^{ul}	ta	Tamil ^u
rn	Rundi	te	Telugu ^{ul}
ro	Romanian ^{ul}	teo	Teso
rof	Rombo	th	Thai ^{ul}
ru	Russian ^{ul}	ti	Tigrinya
rw	Kinyarwanda	tk	Turkmen ^{ul}
rwk	Rwa	to	Tongan
sa-Beng	Sanskrit	tr	Turkish ^{ul}
sa-Deva	Sanskrit	twq	Tasawaq
sa-Gujr	Sanskrit	tzm	Central Atlas Tamazight
sa-Knda	Sanskrit	ug	Uyghur
sa-Mlym	Sanskrit	uk	Ukrainian ^{ul}
sa-Telu	Sanskrit	ur	Urdu ^{ul}
sa	Sanskrit	uz-Arab	Uzbek
sah	Sakha	uz-Cyrl	Uzbek
saq	Samburu	uz-Latn	Uzbek
sbp	Sangu	uz	Uzbek
se	Northern Sami ^{ul}	vai-Latn	Vai
seh	Sena	vai-Vaii	Vai
ses	Koyraboro Senni	vai	Vai
sg	Sango	vi	Vietnamese ^{ul}
shi-Latn	Tachelhit	vun	Vunjo
shi-Tfng	Tachelhit	wae	Walser
shi	Tachelhit	xog	Soga
si	Sinhala	yav	Yangben
sk	Slovak ^{ul}	yi	Yiddish
sl	Slovenian ^{ul}	yo	Yoruba
smn	Inari Sami	yue	Cantonese
sn	Shona	zgh	Standard Moroccan Tamazight
so	Somali		
sq	Albanian ^{ul}	zh-Hans-HK	Chinese
sr-Cyrl-BA	Serbian ^{ul}	zh-Hans-MO	Chinese
sr-Cyrl-ME	Serbian ^{ul}	zh-Hans-SG	Chinese
sr-Cyrl-XK	Serbian ^{ul}	zh-Hans	Chinese
sr-Cyrl	Serbian ^{ul}	zh-Hant-HK	Chinese
sr-Latn-BA	Serbian ^{ul}	zh-Hant-MO	Chinese
sr-Latn-ME	Serbian ^{ul}	zh-Hant	Chinese
sr-Latn-XK	Serbian ^{ul}	zh	Chinese
sr-Latn	Serbian ^{ul}	zu	Zulu

In some contexts (currently `\babel font`) an `ini` file may be loaded by its name. Here is the list of the names currently supported. With these languages, `\babel font` 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 `\babel provide` with a valueless `import`.

aghem	american
akan	amharic
albanian	ancientgreek

arabic	chinese-simplified-hongkongsarchina
arabic-algeria	chinese-simplified-macausarchina
arabic-DZ	chinese-simplified-singapore
arabic-morocco	chinese-simplified
arabic-MA	chinese-traditional-hongkongsarchina
arabic-syria	chinese-traditional-macausarchina
arabic-SY	chinese-traditional
armenian	chinese
assamese	churchslavic
asturian	churchslavic-cyrs
asu	churchslavic-oldcyrillic ¹³
australian	churchsslavic-glag
austrian	churchsslavic-glagolitic
azerbaijani-cyrillic	cognian
azerbaijani-cyrl	cornish
azerbaijani-latin	croatian
azerbaijani-latn	czech
azerbaijani	danish
bafia	duala
bambara	dutch
basaa	dzongkha
basque	embu
belarusian	english-au
bemba	english-australia
ben	english-ca
bengali	english-canada
bodo	english-gb
bosnian-cyrillic	english-newzealand
bosnian-cyrl	english-nz
bosnian-latin	english-unitedkingdom
bosnian-latn	english-unitedstates
bosnian	english-us
brazilian	english
breton	esperanto
british	estonian
bulgarian	ewe
burmese	ewondo
canadian	faroes
cantonese	filipino
catalan	finnish
centralatlastamazight	french-be
centralkurdish	french-belgium
chechen	french-ca
cherokee	french-canada
chiga	french-ch
chinese-hans-hk	french-lu
chinese-hans-mo	french-luxembourg
chinese-hans-sg	french-switzerland
chinese-hans	french
chinese-hant-hk	friulian
chinese-hant-mo	fulah
chinese-hant	galician

¹³The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian

lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt

portuguese	slovak
punjabi-arab	slovene
punjabi-arabic	slovenian
punjabi-gurmukhi	soga
punjabi-guru	somali
punjabi	spanish-mexico
quechua	spanish-mx
romanian	spanish
romansh	standardmoroccantamazight
rombo	swahili
rundi	swedish
russian	swissgerman
rwa	tachelhit-latin
sakha	tachelhit-latn
samburu	tachelhit-tfng
samin	tachelhit-tifinagh
sango	tachelhit
sangu	taita
sanskrit-beng	tamil
sanskrit-bengali	tasawaq
sanskrit-deva	telugu
sanskrit-devanagari	teso
sanskrit-gujarati	thai
sanskrit-gujr	tibetan
sanskrit-kannada	tigrinya
sanskrit-knda	tongan
sanskrit-malayalam	turkish
sanskrit-mlym	turkmen
sanskrit-telu	ukenglish
sanskrit-telugu	ukrainian
sanskrit	upporsorbian
scottishgaelic	urdu
sena	usenglish
serbian-cyrillic-bosniaherzegovina	usorbian
serbian-cyrillic-kosovo	uyghur
serbian-cyrillic-montenegro	uzbek-arab
serbian-cyrillic	uzbek-arabic
serbian-cyrl-ba	uzbek-cyrillic
serbian-cyrl-me	uzbek-cyrl
serbian-cyrl-xk	uzbek-latin
serbian-cyrl	uzbek-latn
serbian-latin-bosniaherzegovina	uzbek
serbian-latin-kosovo	vai-latin
serbian-latin-montenegro	vai-latn
serbian-latin	vai-vai
serbian-latn-ba	vai-vaii
serbian-latn-me	vai
serbian-latn-xk	vietnam
serbian-latn	vietnamese
serbian	vunjo
shambala	walser
shona	welsh
sichuanyi	westernfrisian
sinhala	yangben

yiddish
yoruba

zarma
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=abcdefghijkl`. 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`.¹⁴

`\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.

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}
```

¹⁴See also the package `combofont` for a complementary approach.

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

first loads `danish.ldf`, and then redefines the captions for danish (as provided by the `ini` file) and prevents hyphenation. The rest of the language definitions are not touched.

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: \mylangchaptername not set. Please, define it
(babel)                after the language has been loaded (typically
(babel)                in the preamble) with something like:
(babel)                \renewcommand\mylangchaptername{..}
(babel)                Reported on input line 18.
```

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}
\renewcommand\arhinishchaptername{Chapitula}
\renewcommand\arhinishrefname{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 $\text{T}_{\text{E}}\text{X}$ sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with `\babelpatterns`, as for example:

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

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

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}
```

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

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

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

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

alph= $\langle counter-name \rangle$

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

Alph= $\langle counter-name \rangle$

Same for `\Alph`.

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

onchar= `ids | fonts`

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

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

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

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

intrapenalty= $\langle penalty \rangle$

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

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 \TeX code). This means the local digits have the correct bidirectional behavior (unlike `Numbers=Arabic` in `fontspec`, which is not recommended).

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

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

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

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

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

The styles are:

Ancient Greek `lower.ancient`, `upper.ancient`

Amharic `afar`, `agaw`, `ari`, `blin`, `dizi`, `gedeo`, `gumuz`, `hadiyya`, `harari`, `kaffa`, `kebena`, `kembata`, `konso`, `kunama`, `meen`, `oromo`, `saho`, `sidama`, `silti`, `tigre`, `wolaita`, `yemsa`

Arabic abjad, maghrebi.abjad
Belarusan, Bulgarian, Macedonian, Serbian lower, upper
Bengali alphabetic
Coptic epact, lower.letters
Hebrew letters (neither geresh nor gershayim yet)
Hindi alphabetic
Armenian lower.letter, upper.letter
Japanese hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Georgian letters
Greek lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
Khmer consonant
Korean consonant, syllable, 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. Çiley a Pêşîn 2019*, but with variant=iza fa it prints *31'ê Çiley a Pêşînê 2019*.

1.19 Accessing language info

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

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

\iflanguage $\{\langle language \rangle\}\{\langle true \rangle\}\{\langle false \rangle\}$

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 $\{\langle field \rangle\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\localeid

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

NOTE The `\localeid` is not the same as the `\language` identifier, which refers to a set of hyphenation patterns (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are 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 T_EX 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 T_EX 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 T_EX, - 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 L^AT_EX: (1) the character used is that set for the current font, while in L^AT_EX it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in L^AT_EX, but it can be changed to another value by redefining `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

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

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

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

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

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

New 3.9m *In `luatex` only,*¹⁵ adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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

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

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

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

\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. No rules are currently provided by default, 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 `([îú])`, the replacement could be `{1|îú|íú}`, which maps `î` to `í`, and `ú` to `ó`, so that the diaeresis is removed.

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

¹⁵With `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.

See the [babel wiki](#) for a more detailed description and some examples. It also describes an additional replacement type with the key `string`.

EXAMPLE 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 dictionaries into account). For example, you can use the `string` replacement to 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
\babelposthyphenation{russian-latin}{([sz])h} % Create rule
{
  { string = {1|sz|šž} },
  remove
}
```

In other words, it is a quite general tool. (A counterpart `\babelprehyphenation` is on the way.)

1.21 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.

1.22 Selecting scripts

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

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

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

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

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the

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.

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

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

1.23 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```

\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

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

```

\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}

```

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

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

graphics modifies the `picture` environment so that the whole figure is L but the text is R. It *does not* work with the standard `picture`, and `pict2e` is required if you want sloped lines (**With recent versions of \LaTeX , this feature has stopped working**). It attempts to do the same for `pgf/tikz`. Somewhat experimental. **New 3.32** .

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.24 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).

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.26 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}

```

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

Then you preprocess it with `devnag <file>`, which creates `<file>.tex`; you can then typeset the latter with \LaTeX .

1.27 Unicode character properties in luatex

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

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

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

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

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

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

1.28 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`. For example, you can set `\babeladjust{bidi.text=off}` if you are using an alternative algorithm or with large sections not requiring it. With `luahbtex` you may need `bidi.mirroring=off`. Use with care, because these options do not deactivate other related options (like `paragraph direction` with `bidi.text`).

1.29 Tips, workarounds, known issues and notes

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

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

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

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

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

(A recent version of inputenc is required.)

- 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 `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the .aux file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is 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.

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

1.30 Current and future work

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

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

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

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

1.31 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.

`\babelprehyphenation`

New 3.44 Note it is tentative, but the current behavior for glyphs should be correct. It is similar to `\babelposthyphenation`, but (as its name implies) applied before hyphenation. There are other differences: (1) the first argument is the locale instead the name of hyphenation patterns; (2) in the search patterns = has no special meaning (| is still reserved, but currently unused); (3) in the replacement, discretionaries are not accepted, only remove, , and string = ...

Currently it handles glyphs, not discretionaries or spaces (in particular, it will not catch the hyphen and you can't insert or remove spaces). Also, you are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg. Performance is still somewhat poor.

2 Loading languages with `language.dat`

\TeX and most engines based on it (pdf \TeX , xetex, ϵ - \TeX , the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, \LaTeX , Xe \LaTeX , pdf \LaTeX). babel provides a tool which has become standard in many distributions and based on a “configuration file” named `language.dat`. The exact way this file is used

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

depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

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

2.1 Format

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

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

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

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

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

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

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.

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

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

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

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

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 `\langle lang \rangle hyphenmins`, `\captions\langle lang \rangle`, `\date\langle lang \rangle`, `\extras\langle lang \rangle` and `\noextras\langle lang \rangle` (the last two may be left empty); where `\langle lang \rangle` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date\langle lang \rangle` but not `\captions\langle lang \rangle` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@\langle lang \rangle` to be a dialect of `\language0` when `\l@\langle lang \rangle` 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\langle lang \rangle` 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\langle lang \rangle`.

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁶
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

3.1 Guidelines for contributed languages

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

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

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

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

The following page provides a starting point for ldf files:

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

<https://github.com/latex3/babel/wiki/List-of-locale-templates>.

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

3.2 Basic macros

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

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

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define

²⁶But not removed, for backward compatibility.

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

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

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

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

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

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

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

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

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

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

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

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

`\ldf@quit` The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes resetting the category code of the `@`-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.

`\ldf@finish` The macro `\ldf@finish` does work needed at the end of each `.ldf` file. This includes resetting the category code of the `@`-sign, loading a local configuration file, and preparing the language to be activated at `\begin{document}` time.

`\loadlocalcfg` After processing a language definition file, \LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to `\captions<lang>` to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by `\ldf@finish`.

`\substitutefontfamily` (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct \LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

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

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

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

\bb{<language>}{<attrib>}{%
  \expandafter\addto\expandafter\extras<language>
  \expandafter{\extras<attrib><language>}%
  \let\captions<language>\captions<attrib><language>}

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

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

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

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

\StartBabelCommands*{<dialect>}{date}
\SetString\monthname{<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
```


<code>\savebox{\myeye}{\eye}%</code>	And direct usage
<code>\newsavebox{\myeye}</code>	
<code>\newcommand\myanchor{\anchor}%</code>	But OK inside command

3.4 Support for active characters

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

<code>\initiate@active@char</code>	The internal macro <code>\initiate@active@char</code> 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.
<code>\bbl@activate</code> <code>\bbl@deactivate</code>	The command <code>\bbl@activate</code> is used to change the way an active character expands. <code>\bbl@activate</code> ‘switches on’ the active behavior of the character. <code>\bbl@deactivate</code> lets the active character expand to its former (mostly) non-active self.
<code>\declare@shorthand</code>	The macro <code>\declare@shorthand</code> 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. <code>~</code> or <code>"a</code> ; and the code to be executed when the shorthand is encountered. (It does <i>not</i> raise an error if the shorthand character has not been “initiated”.)
<code>\bbl@add@special</code> <code>\bbl@remove@special</code>	The \TeX book states: “Plain \TeX includes a macro called <code>\dospecials</code> 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 <code>\dospecial</code> . \TeX adds another macro called <code>\@sanitize</code> representing the same character set, but without the curly braces. The macros <code>\bbl@add@special⟨char⟩</code> and <code>\bbl@remove@special⟨char⟩</code> add and remove the character <code>⟨char⟩</code> 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.

<code>\babel@save</code>	To save the current meaning of any control sequence, the macro <code>\babel@save</code> is provided. It takes one argument, <code>⟨cname⟩</code> , the control sequence for which the meaning has to be saved.
<code>\babel@savevariable</code>	A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the <code>\the</code> primitive is considered to be a variable. The macro takes one argument, the <code>⟨variable⟩</code> . The effect of the preceding macros is to append a piece of code to the current definition of <code>\originalTeX</code> . When <code>\originalTeX</code> 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

<code>\addto</code>	The macro <code>\addto{⟨control sequence⟩}{⟨\TeX code⟩}</code> can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or <code>\relax</code>). This macro can, for instance, be used in adding instructions to a macro like <code>\extrasenglish</code> . 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 <code>etoolbox</code> , by Philipp Lehman, consider using the tools provided by this package instead of <code>\addto</code> .
---------------------	--

²⁷This mechanism was introduced by Bernd Raichle.

3.7 Macros common to a number of languages

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

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiiname{M\"a\"rz}
\SetString\monthivname{April}
\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
\SetString\monthviiname{Juli}
\SetString\monthviiiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
```

²⁸In future releases further categories may be added.

```

\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 $\langle date \rangle \langle language \rangle$ exists).

\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

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

arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \LaTeX , we can set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

`\SetHyphenMap` $\{ \langle to\text{-}lower\text{-}macros \rangle \}$

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

1 <<version=3.51.2197>>

2 <<date=2020/11/20>>

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

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

```

3 <<(*Basic macros)>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8   {\def#1{#2}}%
9   {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@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 Because the code that is used in the handling of active characters may need to look ahead,
\bbl@afterfi we take extra care to ‘throw’ it over the \else and \fi parts of an \if-statement30. These
macros will break if another \if... \fi statement appears in one of the arguments and it
is not enclosed in braces.

27 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
28 \long\def\bbl@afterfi#1\fi{\fi#1}

\bbl@exp Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple
and readable. Here \> stands for \noexpand and \<. .> for \noexpand applied to a built
macro name (the latter does not define the macro if undefined to \relax, because it is
created locally). The result may be followed by extra arguments, if necessary.

29 \def\bbl@exp#1{%
30   \begingroup
31   \let\>\noexpand
32   \def\<##1>{\expandafter\>\noexpand\csname##1\endcsname}%
33   \edef\bbl@exp@aux{\endgroup#1}%
34   \bbl@exp@aux}

\bbl@trim The following piece of code is stolen (with some changes) from keyval, by David Carlisle. It
defines two macros: \bbl@trim and \bbl@trim@def. The first one strips the leading and
trailing spaces from the second argument and then applies the first argument (a macro,
\toks@ and the like). The second one, as its name suggests, defines the first argument as
the stripped second argument.

35 \def\bbl@tempa#1{%
36   \long\def\bbl@trim##1##2{%

```

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


```

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

```

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

```

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

```

```

81 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
82   \bbl@trim@def\bbl@forkv@a{#1}%
83   \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}

```

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

```

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

```

\bbl@replace

```

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

```

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

```

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

```

```

129 \endgroup
130 \bbl@tempc}} % empty or expand to set #1 with changes
131 \fi

```

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

```

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

```

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

```

154 \def\bbl@bsphack{%
155 \ifhmode
156 \hskip\z@skip
157 \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158 \else
159 \let\bbl@esphack\@empty
160 \fi}

```

Another hackish tool, to apply case changes inside a protected macros. It's based on the internal `\let's` made by `\MakeUppercase` and `\MakeLowercase` between things like `\oe` and `\OE`.

```

161 \def\bbl@cased{%
162 \ifx\oe\OE
163 \expandafter\in@\expandafter
164 {\expandafter\OE\expandafter}\expandafter{\oe}%
165 \ifin@
166 \bbl@afterelse\expandafter\MakeUppercase
167 \else
168 \bbl@afterfi\expandafter\MakeLowercase
169 \fi
170 \else
171 \expandafter\@firstofone
172 \fi}
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}[\<date> \<version>] The Babel package]
194 \@ifpackagewith{babel}{debug}
195   {\providecommand\bb1@trace[1]{\message{^^J[ #1 ]}}%

```

```

196 \let\bbl@debug\@firstofone}
197 {\providecommand\bbl@trace[1]{}%
198 \let\bbl@debug\@gobble}
199 <<Basic macros>>
200 % Temporarily repeat here the code for errors
201 \def\bbl@error#1#2{%
202   \begingroup
203     \def\{\MessageBreak}%
204     \PackageError{babel}{#1}{#2}%
205   \endgroup}
206 \def\bbl@warning#1{%
207   \begingroup
208     \def\{\MessageBreak}%
209     \PackageWarning{babel}{#1}%
210   \endgroup}
211 \def\bbl@infowarn#1{%
212   \begingroup
213     \def\{\MessageBreak}%
214     \GenericWarning
215       {(babel) \@spaces\@spaces\@spaces}%
216       {Package babel Info: #1}%
217   \endgroup}
218 \def\bbl@info#1{%
219   \begingroup
220     \def\{\MessageBreak}%
221     \PackageInfo{babel}{#1}%
222   \endgroup}
223 \def\bbl@nocaption{\protect\bbl@nocaption@i}
224 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
225   \global\@namedef{#2}{\textbf{?#1?}}%
226   \@nameuse{#2}%
227   \bbl@warning{%
228     \@backslashchar#2 not set. Please, define it\\%
229     after the language has been loaded (typically\\%
230     in the preamble) with something like:\\%
231     \string\renewcommand\@backslashchar#2{..}\\%
232     Reported}}
233 \def\bbl@tentative{\protect\bbl@tentative@i}
234 \def\bbl@tentative@i#1{%
235   \bbl@warning{%
236     Some functions for '#1' are tentative.\\%
237     They might not work as expected and their behavior\\%
238     may change in the future.\\%
239     Reported}}
240 \def\@nolanerr#1{%
241   \bbl@error
242     {You haven't defined the language #1\space yet.\\%
243     Perhaps you misspelled it or your installation\\%
244     is not complete}%
245     {Your command will be ignored, type <return> to proceed}}
246 \def\@nopatterns#1{%
247   \bbl@warning
248     {No hyphenation patterns were preloaded for\\%
249     the language '#1' into the format.\\%
250     Please, configure your TeX system to add them and\\%
251     rebuild the format. Now I will use the patterns\\%
252     preloaded for \bbl@nulllanguage\space instead}}
253   % End of errors
254 \@ifpackagewith{babel}{silent}

```

```

255 {\let\bbl@info\@gobble
256 \let\bbl@infowarn\@gobble
257 \let\bbl@warning\@gobble}
258 {}
259 %
260 \def\AfterBabelLanguage#1{%
261 \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.

```

262 \ifx\bbl@languages\@undefined\else
263 \begingroup
264 \catcode`\^^I=12
265 \@ifpackagewith{babel}{showlanguages}{%
266 \begingroup
267 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
268 \wlog{<*languages>}%
269 \bbl@languages
270 \wlog{</languages>}%
271 \endgroup}{%
272 \endgroup
273 \def\bbl@elt#1#2#3#4{%
274 \ifnum#2=\z@
275 \gdef\bbl@nulllanguage{#1}%
276 \def\bbl@elt##1##2##3##4{}}%
277 \fi}%
278 \bbl@languages
279 \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`.

```

280 \bbl@trace{Defining option 'base'}
281 \@ifpackagewith{babel}{base}{%
282 \let\bbl@onlyswitch\@empty
283 \let\bbl@provide@locale\relax
284 \input babel.def
285 \let\bbl@onlyswitch\@undefined
286 \ifx\directlua\@undefined
287 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
288 \else
289 \input luababel.def
290 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
291 \fi
292 \DeclareOption{base}{}%
293 \DeclareOption{showlanguages}{}%
294 \ProcessOptions
295 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
296 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
297 \global\let\@ifl@ter@@\@ifl@ter
298 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
299 \endinput}{}%
300 % \end{macrocode}

```

```

301%
302% \subsection{\texttt{key=value} options and other general option}
303%
304%     The following macros extract language modifiers, and only real
305%     package options are kept in the option list. Modifiers are saved
306%     and assigned to |\BabelModifiers| at |\bbl@load@language|; when
307%     no modifiers have been given, the former is |\relax|. How
308%     modifiers are handled are left to language styles; they can use
309%     |\in@|, loop them with |\@for| or load |keyval|, for example.
310%
311%     \begin{macrocode}
312\bbl@trace{key=value and another general options}
313\bbl@csarg\let\tempa\expandafter\csname opt@babel.sty\endcsname
314\def\bbl@tempb#1.#2{% Remove trailing dot
315    #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
316\def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
317    \ifx\@empty#2%
318        \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
319    \else
320        \in@{,provide,}{, #1,}%
321        \ifin@
322            \edef\bbl@tempc{%
323                \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
324        \else
325            \in@{=}{#1}%
326            \ifin@
327                \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
328            \else
329                \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
330                \bbl@csarg\edef{mod#1}{\bbl@tempb#2}%
331            \fi
332        \fi
333    \fi}
334\let\bbl@tempc\@empty
335\bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
336\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.

```

337\DeclareOption{KeepShorthandsActive}{}
338\DeclareOption{activeacute}{}
339\DeclareOption{activegrave}{}
340\DeclareOption{debug}{}
341\DeclareOption{noconfigs}{}
342\DeclareOption{showlanguages}{}
343\DeclareOption{silent}{}
344\DeclareOption{mono}{}
345\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
346\chardef\bbl@iniflag\z@
347\DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}    % main -> +1
348\DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}    % add = 2
349\DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
350% A separate option
351\let\bbl@autoload@options\@empty
352\DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
353% Don't use. Experimental. TODO.
354\newif\ifbbl@single

```

```

355 \DeclareOption{selectors=off}{\bbl@singletrue}
356 <<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.

```

357 \let\bbl@opt@shorthands\@nnil
358 \let\bbl@opt@config\@nnil
359 \let\bbl@opt@main\@nnil
360 \let\bbl@opt@headfoot\@nnil
361 \let\bbl@opt@layout\@nnil

```

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

```

362 \def\bbl@tempa#1=#2\bbl@tempa{%
363   \bbl@csarg\ifx{opt@#1}\@nnil
364     \bbl@csarg\edef{opt@#1}{#2}%
365   \else
366     \bbl@error
367     {Bad option `#1=#2'. Either you have misspelled the\\%
368      key or there is a previous setting of `#1'. Valid\\%
369      keys are, among others, `shorthands', `main', `bidi',\\%
370      `strings', `config', `headfoot', `safe', `math'.}%
371     {See the manual for further details.}
372   \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.

```

373 \let\bbl@language@opts\@empty
374 \DeclareOption*{%
375   \bbl@xin@{\string=}{\CurrentOption}%
376   \ifin@
377     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
378   \else
379     \bbl@add@list\bbl@language@opts{\CurrentOption}%
380   \fi}

```

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

```

381 \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=...`

```

382 \bbl@trace{Conditional loading of shorthands}
383 \def\bbl@sh@string#1{%
384   \ifx#1\@empty\else
385     \ifx#1t\string~%
386     \else\ifx#1c\string,%
387     \else\string#1%
388   \fi\fi

```



```

389 \expandafter\bb1@sh@string
390 \fi}
391 \ifx\bb1@opt@shorthands\@nnil
392 \def\bb1@ifshorthand#1#2#3{#2}%
393 \else\ifx\bb1@opt@shorthands\@empty
394 \def\bb1@ifshorthand#1#2#3{#3}%
395 \else

```

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

```

396 \def\bb1@ifshorthand#1{%
397 \bb1@xin@{\string#1}{\bb1@opt@shorthands}%
398 \ifin@
399 \expandafter\@firstoftwo
400 \else
401 \expandafter\@secondoftwo
402 \fi}

```

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

```

403 \edef\bb1@opt@shorthands{%
404 \expandafter\bb1@sh@string\bb1@opt@shorthands\@empty}%

```

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

```

405 \bb1@ifshorthand{'}%
406 {\PassOptionsToPackage{activeacute}{babel}}{}
407 \bb1@ifshorthand`}%
408 {\PassOptionsToPackage{activegrave}{babel}}{}
409 \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.

```

410 \ifx\bb1@opt@headfoot\@nnil\else
411 \g@addto@macro\@resetactivechars{%
412 \set@typeset@protect
413 \expandafter\select@language@x\expandafter{\bb1@opt@headfoot}%
414 \let\protect\noexpand}
415 \fi

```

For the option `safe` we use a different approach – `\bb1@opt@safe` says which macros are redefined (B for bibs and R for refs). By default, both are set.

```

416 \ifx\bb1@opt@safe\@undefined
417 \def\bb1@opt@safe{BR}
418 \fi
419 \ifx\bb1@opt@main\@nnil\else
420 \edef\bb1@language@opts{%
421 \ifx\bb1@language@opts\@empty\else\bb1@language@opts,\fi
422 \bb1@opt@main}
423 \fi

```

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

```

424 \bb1@trace{Defining IfBabelLayout}
425 \ifx\bb1@opt@layout\@nnil
426 \newcommand\IfBabelLayout[3]{#3}%
427 \else
428 \newcommand\IfBabelLayout[1]{%

```

```

429 \expandafter\in@{.#1.}{.\bbl@opt@layout.}%
430 \ifin@
431 \expandafter\@firstoftwo
432 \else
433 \expandafter\@secondoftwo
434 \fi}
435 \fi

```

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

```
436 \input babel.def
```

7.5 Cross referencing macros

The \LaTeX 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.

```

437 <<(*More package options)>> ≡
438 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
439 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
440 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
441 <</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.

```

442 \bbl@trace{Cross referencing macros}
443 \ifx\bbl@opt@safe\@empty\else
444 \def\@newl@bel#1#2#3{%
445   {\@safe@activetrue
446     \bbl@ifunset{#1@#2}%
447     \relax
448     {\gdef\@multiplelabels{%
449       \@latex@warning@no@line{There were multiply-defined labels}}%
450     \@latex@warning@no@line{Label `#2' multiply defined}}%
451   \global\@namedef{#1@#2}{#3}}

```

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

```

452 \CheckCommand*\@testdef[3]{%
453   \def\reserved@a{#3}%
454   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
455   \else
456     \@tempwattrue
457   \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.

```

458 \def\@testdef#1#2#3{% TODO. With @samestring?
459   \@safe@activetrue
460   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
461   \def\bbl@tempb{#3}%
462   \@safe@activesfalse
463   \ifx\bbl@tempa\relax
464   \else
465     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
466   \fi
467   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
468   \ifx\bbl@tempa\bbl@tempb
469   \else
470     \@tempswatrue
471   \fi}
472 \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.

```

473 \bbl@xin@{R}\bbl@opt@safe
474 \ifin@
475   \bbl@redefineroast\ref#1{%
476     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
477   \bbl@redefineroast\pageref#1{%
478     \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
479 \else
480   \let\org@ref\ref
481   \let\org@pageref\pageref
482 \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.

```

483 \bbl@xin@{B}\bbl@opt@safe
484 \ifin@
485   \bbl@redefine\@citex[#1]#2{%
486     \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
487     \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.

```

488 \AtBeginDocument{%
489   \@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.)

```

490   \def\@citex[#1][#2]#3{%
491     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
492     \org@@citex[#1][#2]{\@tempa}}%
493   }{}

```

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

```
494 \AtBeginDocument{%
495   \@ifpackageloaded{cite}{%
496     \def\@citex[#1]#2{%
497       \@safe@activetrue\org@citex[#1]{#2}\@safe@activesfalse}%
498     }{}}
```

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

```
499 \bbl@redefine\nocite#1{%
500   \@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.

```
501 \bbl@redefine\bibcite{%
502   \bbl@cite@choice
503   \bibcite}
```

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

```
504 \def\bbl@bibcite#1#2{%
505   \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.

```
506 \def\bbl@cite@choice{%
507   \global\let\bibcite\bbl@bibcite
508   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
509   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
510   \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.

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

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

```
512 \bbl@redefine\@bibitem#1{%
513   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
514 \else
515   \let\org@nocite\nocite
516   \let\org@citex\citex
517   \let\org@bibcite\bibcite
518   \let\org@bibitem\@bibitem
519 \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.

```

520 \bbl@trace{Marks}
521 \IfBabelLayout{sectioning}
522   {\ifx\bbl@opt@headfoot\@nnil
523     \g@addto@macro\@resetactivechars{%
524       \set@typeset@protect
525       \expandafter\select@language@x\expandafter{\bbl@main@language}%
526       \let\protect\noexpand
527       \ifcase\bbl@bidimode\else % Only with bidi. See also above
528         \edef\thepage{%
529           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
530       \fi}%
531   \fi}
532 {\ifbbl@single\else
533   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
534   \markright#1{%
535     \bbl@ifblank{#1}%
536     {\org@markright{}}%
537     {\toks@{#1}%
538       \bbl@exp{%
539         \org@markright{\protect\foreignlanguage{\language}%
540           \protect\bbl@restore@actives\the\toks@}}}%

```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two
`\@mkboth` 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.)

```

541   \ifx\@mkboth\markboth
542     \def\bbl@tempc{\let\@mkboth\markboth}
543   \else
544     \def\bbl@tempc{}
545   \fi
546   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
547   \markboth#1#2{%
548     \protected@edef\bbl@tempb##1{%
549       \protect\foreignlanguage
550       {\language}{\protect\bbl@restore@actives##1}}%
551     \bbl@ifblank{#1}%
552     {\toks@{}}%
553     {\toks@\expandafter{\bbl@tempb{#1}}}%
554     \bbl@ifblank{#2}%
555     {\@temptokena{}}%
556     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
557     \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
558     \bbl@tempc
559   \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.

```
560 \bbl@trace{Preventing clashes with other packages}
561 \bbl@xin@{R}\bbl@opt@safe
562 \ifin@
563   \AtBeginDocument{%
564     \ifpackageloaded{ifthen}{%
565       \bbl@redefine@long\ifthenelse#1#2#3{%
566         \let\bbl@temp@pref\pageref
567         \let\pageref\org@pageref
568         \let\bbl@temp@ref\ref
569         \let\ref\org@ref
570         \@safe@activestrue
571         \org@ifthenelse{#1}%
572           {\let\pageref\bbl@temp@pref
573            \let\ref\bbl@temp@ref
574            \@safe@activesfalse
575            #2}%
576           {\let\pageref\bbl@temp@pref
577            \let\ref\bbl@temp@ref
578            \@safe@activesfalse
579            #3}%
580       }%
581     }{}%
582   }
```

7.7.2 varioref

`\@@vpageref` When the package `varioref` is in use we need to modify its internal command `\@@vpageref`
`\vrefpagemum` in order to prevent problems when an active character ends up in the argument of `\vref`.
`\Ref` The same needs to happen for `\vrefpagemum`.

```
583 \AtBeginDocument{%
584   \ifpackageloaded{varioref}{%
585     \bbl@redefine\@@vpageref#1[#2]#3{%
586       \@safe@activestrue
587       \org@@@vpageref{#1}[#2]{#3}%
588       \@safe@activesfalse}%
589     \bbl@redefine\vrefpagemum#1#2{%
590       \@safe@activestrue
591       \org\vrefpagemum{#1}{#2}%
592     }
593   }
```

```
592 \safe@activesfalse}%
```

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

```
593 \expandafter\def\csname Ref \endcsname#1{%
594   \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
595   }{}%
596 }
597 \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.

```
598 \AtEndOfPackage{%
599   \AtBeginDocument{%
600     \ifpackageloaded{hhline}%
601       {\expandafter\ifx\csname normal@char\string\endcsname\relax
602         \else
603           \makeatletter
604           \def\@currname{hhline}\input{hhline.sty}\makeatother
605           \fi}%
606     {}}}
```

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.

```
607 % \AtBeginDocument{%
608 %   \ifx\pdfstringdefDisableCommands\@undefined\else
609 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
610 %   \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.

```
611 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
612   \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 .

```
613 \def\substitutefontfamily#1#2#3{%
```

```

614 \lowercase{\immediate\openout15=#1#2.fd\relax}%
615 \immediate\write15{%
616   \string\ProvidesFile{#1#2.fd}%
617   [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
618   \space generated font description file]^J
619   \string\DeclareFontFamily{#1}{#2}{^^J
620   \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
621   \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
622   \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
623   \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
624   \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
625   \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
626   \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
627   \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
628   }%
629   \closeout15
630 }
631 \@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`

```

632 \bbl@trace{Encoding and fonts}
633 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
634 \newcommand\BabelNonText{TS1,T3,TS3}
635 \let\org@TeX\TeX
636 \let\org@LaTeX\LaTeX
637 \let\ensureascii\@firstofone
638 \AtBeginDocument{%
639   \in@false
640   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
641     \ifin@
642       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
643     \fi}%
644   \ifin@ % if a text non-ascii has been loaded
645     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
646     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
647     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
648     \def\bbl@temp#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
649     \def\bbl@tempc#1ENC.DEF#2\@@{%
650       \ifx\@empty#2\else
651         \bbl@ifunset{T#1}%
652         {}%
653         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
654       \ifin@
655         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
656         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
657       \else
658         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
659       \fi}%

```



```

660     \fi}%
661     \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
662     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
663     \ifin@else
664         \edef\ensureascii#1{%
665             \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
666     \fi
667 \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.

```

668 \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.

```

669 \AtBeginDocument{%
670     \ifpackageloaded{fontspec}%
671     {\xdef\latinencoding{%
672         \ifx\UTFencname\@undefined
673             EU\ifcase\bbl@engine\or2\or1\fi
674         \else
675             \UTFencname
676         \fi}}%
677     {\gdef\latinencoding{OT1}%
678         \ifx\cf@encoding\bbl@t@one
679             \xdef\latinencoding{\bbl@t@one}%
680         \else
681             \ifx\@fontenc@load@list\@undefined
682                 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
683             \else
684                 \def\@elt#1{,#1,}%
685                 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
686                 \let\@elt\relax
687                 \bbl@xin@{,T1,}\bbl@tempa
688                 \ifin@
689                     \xdef\latinencoding{\bbl@t@one}%
690                 \fi
691             \fi
692         \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.

```

693 \DeclareRobustCommand{\latintext}{%
694     \fontencoding{\latinencoding}\selectfont
695     \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.

```

696 \ifx\@undefined\DeclareTextFontCommand
697     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}

```

```

698 \else
699   \DeclareTextFontCommand{\textlatin}{\latintext}
700 \fi

```

7.9 Basic bidi support

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

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This `babel` module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I've also looked at `ARABI` (by Youssef Jabri), which is compatible with `babel`.

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

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

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

```

701 \ifodd\bbl@engine
702   \def\bbl@activate@preotf{%
703     \let\bbl@activate@preotf\relax % only once
704     \directlua{
705       Babel = Babel or {}
706       %
707       function Babel.pre_otfload_v(head)
708         if Babel.numbers and Babel.digits_mapped then
709           head = Babel.numbers(head)
710         end
711         if Babel.bidi_enabled then
712           head = Babel.bidi(head, false, dir)
713         end
714         return head
715       end
716       %
717       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
718         if Babel.numbers and Babel.digits_mapped then
719           head = Babel.numbers(head)
720         end
721         if Babel.bidi_enabled then
722           head = Babel.bidi(head, false, dir)
723         end
724         return head

```

```

725     end
726     %
727     luatexbase.add_to_callback('pre_linebreak_filter',
728         Babel.pre_otfload_v,
729         'Babel.pre_otfload_v',
730         luatexbase.priority_in_callback('pre_linebreak_filter',
731             'luaotfload.node_processor') or nil)
732     %
733     luatexbase.add_to_callback('hpack_filter',
734         Babel.pre_otfload_h,
735         'Babel.pre_otfload_h',
736         luatexbase.priority_in_callback('hpack_filter',
737             'luaotfload.node_processor') or nil)
738     }}
739 \fi

```

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

```

740 \bbl@trace{Loading basic (internal) bidi support}
741 \ifodd\bbl@engine
742   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
743     \let\bbl@beforeforeign\leavevmode
744     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
745     \RequirePackage{luatexbase}
746     \bbl@activate@preotf
747     \directlua{
748       require('babel-data-bidi.lua')
749       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
750         require('babel-bidi-basic.lua')
751       \or
752         require('babel-bidi-basic-r.lua')
753     \fi}
754     % TODO - to locale_props, not as separate attribute
755     \newattribute\bbl@attr@dir
756     % TODO. I don't like it, hackish:
757     \bbl@exp{\output{\bodydir\pagedir\the\output}}
758     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
759   \fi\fi
760 \else
761   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
762     \bbl@error
763     {The bidi method `basic' is available only in\\%
764       luatex. I'll continue with `bidi=default', so\\%
765       expect wrong results}%
766     {See the manual for further details.}%
767     \let\bbl@beforeforeign\leavevmode
768     \AtEndOfPackage{%
769       \EnableBabelHook{babel-bidi}%
770       \bbl@xebidipar}
771   \fi\fi
772   \def\bbl@loadxebidi#1{%
773     \ifx\RTLfootnotetext\@undefined
774       \AtEndOfPackage{%
775         \EnableBabelHook{babel-bidi}%
776         \ifx\fontspec\@undefined
777           \bbl@loadfontspec % bidi needs fontspec
778         \fi
779         \usepackage#1{bidi}}%
780   \fi}

```

```

781 \ifnum\bbl@bidimode>200
782   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
783     \bbl@tentative{bidi=bidi}
784     \bbl@loadxebidi{}
785   \or
786     \bbl@loadxebidi{[rldocument]}
787   \or
788     \bbl@loadxebidi{}
789   \fi
790 \fi
791 \fi
792 \ifnum\bbl@bidimode=\@ne
793   \let\bbl@beforeforeign\leavevmode
794   \ifodd\bbl@engine
795     \newattribute\bbl@attr@dir
796     \bbl@exp{\output{\bodydir\pagedir\the\output}}%
797   \fi
798   \AtEndOfPackage{%
799     \EnableBabelHook{babel-bidi}%
800   \ifodd\bbl@engine\else
801     \bbl@xebidipar
802   \fi}
803 \fi

```

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

```

804 \bbl@trace{Macros to switch the text direction}
805 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
806 \def\bbl@rscripts{% TODO. Base on codes ??
807   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
808   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
809   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
810   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
811   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
812   Old South Arabian,%
813 \def\bbl@provide@dirs#1{%
814   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
815   \ifin@
816     \global\bbl@csarg\chardef{wdir@#1}\@ne
817     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
818   \ifin@
819     \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
820   \fi
821   \else
822     \global\bbl@csarg\chardef{wdir@#1}\z@
823   \fi
824   \ifodd\bbl@engine
825     \bbl@csarg\ifcase{wdir@#1}%
826       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
827     \or
828       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
829     \or
830       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
831     \fi
832   \fi}
833 \def\bbl@switchdir{%
834   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
835   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
836   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}

```

```

837 \def\bbl@setdirs#1{% TODO - math
838 \ifcase\bbl@select@type % TODO - strictly, not the right test
839 \bbl@bodydir{#1}%
840 \bbl@pardir{#1}%
841 \fi
842 \bbl@texkdir{#1}}
843 % TODO. Only if \bbl@bidimode > 0?:
844 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
845 \DisableBabelHook{babel-bidi}

```

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

```

846 \ifodd\bbl@engine % luatex=1
847 \chardef\bbl@thetexkdir\z@
848 \chardef\bbl@thepardir\z@
849 \def\bbl@getluadir#1{%
850 \directlua{
851   if tex.#1dir == 'TLT' then
852     tex.sprint('0')
853   elseif tex.#1dir == 'TRT' then
854     tex.sprint('1')
855   end}}
856 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texkdir.. 3=0 lr/1 rl
857 \ifcase#3\relax
858 \ifcase\bbl@getluadir{#1}\relax\else
859 #2 TLT\relax
860 \fi
861 \else
862 \ifcase\bbl@getluadir{#1}\relax
863 #2 TRT\relax
864 \fi
865 \fi}
866 \def\bbl@texkdir#1{%
867 \bbl@setluadir{tex}\texkdir{#1}%
868 \chardef\bbl@thetexkdir#1\relax
869 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
870 \def\bbl@pardir#1{%
871 \bbl@setluadir{par}\pardir{#1}%
872 \chardef\bbl@thepardir#1\relax}
873 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
874 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
875 \def\bbl@dirparastext{\pardir\the\texkdir\relax}% %%%
876 % Sadly, we have to deal with boxes in math with basic.
877 % Activated every math with the package option bidi=:
878 \def\bbl@mathboxdir{%
879 \ifcase\bbl@thetexkdir\relax
880 \everyhbox{\texkdir TLT\relax}%
881 \else
882 \everyhbox{\texkdir TRT\relax}%
883 \fi}
884 \frozen@everymath\expandafter{%
885 \expandafter\bbl@mathboxdir\the\frozen@everymath}
886 \frozen@everydisplay\expandafter{%
887 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
888 \else % pdftex=0, xetex=2
889 \newcount\bbl@dirlevel
890 \chardef\bbl@thetexkdir\z@
891 \chardef\bbl@thepardir\z@
892 \def\bbl@texkdir#1{%
893 \ifcase#1\relax

```

```

894     \chardef\bbl@thetextdir\z@
895     \bbl@textdir@i\beginL\endL
896   \else
897     \chardef\bbl@thetextdir\@ne
898     \bbl@textdir@i\beginR\endR
899   \fi}
900 \def\bbl@textdir@i#1#2{%
901   \ifhmode
902     \ifnum\currentgrouplevel>\z@
903       \ifnum\currentgrouplevel=\bbl@dirlevel
904         \bbl@error{Multiple bidi settings inside a group}%
905         {I'll insert a new group, but expect wrong results.}%
906         \bgroup\aftergroup#2\aftergroup\egroup
907       \else
908         \ifcase\currentgrouptype\or % 0 bottom
909           \aftergroup#2% 1 simple {}
910         \or
911           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
912         \or
913           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
914         \or\or\or % vbox vtop align
915         \or
916           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
917         \or\or\or\or\or\or % output math disc insert vcent mathchoice
918         \or
919           \aftergroup#2% 14 \begingroup
920         \else
921           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
922         \fi
923       \fi
924       \bbl@dirlevel\currentgrouplevel
925     \fi
926   #1%
927 \fi}
928 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
929 \let\bbl@bodydir\@gobble
930 \let\bbl@pagedir\@gobble
931 \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).

```

932 \def\bbl@xebidipar{%
933   \let\bbl@xebidipar\relax
934   \TeXeTstate\@ne
935   \def\bbl@xeverypar{%
936     \ifcase\bbl@thepardir
937       \ifcase\bbl@thetextdir\else\beginR\fi
938     \else
939       {\setbox\z@\lastbox\beginR\box\z@}%
940     \fi}%
941   \let\bbl@severypar\everypar
942   \newtoks\everypar
943   \everypar=\bbl@severypar
944   \bbl@severypar{\bbl@xeverypar\the\everypar}}
945 \ifnum\bbl@bidimode>200
946   \let\bbl@textdir@i\@gobbletwo
947   \let\bbl@xebidipar\@empty
948   \AddBabelHook{bidi}{foreign}{%

```

```

949 \def\bbl@tempa{\def\BabelText####1}%
950 \ifcase\bbl@thetextdir
951 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
952 \else
953 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
954 \fi}
955 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
956 \fi
957 \fi

```

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

```

958 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
959 \AtBeginDocument{%
960 \ifx\pdfstringdefDisableCommands\@undefined\else
961 \ifx\pdfstringdefDisableCommands\relax\else
962 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
963 \fi
964 \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`.

```

965 \bbl@trace{Local Language Configuration}
966 \ifx\loadlocalcfg\@undefined
967 \@ifpackagewith{babel}{noconfigs}%
968 {\let\loadlocalcfg\@gobble}%
969 {\def\loadlocalcfg#1{%
970 \InputIfFileExists{#1.cfg}%
971 {\typeout{*****^J%
972 * Local config file #1.cfg used^^J%
973 *}}}%
974 \@empty}}
975 \fi

```

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

```

976 \ifx\@unexpandable@protect\@undefined
977 \def\@unexpandable@protect{\noexpand\protect\noexpand}
978 \long\def\protected@write#1#2#3{%
979 \begingroup
980 \let\thepage\relax
981 #2%
982 \let\protect\@unexpandable@protect
983 \edef\reserved@a{\write#1{#3}}%
984 \reserved@a
985 \endgroup
986 \if@nobreak\ifvmode\nobreak\fi\fi}
987 \fi
988 %
989 % \subsection{Language options}
990 %
991 % Languages are loaded when processing the corresponding option

```

```

992% \textit{except} if a |main| language has been set. In such a
993% case, it is not loaded until all options has been processed.
994% The following macro inputs the ldf file and does some additional
995% checks (|\input| works, too, but possible errors are not caught).
996%
997%     \begin{macrocode}
998 \bbl@trace{Language options}
999 \let\bbl@afterlang\relax
1000 \let\BabelModifiers\relax
1001 \let\bbl@loaded@empty
1002 \def\bbl@load@language#1{%
1003   \InputIfFileExists{#1.ldf}%
1004   {\edef\bbl@loaded{\CurrentOption
1005     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1006     \expandafter\let\expandafter\bbl@afterlang
1007       \csname\CurrentOption.ldf-h@k\endcsname
1008     \expandafter\let\expandafter\BabelModifiers
1009       \csname bbl@mod@\CurrentOption\endcsname}%
1010   {\bbl@error{%
1011     Unknown option '\CurrentOption'. Either you misspelled it\\%
1012     or the language definition file \CurrentOption.ldf was not found}}%
1013   Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1014   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1015   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.

```

1016 \def\bbl@try@load@lang#1#2#3{%
1017   \IfFileExists{\CurrentOption.ldf}%
1018   {\bbl@load@language{\CurrentOption}}%
1019   {#1\bbl@load@language{#2}#3}}
1020 \DeclareOption{hebrew}{%
1021   \input{rlbabel.def}%
1022   \bbl@load@language{hebrew}}
1023 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1024 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1025 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1026 \DeclareOption{polutonikogreek}{%
1027   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1028 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1029 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1030 \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.

```

1031 \ifx\bbl@opt@config\@nnil
1032   \@ifpackagewith{babel}{noconfigs}{}%
1033   {\InputIfFileExists{bblopts.cfg}%
1034     {\typeout{*****^^J%
1035       * Local config file bblopts.cfg used^^J%
1036       *}}%
1037     {}}%
1038 \else
1039   \InputIfFileExists{\bbl@opt@config.cfg}%

```



```

1040 {\typeout{*****^J%
1041         * Local config file \bbl@opt@config.cfg used^^J%
1042         *}}%
1043 {\bbl@error{%
1044     Local config file '\bbl@opt@config.cfg' not found}{%
1045     Perhaps you misspelled it.}}%
1046 \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.

```

1047 \let\bbl@tempc\relax
1048 \bbl@foreach\bbl@language@opts{%
1049     \ifcase\bbl@iniflag
1050         \bbl@ifunset{ds@#1}%
1051         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1052         {}%
1053     \or
1054         \@gobble % case 2 same as 1
1055     \or
1056         \bbl@ifunset{ds@#1}%
1057         {\IfFileExists{#1.ldf}{}%
1058          {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}}}%
1059          {}%
1060         \bbl@ifunset{ds@#1}%
1061         {\def\bbl@tempc{#1}%
1062          \DeclareOption{#1}{%
1063              \ifnum\bbl@iniflag>\@ne
1064                  \bbl@ldfinit
1065                  \babelprovide[import]{#1}%
1066                  \bbl@afterldf{}}%
1067              \else
1068                  \bbl@load@language{#1}%
1069              \fi}}%
1070         {}%
1071     \or
1072         \def\bbl@tempc{#1}%
1073         \bbl@ifunset{ds@#1}%
1074         {\DeclareOption{#1}{%
1075             \bbl@ldfinit
1076             \babelprovide[import]{#1}%
1077             \bbl@afterldf{}}}%
1078         {}%
1079     \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.

```

1080 \let\bbl@tempb\@nnil
1081 \bbl@foreach\@classoptionslist{%
1082     \bbl@ifunset{ds@#1}%
1083     {\IfFileExists{#1.ldf}{}%
1084      {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}}}%
1085      {}%
1086     \bbl@ifunset{ds@#1}%
1087     {\def\bbl@tempb{#1}%

```

```

1088 \DeclareOption{#1}{%
1089 \ifnum\bbl@iniflag>\@ne
1090 \bbl@ldfinit
1091 \babelprovide[import]{#1}%
1092 \bbl@afterldf{}}%
1093 \else
1094 \bbl@load@language{#1}%
1095 \fi}}%
1096 {}

```

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

```

1097 \ifnum\bbl@iniflag=\z@ \else
1098 \ifx\bbl@opt@main\@nnil
1099 \ifx\bbl@tempc\relax
1100 \let\bbl@opt@main\bbl@tempb
1101 \else
1102 \let\bbl@opt@main\bbl@tempc
1103 \fi
1104 \fi
1105 \fi
1106 \ifx\bbl@opt@main\@nnil \else
1107 \expandafter
1108 \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1109 \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1110 \fi

```

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

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

```

1111 \def\AfterBabelLanguage#1{%
1112 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1113 \DeclareOption*{}
1114 \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`.

```

1115 \bbl@trace{Option 'main'}
1116 \ifx\bbl@opt@main\@nnil
1117 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1118 \let\bbl@tempc\@empty
1119 \bbl@for\bbl@tempb\bbl@tempa{%
1120 \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
1121 \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1122 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1123 \expandafter\bbl@tempa\bbl@loaded,\@nnil
1124 \ifx\bbl@tempb\bbl@tempc \else
1125 \bbl@warning{%
1126 Last declared language option is '\bbl@tempc',\%
1127 but the last processed one was '\bbl@tempb'.\%
1128 The main language cannot be set as both a global\%
1129 and a package option. Use 'main=\bbl@tempc' as\%
1130 option. Reported}%
1131 \fi
1132 \else

```

```

1133 \ifodd\bbl@iniflag % case 1,3
1134 \bbl@ldfinit
1135 \let\CurrentOption\bbl@opt@main
1136 \bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}
1137 \bbl@afterldf{}%
1138 \else % case 0,2
1139 \chardef\bbl@iniflag\z@ % Force ldf
1140 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1141 \ExecuteOptions{\bbl@opt@main}
1142 \DeclareOption*{}%
1143 \ProcessOptions*
1144 \fi
1145 \fi
1146 \def\AfterBabelLanguage{%
1147 \bbl@error
1148 {Too late for \string\AfterBabelLanguage}%
1149 {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.

1150 \ifx\bbl@main@language\@undefined
1151 \bbl@info{%
1152 You haven't specified a language. I'll use 'nil'\%
1153 as the main language. Reported}
1154 \bbl@load@language{nil}
1155 \fi
1156 \</package>
1157 \<core>

```

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

The kernel of the babel system is currently stored in babel.def. The file babel.def contains most of the code. The file hyphen.cfg is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns. Because plain T_EX users might want to use some of the features of the babel system too, care has to be taken that plain T_EX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain T_EX and L^AT_EX, some of it is for the L^AT_EX case only. Plain formats based on etex (etex, xetex, luatex) don't load hyphen.cfg but etex.src, which follows a different naming convention, so we need to define the babel names. It presumes language.def exists and it is the same file used when formats were created.

8.1 Tools

```

1158 \ifx\ldf@quit\@undefined\else
1159 \endinput\fi % Same line!
1160 \<<Make sure ProvidesFile is defined>>
1161 \ProvidesFile{babel.def}[\<<date>> \<<version>>] Babel common definitions]

```

The file babel.def expects some definitions made in the L^AT_EX 2_ε style file. So, In L^AT_EX 2.09 and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and 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).

```

1162 \ifx\AtBeginDocument\@undefined % TODO. change test.
1163 <<Emulate LaTeX>>
1164 \def\language{english}%
1165 \let\bbl@opt@shorthands\@nnil
1166 \def\bbl@ifshorthand#1#2#3#2}%
1167 \let\bbl@language@opts\@empty
1168 \ifx\babeloptionstrings\@undefined
1169   \let\bbl@opt@strings\@nnil
1170 \else
1171   \let\bbl@opt@strings\babeloptionstrings
1172 \fi
1173 \def\BabelStringsDefault{generic}
1174 \def\bbl@tempa{normal}
1175 \ifx\babeloptionmath\bbl@tempa
1176   \def\bbl@mathnormal{\noexpand\textormath}
1177 \fi
1178 \def\AfterBabelLanguage#1#2{}
1179 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1180 \let\bbl@afterlang\relax
1181 \def\bbl@opt@safe{BR}
1182 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1183 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1184 \expandafter\newif\csname ifbbl@single\endcsname
1185 \chardef\bbl@bidimode\z@
1186 \fi

```

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

```

1187 \ifx\bbl@trace\@undefined
1188   \let\LdfInit\endinput
1189   \def\ProvidesLanguage#1{\endinput}
1190 \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.

```

1191 <<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.

```

1192 \def\bbl@version{\<version>}
1193 \def\bbl@date{\<date>}
1194 \def\adddialect#1#2{%
1195   \global\chardef#1#2\relax
1196   \bbl@usehooks{adddialect}{#1}{#2}}%
1197   \begingroup
1198     \count@#1\relax
1199     \def\bbl@elt##1##2##3##4{%
1200       \ifnum\count@=##2\relax
1201         \bbl@info{\string#1 = using hyphenrules for ##1\%
1202           (\string\language\the\count@)}%
1203         \def\bbl@elt####1####2####3####4{}%
1204       \fi}%
1205   \bbl@cs{languages}%

```

```
1206 \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.

```
1207 \def\bbl@fixname#1{%
1208   \begingroup
1209   \def\bbl@tempe{l@}%
1210   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1211   \bbl@tempd
1212   {\lowercase\expandafter{\bbl@tempd}%
1213    {\uppercase\expandafter{\bbl@tempd}%
1214     \@empty
1215     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1216      \uppercase\expandafter{\bbl@tempd}}}%
1217    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1218     \lowercase\expandafter{\bbl@tempd}}}%
1219   \@empty
1220   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1221   \bbl@tempd
1222   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1223 \def\bbl@iflanguage#1{%
1224   \@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.

```
1225 \def\bbl@bcpcase#1#2#3#4\@#5{%
1226   \ifx\@empty#3%
1227     \uppercase{\def#5{#1#2}}%
1228   \else
1229     \uppercase{\def#5{#1}}%
1230     \lowercase{\edef#5{#5#2#3#4}}%
1231   \fi}
1232 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
1233   \let\bbl@bcp\relax
1234   \lowercase{\def\bbl@tempa{#1}}%
1235   \ifx\@empty#2%
1236     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1237   \else\ifx\@empty#3%
1238     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
1239     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1240     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1241     {}%
1242   \ifx\bbl@bcp\relax
1243     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1244   \fi
1245   \else
1246     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
1247     \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
```

```

1248 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1249 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1250 {}}%
1251 \ifx\bbl@bcp\relax
1252 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1253 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1254 {}}%
1255 \fi
1256 \ifx\bbl@bcp\relax
1257 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1258 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1259 {}}%
1260 \fi
1261 \ifx\bbl@bcp\relax
1262 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}}}%
1263 \fi
1264 \fi\fi}
1265 \let\bbl@initoload\relax
1266 \def\bbl@provide@locale{%
1267 \ifx\babelprovide\undefined
1268 \bbl@error{For a language to be defined on the fly 'base'\\%
1269 is not enough, and the whole package must be\\%
1270 loaded. Either delete the 'base' option or\\%
1271 request the languages explicitly}%
1272 {See the manual for further details.}%
1273 \fi
1274 % TODO. Option to search if loaded, with \LocaleForEach
1275 \let\bbl@auxname\language % Still necessary. TODO
1276 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1277 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1278 \ifbbl@bcpallowed
1279 \expandafter\ifx\csname date\language\endcsname\relax
1280 \expandafter
1281 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1282 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1283 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1284 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1285 \expandafter\ifx\csname date\language\endcsname\relax
1286 \let\bbl@initoload\bbl@bcp
1287 \bbl@exp{\\\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1288 \let\bbl@initoload\relax
1289 \fi
1290 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1291 \fi
1292 \fi
1293 \fi
1294 \expandafter\ifx\csname date\language\endcsname\relax
1295 \IfFileExists{babel-\language.tex}%
1296 {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\language}}}%
1297 {}}%
1298 \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.

```

1299 \def\iflanguage#1{%

```

```

1300 \bbl@iflanguage{#1}{%
1301   \ifnum\csname l@#1\endcsname=\language
1302     \expandafter\@firstoftwo
1303   \else
1304     \expandafter\@secondoftwo
1305   \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.

```

1306 \let\bbl@select@type\z@
1307 \edef\selectlanguage{%
1308   \noexpand\protect
1309   \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`.

```
1310 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1311 \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.

```
1312 \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:

```

1313 \def\bbl@push@language{%
1314   \ifx\language\@undefined\else
1315     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1316   \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`.

```
1317 \def\bbl@pop@lang#1+#2\@@{%
1318   \edef\language{#1}%
1319   \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).

```
1320 \let\bbl@ifrestoring\@secondoftwo
1321 \def\bbl@pop@language{%
1322   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1323   \let\bbl@ifrestoring\@firstoftwo
1324   \expandafter\bbl@set@language\expandafter{\language}%
1325   \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).

```
1326 \chardef\localeid\z@
1327 \def\bbl@id@last{0} % No real need for a new counter
1328 \def\bbl@id@assign{%
1329   \bbl@ifunset{bbl@id@\language}%
1330   {\count@bbl@id@last\relax
1331    \advance\count@bbl@id@last\ne
1332    \bbl@csarg\chardef{id@\language}\count@
1333    \edef\bbl@id@last{\the\count@}%
1334    \ifcase\bbl@engine\or
1335      \directlua{
1336        Babel = Babel or {}
1337        Babel.locale_props = Babel.locale_props or {}
1338        Babel.locale_props[\bbl@id@last] = {}
1339        Babel.locale_props[\bbl@id@last].name = '\language'
1340      }%
1341    \fi}%
1342   {}}%
1343   \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of `\selectlanguage`.

```
1344 \expandafter\def\csname selectlanguage \endcsname#1{%
1345   \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\tw\fi
1346   \bbl@push@language
1347   \aftergroup\bbl@pop@language
1348   \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.

```

1349 \def\BabelContentsFiles{toc,lof,lot}
1350 \def\bbl@set@language#1{% from selectlanguage, pop@
1351 % The old buggy way. Preserved for compatibility.
1352 \edef\language{%
1353   \ifnum\escapechar=\expandafter`\string#1\@empty
1354   \else\string#1\@empty\fi}%
1355 \ifcat\relax\noexpand#1%
1356   \expandafter\ifx\csname date\language\endcsname\relax
1357   \edef\language{#1}%
1358   \let\locale\language
1359 \else
1360   \bbl@info{Using '\string\language' instead of 'language' is\\%
1361             deprecated. If what you want is to use a\\%
1362             macro containing the actual locale, make\\%
1363             sure it does not not match any language.\\%
1364             Reported}%
1365   I'll\\%
1366   try to fix '\string\locale', but I cannot promise\\%
1367   anything. Reported}%
1368   \ifx\scantokens\undefined
1369   \def\locale{??}%
1370 \else
1371   \scantokens\expandafter{\expandafter
1372     \def\expandafter\locale\expandafter{\language}}%
1373 \fi
1374 \fi
1375 \else
1376   \def\locale{#1}% This one has the correct catcodes
1377 \fi
1378 \select@language{\language}%
1379 % write to aux
1380 \expandafter\ifx\csname date\language\endcsname\relax\else
1381   \if@files
1382     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1383       \protected@write\@auxout{\string\babel@aux{\bbl@auxname}}}%
1384     \fi
1385     \bbl@usehooks{write}%
1386   \fi
1387 \fi}
1388 %
1389 \newif\ifbbl@bcpallowed
1390 \bbl@bcpallowedfalse
1391 \def\select@language#1{% from set@, babel@aux
1392 % set hymap
1393 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
1394 % set name
1395 \edef\language{#1}%
1396 \bbl@fixname\language
1397 % TODO. name@map must be here?
1398 \bbl@provide@locale
1399 \bbl@iflanguage\language{%
1400   \expandafter\ifx\csname date\language\endcsname\relax
1401   \bbl@error
1402     {Unknown language '\language'. Either you have\\%
1403     misspelled its name, it has not been installed,\\%

```

```

1404         or you requested it in a previous run. Fix its name,\%
1405         install it or just rerun the file, respectively. In\%
1406         some cases, you may need to remove the aux file}%
1407         {You may proceed, but expect wrong results}%
1408     \else
1409         % set type
1410         \let\bbl@select@type\z@
1411         \expandafter\bbl@switch\expandafter{\language}%
1412     \fi}}
1413 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1414     \select@language{#1}%
1415     \bbl@foreach\BabelContentsFiles{%
1416         \@writefile{##1}{\babel@toc{#1}{#2}}}% %% TODO - ok in plain?
1417 \def\babel@toc#1#2{%
1418     \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.

```

1419 \newif\ifbbl@usedatagroup
1420 \def\bbl@switch#1{% from select@, foreign@
1421     % make sure there is info for the language if so requested
1422     \bbl@ensureinfo{#1}%
1423     % restore
1424     \originalTeX
1425     \expandafter\def\expandafter\originalTeX\expandafter{%
1426         \csname noextras#1\endcsname
1427         \let\originalTeX\@empty
1428         \babel@beginsave}%
1429     \bbl@usehooks{afterreset}{}%
1430     \languageshorthands{none}%
1431     % set the locale id
1432     \bbl@id@assign
1433     % switch captions, date
1434     % No text is supposed to be added here, so we remove any
1435     % spurious spaces.
1436     \bbl@bsphack
1437     \ifcase\bbl@select@type
1438         \csname captions#1\endcsname\relax
1439         \csname date#1\endcsname\relax
1440     \else
1441         \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1442         \ifin@
1443             \csname captions#1\endcsname\relax
1444         \fi
1445         \bbl@xin@{,date,}{, \bbl@select@opts,}%

```

```

1446     \ifin@ % if \foreign... within \<lang>date
1447     \csname date#1\endcsname\relax
1448     \fi
1449     \fi
1450     \bbl@esphack
1451     % switch extras
1452     \bbl@usehooks{beforeextras}{}%
1453     \csname extras#1\endcsname\relax
1454     \bbl@usehooks{afterextras}{}%
1455     % > babel-ensure
1456     % > babel-sh-<short>
1457     % > babel-bidi
1458     % > babel-fontspec
1459     % hyphenation - case mapping
1460     \ifcase\bbl@opt@hyphenmap\or
1461         \def\BabelLower##1##2{\lccode##1=##2\relax}%
1462         \ifnum\bbl@hymapsel>4\else
1463             \csname\language @bbl@hyphenmap\endcsname
1464             \fi
1465         \chardef\bbl@opt@hyphenmap\z@
1466     \else
1467         \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1468             \csname\language @bbl@hyphenmap\endcsname
1469             \fi
1470     \fi
1471     \global\let\bbl@hymapsel@cclv
1472     % hyphenation - select patterns
1473     \bbl@patterns{#1}%
1474     % hyphenation - allow stretching with babelnohyphens
1475     \ifnum\language=\l@babelnohyphens
1476         \babel@savevariable\emergencystretch
1477         \emergencystretch\maxdimen
1478         \babel@savevariable\hbadness
1479         \hbadness\@M
1480     \fi
1481     % hyphenation - mins
1482     \babel@savevariable\lefthyphenmin
1483     \babel@savevariable\righthyphenmin
1484     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1485         \set@hyphenmins\tw@\thr@@\relax
1486     \else
1487         \expandafter\expandafter\expandafter\set@hyphenmins
1488         \csname #1hyphenmins\endcsname\relax
1489     \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.

```

1490 \long\def\otherlanguage#1{%
1491     \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\thr@@\fi
1492     \csname selectlanguage \endcsname{#1}%
1493     \ignorespaces}

```

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

```

1494 \long\def\endotherlanguage{%
1495   \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`.

```

1496 \expandafter\def\csname otherlanguage*\endcsname{%
1497   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1498 \def\bbl@otherlanguage@s[#1]#2{%
1499   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1500   \def\bbl@select@opts{#1}%
1501   \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”.

```

1502 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras⟨lang⟩` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`. `\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op. (3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in `vmode` and then selects the language (which in turn sets the paragraph direction). (3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises. In other words, at the beginning of a paragraph `\foreignlanguage` enters into `hmode` with the surrounding `lang`, and with `\foreignlanguage*` with the new `lang`.

```

1503 \providecommand\bbl@beforeforeign{}
1504 \edef\foreignlanguage{%
1505   \noexpand\protect
1506   \expandafter\noexpand\csname foreignlanguage \endcsname}
1507 \expandafter\def\csname foreignlanguage \endcsname{%
1508   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1509 \providecommand\bbl@foreign@x[3][]{%
1510   \begingroup
1511     \def\bbl@select@opts{#1}%
1512     \let\BabelText\@firstofone
1513     \bbl@beforeforeign
1514     \foreign@language{#2}%
1515     \bbl@usehooks{foreign}{}%
1516     \BabelText{#3}% Now in horizontal mode!
1517   \endgroup}
1518 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par

```

```

1519 \begingroup
1520 {\par}%
1521 \let\BabelText\@firstofone
1522 \foreign@language{#1}%
1523 \bbl@usehooks{foreign*}{}%
1524 \bbl@dirparastext
1525 \BabelText{#2}% Still in vertical mode!
1526 {\par}%
1527 \endgroup}

```

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

```

1528 \def\foreign@language#1{%
1529 % set name
1530 \edef\language#1}%
1531 \ifbbl@usedategroup
1532 \bbl@add\bbl@select@opts{,date,}%
1533 \bbl@usedategroupfalse
1534 \fi
1535 \bbl@fixname\language
1536 % TODO. name@map here?
1537 \bbl@provide@locale
1538 \bbl@iflanguage\language{%
1539 \expandafter\ifx\csname date\language\endcsname\relax
1540 \bbl@warning % TODO - why a warning, not an error?
1541 {Unknown language `#1'. Either you have\\%
1542 misspelled its name, it has not been installed,\\%
1543 or you requested it in a previous run. Fix its name,\\%
1544 install it or just rerun the file, respectively. In\\%
1545 some cases, you may need to remove the aux file.\\%
1546 I'll proceed, but expect wrong results.\\%
1547 Reported}%
1548 \fi
1549 % set type
1550 \let\bbl@select@type\@ne
1551 \expandafter\bbl@switch\expandafter{\language}}

```

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

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

```

1552 \let\bbl@hyphlist\@empty
1553 \let\bbl@hyphenation@\relax
1554 \let\bbl@pttnlist\@empty
1555 \let\bbl@patterns@\relax
1556 \let\bbl@hymapsel=\@cclv
1557 \def\bbl@patterns#1{%
1558 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1559 \csname l@#1\endcsname
1560 \edef\bbl@tempa{#1}%
1561 \else

```

```

1562 \csname l@#1:\f@encoding\endcsname
1563 \edef\bbl@tempa{#1:\f@encoding}%
1564 \fi
1565 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1566 % > luatex
1567 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1568 \begingroup
1569 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1570 \ifin@else
1571 \expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1572 \hyphenation{%
1573 \bbl@hyphenation@
1574 \@ifundefined{bbl@hyphenation@#1}%
1575 \@empty
1576 {\space\csname bbl@hyphenation@#1\endcsname}}%
1577 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1578 \fi
1579 \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 other language*.

```

1580 \def\hyphenrules#1{%
1581 \edef\bbl@tempf{#1}%
1582 \bbl@fixname\bbl@tempf
1583 \bbl@iflanguage\bbl@tempf{%
1584 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1585 \ifx\languageshorthands\undefined\else
1586 \languageshorthands{none}%
1587 \fi
1588 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1589 \set@hyphenmins\tw@\thr@@\relax
1590 \else
1591 \expandafter\expandafter\expandafter\set@hyphenmins
1592 \csname\bbl@tempf hyphenmins\endcsname\relax
1593 \fi}}
1594 \let\endhyphenrules\@empty

```

\providehyphenmins The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\(lang)hyphenmins` is already defined this command has no effect.

```

1595 \def\providehyphenmins#1#2{%
1596 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1597 \@namedef{#1hyphenmins}{#2}%
1598 \fi}

```

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

```

1599 \def\set@hyphenmins#1#2{%
1600 \lefthyphenmin#1\relax
1601 \righthyphenmin#2\relax}

```

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

```

1602 \ifx\ProvidesFile\undefined
1603   \def\ProvidesLanguage#1[#2 #3 #4]{%
1604     \wlog{Language: #1 #4 #3 <#2>}%
1605   }
1606 \else
1607   \def\ProvidesLanguage#1{%
1608     \begingroup
1609       \catcode`\ 10 %
1610       \@makeother\/%
1611       \@ifnextchar[%]
1612         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1613   \def\@provideslanguage#1[#2]{%
1614     \wlog{Language: #1 #2}%
1615     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1616     \endgroup}
1617 \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`.

```

1618 \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.

```

1619 \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’:

```

1620 \providecommand\setlocale{%
1621   \bbl@error
1622   {Not yet available}%
1623   {Find an armchair, sit down and wait}}
1624 \let\uselocale\setlocale
1625 \let\locale\setlocale
1626 \let\selectlocale\setlocale
1627 \let\localename\setlocale
1628 \let\textlocale\setlocale
1629 \let\textlanguage\setlocale
1630 \let\languagetext\setlocale

```

9.2 Errors

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

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

```

1631 \edef\bbl@nulllanguage{\string\language=0}
1632 \ifx\PackageError\undefined % TODO. Move to Plain

```

```

1633 \def\bbl@error#1#2{%
1634 \begingroup
1635 \newlinechar=`^^J
1636 \def\{^^J(babel) }%
1637 \errhelp{#2}\errmessage{\#1}%
1638 \endgroup}
1639 \def\bbl@warning#1{%
1640 \begingroup
1641 \newlinechar=`^^J
1642 \def\{^^J(babel) }%
1643 \message{\#1}%
1644 \endgroup}
1645 \let\bbl@infowarn\bbl@warning
1646 \def\bbl@info#1{%
1647 \begingroup
1648 \newlinechar=`^^J
1649 \def\{^^J}%
1650 \wlog{#1}%
1651 \endgroup}
1652 \fi
1653 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1654 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1655 \global\@namedef{#2}{\textbf{?#1?}}%
1656 \@nameuse{#2}%
1657 \bbl@warning{%
1658 \@backslashchar#2 not set. Please, define it\\%
1659 after the language has been loaded (typically\\%
1660 in the preamble) with something like:\\%
1661 \string\renewcommand\@backslashchar#2{..}\\%
1662 Reported}}
1663 \def\bbl@tentative{\protect\bbl@tentative@i}
1664 \def\bbl@tentative@i#1{%
1665 \bbl@warning{%
1666 Some functions for '#1' are tentative.\\%
1667 They might not work as expected and their behavior\\%
1668 could change in the future.\\%
1669 Reported}}
1670 \def\@nolanerr#1{%
1671 \bbl@error
1672 {You haven't defined the language #1\space yet.\\%
1673 Perhaps you misspelled it or your installation\\%
1674 is not complete}%
1675 {Your command will be ignored, type <return> to proceed}}
1676 \def\@nopatterns#1{%
1677 \bbl@warning
1678 {No hyphenation patterns were preloaded for\\%
1679 the language '#1' into the format.\\%
1680 Please, configure your TeX system to add them and\\%
1681 rebuild the format. Now I will use the patterns\\%
1682 preloaded for \bbl@nulllanguage\space instead}}
1683 \let\bbl@usehooks\@gobbletwo
1684 \ifx\bbl@onlyswitch\@empty\endinput\fi
1685 % Here ended switch.def

Here ended switch.def.

1686 \ifx\directlua\@undefined\else
1687 \ifx\bbl@luapatterns\@undefined
1688 \input luabel.def
1689 \fi

```



```

1690 \fi
1691 <<Basic macros>>
1692 \bbl@trace{Compatibility with language.def}
1693 \ifx\bbl@languages@undefined
1694   \ifx\directlua@undefined
1695     \openin1 = language.def % TODO. Remove hardcoded number
1696     \ifeof1
1697       \closein1
1698       \message{I couldn't find the file language.def}
1699     \else
1700       \closein1
1701       \begingroup
1702         \def\addlanguage#1#2#3#4#5{%
1703           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1704             \global\expandafter\let\csname l@#1\expandafter\endcsname
1705               \csname lang@#1\endcsname
1706           \fi}%
1707         \def\uselanguage#1{%
1708           \input language.def
1709         \endgroup
1710       \fi
1711     \fi
1712   \chardef\l@english\z@
1713 \fi

```

\addto It takes two arguments, a *<control sequence>* and TeX-code to be added to the *<control sequence>*.

If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

1714 \def\addto#1#2{%
1715   \ifx#1\@undefined
1716     \def#1{#2}%
1717   \else
1718     \ifx#1\relax
1719       \def#1{#2}%
1720     \else
1721       {\toks@\expandafter{#1#2}%
1722        \xdef#1{\the\toks@}}%
1723     \fi
1724   \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?

```

1725 \def\bbl@withactive#1#2{%
1726   \begingroup
1727   \lccode`~=`#2\relax
1728   \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 L^AT_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`.

```

1729 \def\bbl@redefine#1{%
1730   \edef\bbl@tempa{\bbl@stripslash#1}%
1731   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1732   \expandafter\def\csname\bbl@tempa\endcsname}
1733 \@onlypreamble\bbl@redefine

```

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

```

1734 \def\bbl@redefine@long#1{%
1735   \edef\bbl@tempa{\bbl@stripslash#1}%
1736   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1737   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1738 \@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_␣.

```

1739 \def\bbl@redefineroobust#1{%
1740   \edef\bbl@tempa{\bbl@stripslash#1}%
1741   \bbl@ifunset{\bbl@tempa\space}%
1742   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1743     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1744   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1745   \@namedef{\bbl@tempa\space}}
1746 \@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.

```

1747 \bbl@trace{Hooks}
1748 \newcommand\AddBabelHook[3][{}]{%
1749   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}}%
1750   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1751   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1752   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1753   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1754   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1755   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1756 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1757 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1758 \def\bbl@usehooks#1#2{%
1759   \def\bbl@elth##1{%
1760     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}}%
1761     \bbl@cs{ev@#1@}%
1762     \ifx\language\@undefined\else % Test required for Plain (?)
1763       \def\bbl@elth##1{%
1764         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}}%
1765         \bbl@cl{ev@#1}%
1766       \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).

```

1767 \def\bbl@evargs{,% <- don't delete this comma

```

```

1768 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1769 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1770 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1771 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1772 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.

```

1773 \bbl@trace{Defining babelensure}
1774 \newcommand\babelensure[2][{}% TODO - revise test files
1775   \AddBabelHook{babel-ensure}{afterextras}{%
1776     \ifcase\bbl@select@type
1777       \bbl@cl{e}%
1778     \fi}%
1779   \begingroup
1780     \let\bbl@ens@include\@empty
1781     \let\bbl@ens@exclude\@empty
1782     \def\bbl@ens@fontenc{\relax}%
1783     \def\bbl@tempb##1{%
1784       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1785     \def\bbl@tempa{\bbl@tempb#1\@empty}%
1786     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1787     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1788     \def\bbl@tempc{\bbl@ensure}%
1789     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1790       \expandafter{\bbl@ens@include}}%
1791     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1792       \expandafter{\bbl@ens@exclude}}%
1793     \toks@\expandafter{\bbl@tempc}%
1794     \bbl@exp{%
1795   \endgroup
1796   \def<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1797 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1798   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1799     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1800       \edef##1{\noexpand\bbl@nocaption
1801         {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1802     \fi
1803     \ifx##1\@empty\else
1804       \in@{##1}{#2}%
1805     \ifin\else
1806       \bbl@ifunset{\bbl@ensure@\language}%
1807         {\bbl@exp{%
1808           \\DeclareRobustCommand\<\bbl@ensure@\language>[1]{%
1809             \\foreignlanguage{\language}%
1810             {\ifx\relax#3\else
1811               \\fontencoding{#3}\\selectfont
1812             \fi
1813             #####1}}}%

```

```

1814      {}%
1815      \toks@\expandafter{##1}%
1816      \edef##1{%
1817        \bbl@csarg\noexpand{ensure@\language}%
1818        {\the\toks@}}%
1819      \fi
1820      \expandafter\bbl@tempb
1821    \fi}%
1822    \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1823    \def\bbl@tempa##1{% elt for include list
1824      \ifx##1\@empty\else
1825        \bbl@csarg\in{ensure@\language\expandafter}\expandafter{##1}%
1826        \ifin\else
1827          \bbl@tempb##1\@empty
1828        \fi
1829        \expandafter\bbl@tempa
1830      \fi}%
1831    \bbl@tempa#1\@empty}
1832  \def\bbl@captionslist{%
1833    \prefacename\refname\abstractname\bibname\chaptername\appendixname
1834    \contentsname\listfigurename\listtablename\indexname\figurename
1835    \tablename\partname\encname\ccname\headtoname\pagename\seename
1836    \alsiname\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`.

```

1837 \bbl@trace{Macros for setting language files up}
1838 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1839   \let\bbl@screset\@empty
1840   \let\BabelStrings\bbl@opt@string
1841   \let\BabelOptions\@empty
1842   \let\BabelLanguages\relax
1843   \ifx\originalTeX\@undefined
1844     \let\originalTeX\@empty
1845   \else
1846     \originalTeX
1847   \fi}

```

```

1848 \def\LdfInit#1#2{%
1849   \chardef\atcatcode=\catcode`\@
1850   \catcode`\@=11\relax
1851   \chardef\eqcatcode=\catcode`\=
1852   \catcode`\==12\relax
1853   \expandafter\if\expandafter\@backslashchar
1854     \expandafter\@car\string#2@nil
1855     \ifx#2\@undefined\else
1856       \ldf@quit{#1}%
1857     \fi
1858   \else
1859     \expandafter\ifx\csname#2\endcsname\relax\else
1860       \ldf@quit{#1}%
1861     \fi
1862   \fi
1863   \bbl@ldfinit}

```

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

```

1864 \def\ldf@quit#1{%
1865   \expandafter\main@language\expandafter{#1}%
1866   \catcode`\@=\atcatcode \let\atcatcode\relax
1867   \catcode`\==\eqcatcode \let\eqcatcode\relax
1868   \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.

```

1869 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1870   \bbl@afterlang
1871   \let\bbl@afterlang\relax
1872   \let\BabelModifiers\relax
1873   \let\bbl@screset\relax}%
1874 \def\ldf@finish#1{%
1875   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1876     \loadlocalcfg{#1}%
1877   \fi
1878   \bbl@afterldf{#1}%
1879   \expandafter\main@language\expandafter{#1}%
1880   \catcode`\@=\atcatcode \let\atcatcode\relax
1881   \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 .

```

1882 \@onlypreamble\LdfInit
1883 \@onlypreamble\ldf@quit
1884 \@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.

```

1885 \def\main@language#1{%
1886   \def\bbl@main@language{#1}%
1887   \let\language\bbl@main@language % TODO. Set localename
1888   \bbl@id@assign
1889   \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`.

```

1890 \def\bbl@beforestart{%
1891   \bbl@usehooks{beforestart}{}%
1892   \global\let\bbl@beforestart\relax}
1893 \AtBeginDocument{%
1894   \@nameuse{bbl@beforestart}%
1895   \if@filesw
1896     \providecommand\babel@aux[2]{}%
1897     \immediate\write\@mainaux{%
1898       \string\providecommand\string\babel@aux[2]{}%
1899       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1900   \fi
1901   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1902   \ifbbl@single % must go after the line above.
1903     \renewcommand\selectlanguage[1]{}%
1904     \renewcommand\foreignlanguage[2]{#2}%
1905     \global\let\babel@aux\@gobbletwo % Also as flag
1906   \fi
1907   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1908 \def\select@language@x#1{%
1909   \ifcase\bbl@select@type
1910     \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1911   \else
1912     \select@language{#1}%
1913   \fi}

```

9.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if \TeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1914 \bbl@trace{Shorhands}
1915 \def\bbl@add@special#1{% 1:a macro like "\, \?, etc.
1916   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1917   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1918   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1919     \begingroup
1920       \catcode`#1\active
1921       \nfss@catcodes
1922       \ifnum\catcode`#1=\active
1923         \endgroup
1924         \bbl@add\nfss@catcodes{\@makeother#1}%
1925       \else
1926         \endgroup
1927       \fi
1928   \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.

```

1929 \def\bbl@remove@special#1{%
1930   \begingroup
1931     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1932       \else\noexpand##1\noexpand##2\fi}%
1933     \def\do{\x\do}%
1934     \def\@makeother{\x\@makeother}%
1935   \edef\x{\endgroup
1936     \def\noexpand\dospecials{\dospecials}%
1937     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1938       \def\noexpand\@sanitize{\@sanitize}%
1939     \fi}%
1940   \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).

```

1941 \def\bbl@active@def#1#2#3#4{%
1942   \@namedef{#3#1}{%
1943     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1944       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1945     \else
1946       \bbl@afterfi\csname#2@sh@#1\endcsname
1947     \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.

```

1948   \long\@namedef{#3@arg#1}##1{%
1949     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1950       \bbl@afterelse\csname#4#1\endcsname##1%
1951     \else
1952       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1953     \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.

```

1954 \def\initiate@active@char#1{%

```

```

1955 \bbl@ifunset{active@char\string#1}%
1956   {\bbl@withactive
1957     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1958   {}}

```

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).

```

1959 \def\@initiate@active@char#1#2#3{%
1960   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1961   \ifx#1\@undefined
1962     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1963   \else
1964     \bbl@csarg\let{oridef@#2}#1%
1965     \bbl@csarg\edef{oridef@#2}{%
1966       \let\noexpand#1%
1967       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1968   \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*").

```

1969   \ifx#1#3\relax
1970     \expandafter\let\csname normal@char#2\endcsname#3%
1971   \else
1972     \bbl@info{Making #2 an active character}%
1973     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1974     \@namedef{normal@char#2}{%
1975       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1976   \else
1977     \@namedef{normal@char#2}{#3}%
1978   \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).

```

1979   \bbl@restoreactive{#2}%
1980   \AtBeginDocument{%
1981     \catcode`#2\active
1982     \if@filesw
1983       \immediate\write\@mainaux{\catcode`\string#2\active}%
1984     \fi}%
1985   \expandafter\bbl@add@special\csname#2\endcsname
1986   \catcode`#2\active
1987 \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⟩).


```

1988 \let\bbl@tempa\@firstoftwo
1989 \if\string^#2%
1990 \def\bbl@tempa{\noexpand\textormath}%
1991 \else
1992 \ifx\bbl@mathnormal\@undefined\else
1993 \let\bbl@tempa\bbl@mathnormal
1994 \fi
1995 \fi
1996 \expandafter\edef\csname active@char#2\endcsname{%
1997 \bbl@tempa
1998 {\noexpand\if@safe@actives
1999 \noexpand\expandafter
2000 \expandafter\noexpand\csname normal@char#2\endcsname
2001 \noexpand\else
2002 \noexpand\expandafter
2003 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2004 \noexpand\fi}%
2005 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2006 \bbl@csarg\edef{doactive#2}{%
2007 \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!).

```

2008 \bbl@csarg\edef{active@#2}{%
2009 \noexpand\active@prefix\noexpand#1%
2010 \expandafter\noexpand\csname active@char#2\endcsname}%
2011 \bbl@csarg\edef{normal@#2}{%
2012 \noexpand\active@prefix\noexpand#1%
2013 \expandafter\noexpand\csname normal@char#2\endcsname}%
2014 \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.

```

2015 \bbl@active@def#2\user@group{user@active}{language@active}%
2016 \bbl@active@def#2\language@group{language@active}{system@active}%
2017 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as `' '` ends up in a heading \TeX would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```

2018 \expandafter\edef\csname\user@group @sh#2@@\endcsname
2019 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2020 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
2021 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (`'`) active we need to change `\pr@m@s` as well. Also, make sure that a single `'` in math mode 'does the right thing'. (2) If we are using the caret (`^`) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```

2022 \if\string'#2%
2023 \let\prim@s\bbl@prim@s

```

```

2024 \let\active@math@prime#1%
2025 \fi
2026 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

2027 <<(*More package options)>> ≡
2028 \DeclareOption{math=active}{}
2029 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2030 <</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* and the end of the ldf.

```

2031 \@ifpackagewith{babel}{KeepShorthandsActive}%
2032 {\let\bbl@restoreactive\@gobble}%
2033 {\def\bbl@restoreactive#1{%
2034   \bbl@exp{%
2035     \\\AfterBabelLanguage\\CurrentOption
2036     {\catcode`#1=\the\catcode`#1\relax}%
2037     \\\AtEndOfPackage
2038     {\catcode`#1=\the\catcode`#1\relax}}}%
2039   \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.

```

2040 \def\bbl@sh@select#1#2{%
2041   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2042     \bbl@afterelse\bbl@scndcs
2043   \else
2044     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2045   \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.

```

2046 \begingroup
2047 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2048 {\gdef\active@prefix#1{%
2049   \ifx\protect\@typeset@protect
2050   \else
2051     \ifx\protect\@unexpandable@protect
2052       \noexpand#1%
2053     \else
2054       \protect#1%
2055     \fi
2056     \expandafter\@gobble
2057   \fi}}
2058 {\gdef\active@prefix#1{%
2059   \ifincsname
2060     \string#1%

```

```

2061      \expandafter\@gobble
2062    \else
2063      \ifx\protect\@typeset@protect
2064    \else
2065      \ifx\protect\@unexpandable@protect
2066        \noexpand#1%
2067      \else
2068        \protect#1%
2069      \fi
2070      \expandafter\expandafter\expandafter\@gobble
2071    \fi
2072  \fi}}
2073 \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⟩`.

```

2074 \newif\if@safe@actives
2075 \@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.

```

2076 \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
`\bbl@deactivate` 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`.

```

2077 \def\bbl@activate#1{%
2078   \bbl@withactive{\expandafter\let\expandafter}#1%
2079   \csname bbl@active@\string#1\endcsname}
2080 \def\bbl@deactivate#1{%
2081   \bbl@withactive{\expandafter\let\expandafter}#1%
2082   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2083 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2084 \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.

```

2085 \def\babel@texpdf#1#2#3#4{%
2086   \ifx\texorpdfstring\undefined
2087     \textormath{#1}{#2}%
2088   \else

```

```

2089 \texorpdfstring{\textormath{#1}{#3}}{#2}%
2090 % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2091 \fi}
2092 %
2093 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2094 \def\@decl@short#1#2#3\@nil#4{%
2095 \def\bbl@tempa{#3}%
2096 \ifx\bbl@tempa\@empty
2097 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2098 \bbl@ifunset{#1@sh@\string#2@}{}%
2099 {\def\bbl@tempa{#4}%
2100 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2101 \else
2102 \bbl@info
2103 {Redefining #1 shorthand \string#2\\%
2104 in language \CurrentOption}%
2105 \fi}%
2106 \@namedef{#1@sh@\string#2@}{#4}%
2107 \else
2108 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2109 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2110 {\def\bbl@tempa{#4}%
2111 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2112 \else
2113 \bbl@info
2114 {Redefining #1 shorthand \string#2\string#3\\%
2115 in language \CurrentOption}%
2116 \fi}%
2117 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2118 \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.

```

2119 \def\textormath{%
2120 \ifmmode
2121 \expandafter\@secondoftwo
2122 \else
2123 \expandafter\@firstoftwo
2124 \fi}

```

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

```

2125 \def\user@group{user}
2126 \def\language@group{english} % TODO. I don't like defaults
2127 \def\system@group{system}

```

`\usesshorthands` 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.

```

2128 \def\usesshorthands{%
2129 \@ifstar\bbl@usesesh{s}\bbl@usesesh{x}}
2130 \def\bbl@usesesh@s#1{%
2131 \bbl@usesesh@x
2132 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%

```

```

2133     {#1}}
2134 \def\bbl@usesesh@x#1#2{%
2135   \bbl@ifshorthand{#2}%
2136   {\def\user@group{user}%
2137     \initiate@active@char{#2}%
2138     #1%
2139     \bbl@activate{#2}}%
2140   {\bbl@error
2141     {Cannot declare a shorthand turned off (\string#2)}
2142     {Sorry, but you cannot use shorthands which have been\\%
2143       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.

```

2144 \def\user@language@group{user@\language@group}
2145 \def\bbl@set@user@generic#1#2{%
2146   \bbl@ifunset{user@generic@active#1}%
2147   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2148     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2149     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2150       \expandafter\noexpand\csname normal@char#1\endcsname}%
2151     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2152       \expandafter\noexpand\csname user@active#1\endcsname}%
2153   \@empty}
2154 \newcommand\defineshorthand[3][user]{%
2155   \edef\bbl@tempa{\zap@space#1 \@empty}%
2156   \bbl@for\bbl@tempb\bbl@tempa{%
2157     \if*\expandafter\@car\bbl@tempb\@nil
2158       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2159       \@expandtwoargs
2160       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2161     \fi
2162     \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].

```

2163 \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`.

```

2164 \def\aliasshorthand#1#2{%
2165   \bbl@ifshorthand{#2}%
2166   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2167     \ifx\document\@notprerr
2168       \@notshorthand{#2}%
2169     \else
2170       \initiate@active@char{#2}%
2171       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2172         \csname active@char\string#1\endcsname
2173       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2174         \csname normal@char\string#1\endcsname

```

```

2175 \bbl@activate{#2}%
2176 \fi
2177 \fi}%
2178 {\bbl@error
2179 {Cannot declare a shorthand turned off (\string#2)}
2180 {Sorry, but you cannot use shorthands which have been\\%
2181 turned off in the package options}}}

```

\@notshorthand

```

2182 \def\@notshorthand#1{%
2183 \bbl@error{%
2184 The character '\string #1' should be made a shorthand character;\\%
2185 add the command \string\useshorthands\string{#1\string} to
2186 the preamble.\\%
2187 I will ignore your instruction}%
2188 {You may proceed, but expect unexpected results}}}

```

\shorthandon The first level definition of these macros just passes the argument on to \bbl@switch@sh,
\shorthandoff adding \@nil at the end to denote the end of the list of characters.

```

2189 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2190 \DeclareRobustCommand*\shorthandoff{%
2191 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2192 \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.

```

2193 \def\bbl@switch@sh#1#2{%
2194 \ifx#2\@nnil\else
2195 \bbl@ifunset{\bbl@active@\string#2}%
2196 {\bbl@error
2197 {I cannot switch '\string#2' on or off--not a shorthand}%
2198 {This character is not a shorthand. Maybe you made\\%
2199 a typing mistake? I will ignore your instruction}}}%
2200 {\ifcase#1%
2201 \catcode`#212\relax
2202 \or
2203 \catcode`#2\active
2204 \or
2205 \csname bbl@oricat@\string#2\endcsname
2206 \csname bbl@oridef@\string#2\endcsname
2207 \fi}%
2208 \bbl@afterfi\bbl@switch@sh#1%
2209 \fi}

```

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

```

2210 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2211 \def\bbl@putsh#1{%
2212 \bbl@ifunset{\bbl@active@\string#1}%
2213 {\bbl@putsh#1\@empty\@nnil}%

```

```

2214      {\csname bbl@active@string#1\endcsname}}
2215 \def\bbl@putsh@i#1#2\@nnil{%
2216   \csname\language@group @sh@string#1@%
2217     \ifx\@empty#2\else\string#2\fi\endcsname}
2218 \ifx\bbl@opt@shorthands\@nnil\else
2219   \let\bbl@s@initiate@active@char\initiate@active@char
2220   \def\initiate@active@char#1{%
2221     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2222 \let\bbl@s@switch@sh\bbl@switch@sh
2223 \def\bbl@switch@sh#1#2{%
2224   \ifx#2\@nnil\else
2225     \bbl@afterfi
2226     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2227   \fi}
2228 \let\bbl@s@activate\bbl@activate
2229 \def\bbl@activate#1{%
2230   \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2231 \let\bbl@s@deactivate\bbl@deactivate
2232 \def\bbl@deactivate#1{%
2233   \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2234 \fi

```

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

```

2235 \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.

```

2236 \def\bbl@prim@s{%
2237   \prime\futurelet\@let@token\bbl@pr@m@s}
2238 \def\bbl@if@primes#1#2{%
2239   \ifx#1\@let@token
2240     \expandafter\@firstoftwo
2241   \else\ifx#2\@let@token
2242     \bbl@afterelse\expandafter\@firstoftwo
2243   \else
2244     \bbl@afterfi\expandafter\@secondoftwo
2245   \fi\fi}
2246 \begingroup
2247   \catcode`\^=7 \catcode`\*=\active \lccode`\*='^
2248   \catcode`\'=12 \catcode`\"=\active \lccode`\"=' '
2249   \lowercase{%
2250     \gdef\bbl@pr@m@s{%
2251       \bbl@if@primes" '%
2252       \pr@@@s
2253       {\bbl@if@primes*^ \pr@@@t\egroup}}}
2254 \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).

```

2255 \initiate@active@char{~}
2256 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }

```

```
2257 \bbl@activate{~}
```

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

```
2258 \expandafter\def\csname OT1dqpos\endcsname{127}
```

```
2259 \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

```
2260 \ifx\f@encoding\@undefined
```

```
2261 \def\f@encoding{OT1}
```

```
2262 \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.

```
2263 \bbl@trace{Language attributes}
```

```
2264 \newcommand\languageattribute[2]{%
```

```
2265 \def\bbl@tempc{#1}%
```

```
2266 \bbl@fixname\bbl@tempc
```

```
2267 \bbl@iflanguage\bbl@tempc{%
```

```
2268 \bbl@vforeach{#2}{%
```

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

```
2269 \ifx\bbl@known@attrs\@undefined
```

```
2270 \in@false
```

```
2271 \else
```

```
2272 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
```

```
2273 \fi
```

```
2274 \ifin@
```

```
2275 \bbl@warning{%
```

```
2276 You have more than once selected the attribute '##1'\%
```

```
2277 for language #1. Reported}%
```

```
2278 \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.

```
2279 \bbl@exp{%
```

```
2280 \bbl@add@list\bbl@known@attrs{\bbl@tempc-##1}}%
```

```
2281 \edef\bbl@tempa{\bbl@tempc-##1}%
```

```
2282 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
```

```
2283 {\csname\bbl@tempc @attr##1\endcsname}%
```

```
2284 {\@attrerr{\bbl@tempc}{##1}}%
```

```
2285 \fi}}
```

```
2286 \@onlypreamble\languageattribute
```

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

```
2287 \newcommand*{\@attrerr}[2]{%
```

```
2288 \bbl@error
```

```
2289 {The attribute #2 is unknown for language #1.}%
```

```
2290 {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}`.

```

2291 \def\bbl@declare@ttribute#1#2#3{%
2292   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2293   \ifin@
2294     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2295   \fi
2296   \bbl@add@list\bbl@attributes{#1-#2}%
2297   \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.
First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```

2298 \def\bbl@ifattributeset#1#2#3#4{%
2299   \ifx\bbl@known@attribs\@undefined
2300     \in@false
2301   \else
2302     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2303   \fi
2304   \ifin@
2305     \bbl@afterelse#3%
2306   \else
2307     \bbl@afterfi#4%
2308   \fi
2309 }

```

`\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. When a match is found the definition of `\bbl@tempa` is changed. Finally we execute `\bbl@tempa`.

```

2310 \def\bbl@ifknown@ttrib#1#2{%
2311   \let\bbl@tempa\@secondoftwo
2312   \bbl@loopx\bbl@tempb{#2}{%
2313     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2314     \ifin@
2315       \let\bbl@tempa\@firstoftwo
2316     \else
2317       \fi}%
2318   \bbl@tempa
2319 }

```

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

```

2320 \def\bbl@clear@ttribs{%
2321   \ifx\bbl@attributes\@undefined\else
2322     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2323       \expandafter\bbl@clear@ttrib\bbl@tempa.
2324     }%
2325     \let\bbl@attributes\@undefined
2326   \fi}
2327 \def\bbl@clear@ttrib#1-#2.{%
2328   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2329 \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` 2330 `\bbl@trace{Macros for saving definitions}`
 2331 `\def\babel@beginsave{\babel@savecnt\z@}`

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

```

2332 \newcount\babel@savecnt
2333 \babel@beginsave

```

`\babel@save` The macro `\babel@save<csname>` saves the current meaning of the control sequence
`\babel@savevariable` `<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.

```

2334 \def\babel@save#1{%
2335   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2336   \toks@\expandafter{\originalTeX\let#1=}%
2337   \bbl@exp{%
2338     \def\\originalTeX{\the\toks@<\babel@\number\babel@savecnt>\relax}}%
2339   \advance\babel@savecnt\@ne}
2340 \def\babel@savevariable#1{%
2341   \toks@\expandafter{\originalTeX #1=}%
2342   \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
`\bbl@nonfrenchspacing` command `\bbl@frenchspacing` switches it on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary.

```

2343 \def\bbl@frenchspacing{%
2344   \ifnum\the\sfcodes\@m
2345     \let\bbl@nonfrenchspacing\relax
2346   \else
2347     \frenchspacing
2348     \let\bbl@nonfrenchspacing\nonfrenchspacing
2349   \fi}
2350 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

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

```

2351 %
2352 \let\bbl@elt\relax
2353 \edef\bbl@fs@chars{%
2354   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2355   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2356   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}

```

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.

```

2357 \bbl@trace{Short tags}
2358 \def\babeltags#1{%
2359   \edef\bbl@tempa{\zap@space#1 \@empty}%
2360   \def\bbl@tempb##1=##2\@{%
2361     \edef\bbl@tempc{%
2362       \noexpand\newcommand
2363       \expandafter\noexpand\csname ##1\endcsname{%
2364         \noexpand\protect
2365         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2366       \noexpand\newcommand
2367       \expandafter\noexpand\csname text##1\endcsname{%
2368         \noexpand\foreignlanguage{##2}}
2369     \bbl@tempc}%
2370   \bbl@for\bbl@tempa\bbl@tempa{%
2371     \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.

```

2372 \bbl@trace{Hyphens}
2373 \@onlypreamble\babelhyphenation
2374 \AtEndOfPackage{%
2375   \newcommand\babelhyphenation[2][\@empty]{%
2376     \ifx\bbl@hyphenation@relax
2377       \let\bbl@hyphenation@\@empty
2378     \fi
2379     \ifx\bbl@hyphlist\@empty\else
2380       \bbl@warning{%
2381         You must not intermingle \string\selectlanguage\space and\%
2382         \string\babelhyphenation\space or some exceptions will not\%
2383         be taken into account. Reported}%
2384     \fi
2385     \ifx\@empty#1%
2386       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2387     \else
2388       \bbl@vforeach{#1}{%
2389         \def\bbl@tempa{##1}%
2390         \bbl@fixname\bbl@tempa
2391         \bbl@iflanguage\bbl@tempa{%
2392           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2393             \bbl@ifunset\bbl@hyphenation@\bbl@tempa}%
2394           \@empty

```

```

2395          {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2396          #2}}}%
2397      \fi}}

```

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

```

2398 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2399 \def\bbl@t@one{T1}
2400 \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.

```

2401 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2402 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2403 \def\bbl@hyphen{%
2404   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2405 \def\bbl@hyphen@i#1#2{%
2406   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
2407   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2408   {\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.

```

2409 \def\bbl@usehyphen#1{%
2410   \leavevmode
2411   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2412   \nobreak\hskip\z@skip}
2413 \def\bbl@usehyphen#1{%
2414   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2415 \def\bbl@hyphenchar{%
2416   \ifnum\hyphenchar\font=\m@ne
2417     \babellnullhyphen
2418   \else
2419     \char\hyphenchar\font
2420   \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.

```

2421 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2422 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2423 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2424 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2425 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2426 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2427 \def\bbl@hy@repeat{%
2428   \bbl@usehyphen{%
2429     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2430 \def\bbl@hy@repeat{%

```

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

```

2431 \bbl@usehyphen{%
2432   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2433 \def\bbl@hy@empty{\hskip\z@skip}
2434 \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.

```

2435 \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.

```

2436 \bbl@trace{Multiencoding strings}
2437 \def\bbl@tglobal#1{\global\let#1#1}
2438 \def\bbl@recatcode#1{% TODO. Used only once?
2439   \@tempcnta="7F
2440   \def\bbl@tempa{%
2441     \ifnum\@tempcnta>"FF\else
2442       \catcode\@tempcnta=#1\relax
2443       \advance\@tempcnta\@ne
2444       \expandafter\bbl@tempa
2445     \fi}%
2446   \bbl@tempa}

```

The second one. We need to patch \@uclclist, but it is done once and only if \SetCase is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact \@uclclist is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually \reserved@a), we pass it as argument to \bbl@ucllc. The parser is restarted inside \<lang>\bbl@ucllc because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

```
\let\bbl@tolower\@empty\bbl@toupper\@empty
```

and starts over (and similarly when lowercasing).

```

2447 \@ifpackagewith{babel}{nocase}%
2448   {\let\bbl@patchucllc\relax}%
2449   {\def\bbl@patchucllc{%
2450     \global\let\bbl@patchucllc\relax
2451     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@ucllc}}%
2452     \gdef\bbl@ucllc##1{%
2453       \let\bbl@encoded\bbl@encoded@ucllc
2454       \bbl@ifunset{\language @bbl@ucllc}% and resumes it
2455       {##1}%
2456       {\let\bbl@tempa##1\relax % Used by LANG@bbl@ucllc
2457        \csname\language @bbl@ucllc\endcsname}%
2458       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2459     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2460     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}

```

```

2461 <<*More package options>> ≡
2462 \DeclareOption{nocase}{}
2463 <</More package options>>

```

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

```

2464 <<*More package options>> ≡
2465 \let\bbl@opt@strings\@nnil % accept strings=value
2466 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2467 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2468 \def\BabelStringsDefault{generic}
2469 <</More package options>>

```

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

```

2470 \@onlypreamble\StartBabelCommands
2471 \def\StartBabelCommands{%
2472   \begingroup
2473   \bbl@recatcode{11}%
2474   <<Macros local to BabelCommands>>
2475   \def\bbl@provstring##1##2{%
2476     \providecommand##1{##2}%
2477     \bbl@tglobal##1}%
2478   \global\let\bbl@scafter\@empty
2479   \let\StartBabelCommands\bbl@startcmds
2480   \ifx\BabelLanguages\relax
2481     \let\BabelLanguages\CurrentOption
2482   \fi
2483   \begingroup
2484   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2485   \StartBabelCommands}
2486 \def\bbl@startcmds{%
2487   \ifx\bbl@screset\@nnil\else
2488     \bbl@usehooks{stopcommands}{}%
2489   \fi
2490   \endgroup
2491   \begingroup
2492   \@ifstar
2493   {\ifx\bbl@opt@strings\@nnil
2494     \let\bbl@opt@strings\BabelStringsDefault
2495     \fi
2496     \bbl@startcmds@i}%
2497   \bbl@startcmds@i}
2498 \def\bbl@startcmds@i#1#2{%
2499   \edef\bbl@L{\zap@space#1 \@empty}%
2500   \edef\bbl@G{\zap@space#2 \@empty}%
2501   \bbl@startcmds@ii}
2502 \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.

```

2503 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2504   \let\SetString\@gobbletwo
2505   \let\bbl@stringdef\@gobbletwo
2506   \let\AfterBabelCommands\@gobble
2507   \ifx\@empty#1%
2508     \def\bbl@sc@label{generic}%
2509     \def\bbl@encstring##1##2{%
2510       \ProvideTextCommandDefault##1{##2}%
2511       \bbl@tglobal##1%
2512       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2513     \let\bbl@sctest\in@true
2514   \else
2515     \let\bbl@sc@charset\space % <- zapped below
2516     \let\bbl@sc@fontenc\space % <- " "
2517     \def\bbl@tempa##1=##2\@nil{%
2518       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2519     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2520     \def\bbl@tempa##1 ##2{% space -> comma
2521       ##1%
2522       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2523     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2524     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2525     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2526     \def\bbl@encstring##1##2{%
2527       \bbl@foreach\bbl@sc@fontenc{%
2528         \bbl@ifunset{T#####1}%
2529         {}%
2530         {\ProvideTextCommand##1{#####1}{##2}%
2531         \bbl@tglobal##1%
2532         \expandafter
2533         \bbl@tglobal\csname#####1\string##1\endcsname}}}%
2534     \def\bbl@sctest{%
2535       \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
2536   \fi
2537   \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2538   \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2539     \let\AfterBabelCommands\bbl@aftercmds
2540     \let\SetString\bbl@setstring
2541     \let\bbl@stringdef\bbl@encstring
2542   \else % ie, strings=value
2543     \bbl@sctest
2544   \fin@
2545     \let\AfterBabelCommands\bbl@aftercmds
2546     \let\SetString\bbl@setstring
2547     \let\bbl@stringdef\bbl@provstring
2548   \fi\fi\fi
2549   \bbl@scswitch
2550   \ifx\bbl@G\@empty
2551     \def\SetString##1##2{%
2552       \bbl@error{Missing group for string \string##1}%
2553       {You must assign strings to some category, typically\\%
2554       captions or extras, but you set none}}%
2555   \fi
2556   \ifx\@empty#1%
2557     \bbl@usehooks{defaultcommands}{}%

```

```

2558 \else
2559 \expandafter\@expandtwoargs
2560 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
2561 \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) .

```

2562 \def\bbl@forlang#1#2{%
2563 \bbl@for#1\bbl@L{%
2564 \bbl@xin{,#1,},{\BabelLanguages,}%
2565 \ifin#2\relax\fi}}
2566 \def\bbl@scswitch{%
2567 \bbl@forlang\bbl@tempa{%
2568 \ifx\bbl@G\@empty\else
2569 \ifx\SetString\@gobbletwo\else
2570 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2571 \bbl@xin{\bbl@GL,}{\bbl@screset,}%
2572 \ifin\else
2573 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2574 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2575 \fi
2576 \fi
2577 \fi}}
2578 \AtEndOfPackage{%
2579 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
2580 \let\bbl@scswitch\relax}
2581 \onlypreamble\EndBabelCommands
2582 \def\EndBabelCommands{%
2583 \bbl@usehooks{stopcommands}{}%
2584 \endgroup
2585 \endgroup
2586 \bbl@scafter}
2587 \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.

```

2588 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2589 \bbl@forlang\bbl@tempa{%
2590 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2591 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2592 {\bbl@exp{%
2593 \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2594 {}}%
2595 \def\BabelString{#2}%
2596 \bbl@usehooks{stringprocess}{}%

```



```

2597 \expandafter\bb1@stringdef
2598 \csname\bb1@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include `\bb1@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`.

```

2599 \ifx\bb1@opt@strings\relax
2600 \def\bb1@scset#1#2{\def#1{\bb1@encoded#2}}
2601 \bb1@patchuclc
2602 \let\bb1@encoded\relax
2603 \def\bb1@encoded@uclc#1{%
2604   \@inmathwarn#1%
2605   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2606     \expandafter\ifx\csname ?\string#1\endcsname\relax
2607       \TextSymbolUnavailable#1%
2608     \else
2609       \csname ?\string#1\endcsname
2610     \fi
2611   \else
2612     \csname\cf@encoding\string#1\endcsname
2613   \fi}
2614 \else
2615 \def\bb1@scset#1#2{\def#1{#2}}
2616 \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.

```

2617 <<(*Macros local to BabelCommands)>> ≡
2618 \def\SetStringLoop##1##2{%
2619   \def\bb1@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2620   \count@\z@
2621   \bb1@loop\bb1@tempa{##2}{% empty items and spaces are ok
2622     \advance\count@\@ne
2623     \toks@\expandafter{\bb1@tempa}%
2624     \bb1@exp{%
2625       \\SetString\bb1@templ{\romannumeral\count@}{\the\toks@}%
2626       \count@=\the\count@\relax}}}%
2627 <</Macros local to BabelCommands>>

```

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

```

2628 \def\bb1@aftercmds#1{%
2629   \toks@\expandafter{\bb1@scafter#1}%
2630   \xdef\bb1@scafter{\the\toks@}}

```

Case mapping The command `\SetCase` provides a way to change the behavior of `\MakeUppercase` and `\MakeLowercase`. `\bb1@tempa` is set by the patched `@uclclist` to the parsing command.

```

2631 <<(*Macros local to BabelCommands)>> ≡
2632 \newcommand\SetCase[3][{}]{%
2633   \bb1@patchuclc
2634   \bb1@forlang\bb1@tempa{%
2635     \expandafter\bb1@encstring
2636     \csname\bb1@tempa @bb1@uclc\endcsname{\bb1@tempa##1}%

```

```

2637 \expandafter\bb1@encstring
2638 \csname\bb1@tempa @bb1@uc\endcsname{##2}%
2639 \expandafter\bb1@encstring
2640 \csname\bb1@tempa @bb1@lc\endcsname{##3}}}%
2641 <</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.

```

2642 <<(*Macros local to BabelCommands)>> ≡
2643 \newcommand\SetHyphenMap[1]{%
2644 \bb1@forlang\bb1@tempa{%
2645 \expandafter\bb1@stringdef
2646 \csname\bb1@tempa @bb1@hyphenmap\endcsname{##1}}}%
2647 <</Macros local to BabelCommands>>

```

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

```

2648 \newcommand\BabelLower[2]{% one to one.
2649 \ifnum\lccode#1=#2\else
2650 \babel@savevariable{\lccode#1}%
2651 \lccode#1=#2\relax
2652 \fi}
2653 \newcommand\BabelLowerMM[4]{% many-to-many
2654 \@tempcnta=#1\relax
2655 \@tempcntb=#4\relax
2656 \def\bb1@tempa{%
2657 \ifnum\@tempcnta>#2\else
2658 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2659 \advance\@tempcnta#3\relax
2660 \advance\@tempcntb#3\relax
2661 \expandafter\bb1@tempa
2662 \fi}%
2663 \bb1@tempa}
2664 \newcommand\BabelLowerMO[4]{% many-to-one
2665 \@tempcnta=#1\relax
2666 \def\bb1@tempa{%
2667 \ifnum\@tempcnta>#2\else
2668 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2669 \advance\@tempcnta#3
2670 \expandafter\bb1@tempa
2671 \fi}%
2672 \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2673 <<(*More package options)>> ≡
2674 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
2675 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap@ne}
2676 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
2677 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
2678 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2679 <</More package options>>

```

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

```

2680 \AtEndOfPackage{%
2681 \ifx\bb1@opt@hyphenmap\undefined
2682 \bb1@xin@{,}{\bb1@language@opts}%
2683 \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
2684 \fi}

```

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```

2685 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2686 \@ifstar\bb1@setcaption@s\bb1@setcaption@x}
2687 \def\bb1@setcaption@x#1#2#3{% language caption-name string
2688 \edef\bb1@tempa{#1}%
2689 \edef\bb1@tempd{%
2690 \expandafter\expandafter\expandafter
2691 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2692 \bb1@xin@
2693 {\expandafter\string\csname #2name\endcsname}%
2694 {\bb1@tempd}%
2695 \ifin@ % Renew caption
2696 \bb1@xin@{\string\bb1@scset}{\bb1@tempd}%
2697 \ifin@
2698 \bb1@exp{%
2699 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2700 {\\\bb1@scset\<#2name>\<#1#2name>}}%
2701 {}}%
2702 \else % Old way converts to new way
2703 \bb1@ifunset{#1#2name}%
2704 {\bb1@exp{%
2705 \\\bb1@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2706 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2707 {\def\<#2name>\<#1#2name>}}%
2708 {}}%
2709 {}%
2710 \fi
2711 \else
2712 \bb1@xin@{\string\bb1@scset}{\bb1@tempd}% New
2713 \ifin@ % New way
2714 \bb1@exp{%
2715 \\\bb1@add\<captions#1>\\\bb1@scset\<#2name>\<#1#2name>}}%
2716 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2717 {\\\bb1@scset\<#2name>\<#1#2name>}}%
2718 {}}%
2719 \else % Old way, but defined in the new way
2720 \bb1@exp{%
2721 \\\bb1@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2722 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2723 {\def\<#2name>\<#1#2name>}}%
2724 {}}%
2725 \fi%
2726 \fi
2727 \@namedef{#1#2name}{#3}%
2728 \toks@ \expandafter{\bb1@captionslist}%
2729 \bb1@exp{\in@{\<#2name>}{\the\toks@}}%
2730 \ifin@ \else
2731 \bb1@exp{\\\bb1@add\\bb1@captionslist{\<#2name>}}%
2732 \bb1@tglobal\bb1@captionslist
2733 \fi}
2734 % \def\bb1@setcaption@s#1#2#3{} % 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.

```
2735 \bbl@trace{Macros related to glyphs}
2736 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2737   \dimen\z@ht\z@ \advance\dimen\z@ -ht\tw@%
2738   \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.

```
2739 \def\save@sf@q#1{\leavevmode
2740   \begingroup
2741   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2742   \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.

```
2743 \ProvideTextCommand{\quotedblbase}{OT1}{%
2744   \save@sf@q{\set@low@box{\textquotedblright\}%
2745     \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2746 \ProvideTextCommandDefault{\quotedblbase}{%
2747   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2748 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2749   \save@sf@q{\set@low@box{\textquoteright\}%
2750     \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2751 \ProvideTextCommandDefault{\quotesinglbase}{%
2752   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2753 \ProvideTextCommand{\guillemetleft}{OT1}{%
2754   \ifmmode
2755     \ll
2756   \else
2757     \save@sf@q{\nobreak
2758       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2759   \fi}
2760 \ProvideTextCommand{\guillemetright}{OT1}{%
2761   \ifmmode
2762     \gg
```

```

2763 \else
2764   \save@sf@q{\nobreak
2765     \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2766   \fi}
2767 \ProvideTextCommand{\guillemotleft}{OT1}{%
2768   \ifmmode
2769     \ll
2770   \else
2771     \save@sf@q{\nobreak
2772       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2773     \fi}
2774 \ProvideTextCommand{\guillemotright}{OT1}{%
2775   \ifmmode
2776     \gg
2777   \else
2778     \save@sf@q{\nobreak
2779       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2780     \fi}

```

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

```

2781 \ProvideTextCommandDefault{\guillemetleft}{%
2782   \UseTextSymbol{OT1}{\guillemetleft}}
2783 \ProvideTextCommandDefault{\guillemetright}{%
2784   \UseTextSymbol{OT1}{\guillemetright}}
2785 \ProvideTextCommandDefault{\guillemotleft}{%
2786   \UseTextSymbol{OT1}{\guillemotleft}}
2787 \ProvideTextCommandDefault{\guillemotright}{%
2788   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2789 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2790   \ifmmode
2791     <%
2792   \else
2793     \save@sf@q{\nobreak
2794       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2795     \fi}
2796 \ProvideTextCommand{\guilsinglright}{OT1}{%
2797   \ifmmode
2798     >%
2799   \else
2800     \save@sf@q{\nobreak
2801       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2802     \fi}

```

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

```

2803 \ProvideTextCommandDefault{\guilsinglleft}{%
2804   \UseTextSymbol{OT1}{\guilsinglleft}}
2805 \ProvideTextCommandDefault{\guilsinglright}{%
2806   \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 fonts. Therefore we fake it for the OT1 encoding.

```

2807 \DeclareTextCommand{\ij}{OT1}{%

```

```

2808 i\kern-0.02em\bbl@allowhyphens j}
2809 \DeclareTextCommand{\IJ}{OT1}{%
2810 I\kern-0.02em\bbl@allowhyphens J}
2811 \DeclareTextCommand{\ij}{T1}{\char188}
2812 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2813 \ProvideTextCommandDefault{\ij}{%
2814 \UseTextSymbol{OT1}{\ij}}
2815 \ProvideTextCommandDefault{\IJ}{%
2816 \UseTextSymbol{OT1}{\IJ}}

```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, `\DJ` but not in 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).

```

2817 \def\crrtic@{\hrule height0.1ex width0.3em}
2818 \def\crttic@{\hrule height0.1ex width0.33em}
2819 \def\ddj@{%
2820 \setbox0\hbox{d}\dimen@=\ht0
2821 \advance\dimen@1ex
2822 \dimen@.45\dimen@
2823 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2824 \advance\dimen@ii.5ex
2825 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2826 \def\DDJ@{%
2827 \setbox0\hbox{D}\dimen@=.55\ht0
2828 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2829 \advance\dimen@ii.15ex % correction for the dash position
2830 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2831 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2832 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2833 %
2834 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2835 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2836 \ProvideTextCommandDefault{\dj}{%
2837 \UseTextSymbol{OT1}{\dj}}
2838 \ProvideTextCommandDefault{\DJ}{%
2839 \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.

```

2840 \DeclareTextCommand{\SS}{OT1}{SS}
2841 \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 2842 \ProvideTextCommandDefault{\glq}{%
2843 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2844 \ProvideTextCommand{\grq}{T1}{%
2845 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2846 \ProvideTextCommand{\grq}{TU}{%
2847 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2848 \ProvideTextCommand{\grq}{OT1}{%
2849 \save@sf@q{\kern-.0125em
2850 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2851 \kern.07em\relax}}
2852 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2853 \ProvideTextCommandDefault{\glqq}{%
2854 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2855 \ProvideTextCommand{\grqq}{T1}{%
2856 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2857 \ProvideTextCommand{\grqq}{TU}{%
2858 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2859 \ProvideTextCommand{\grqq}{OT1}{%
2860 \save@sf@q{\kern-.07em
2861 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2862 \kern.07em\relax}}
2863 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2864 \ProvideTextCommandDefault{\flq}{%
2865 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2866 \ProvideTextCommandDefault{\frq}{%
2867 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2868 \ProvideTextCommandDefault{\flqq}{%
2869 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2870 \ProvideTextCommandDefault{\frqq}{%
2871 \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
`\umlautlow` positioning, the default will be `\umlauthigh` (the normal positioning).

```
2872 \def\umlauthigh{%
2873 \def\bbl@umlauta##1{\leavevmode\bggroup%
2874 \expandafter\accent\csname\f@encoding dqpos\endcsname
```

```

2875      ##1\bb1@allowhyphens\egroup}%
2876 \let\bb1@umlaut\bb1@umlauta}
2877 \def\umlautlow{%
2878 \def\bb1@umlauta{\protect\lower@umlaut}}
2879 \def\umlautelow{%
2880 \def\bb1@umlaute{\protect\lower@umlaut}}
2881 \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.

```

2882 \expandafter\ifx\csname U@D\endcsname\relax
2883 \csname newdimen\endcsname\U@D
2884 \fi

```

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

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

```

2885 \def\lower@umlaut#1{%
2886 \leavevmode\bgroup
2887 \U@D 1ex%
2888 {\setbox\z@\hbox{%
2889 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2890 \dimen@ -.45ex\advance\dimen@\ht\z@
2891 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2892 \expandafter\accent\csname\fontencoding dqpos\endcsname
2893 \fontdimen5\font\U@D #1%
2894 \egroup}

```

For all vowels we declare \" to be a composite command which uses \bb1@umlauta or \bb1@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 \bb1@umlauta and/or \bb1@umlaute for a language in the corresponding ldf (using the babel switching mechanism, of course).

```

2895 \AtBeginDocument{%
2896 \DeclareTextCompositeCommand{\"}{OT1}{a}{\bb1@umlauta{a}}%
2897 \DeclareTextCompositeCommand{\"}{OT1}{e}{\bb1@umlaute{e}}%
2898 \DeclareTextCompositeCommand{\"}{OT1}{i}{\bb1@umlaute{i}}%
2899 \DeclareTextCompositeCommand{\"}{OT1}{i}{\bb1@umlaute{i}}%
2900 \DeclareTextCompositeCommand{\"}{OT1}{o}{\bb1@umlauta{o}}%
2901 \DeclareTextCompositeCommand{\"}{OT1}{u}{\bb1@umlauta{u}}%
2902 \DeclareTextCompositeCommand{\"}{OT1}{A}{\bb1@umlauta{A}}%
2903 \DeclareTextCompositeCommand{\"}{OT1}{E}{\bb1@umlaute{E}}%
2904 \DeclareTextCompositeCommand{\"}{OT1}{I}{\bb1@umlaute{I}}%
2905 \DeclareTextCompositeCommand{\"}{OT1}{O}{\bb1@umlauta{O}}%
2906 \DeclareTextCompositeCommand{\"}{OT1}{U}{\bb1@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.


```

2907 \ifx\l@english\@undefined
2908   \chardef\l@english\z@
2909 \fi
2910 % The following is used to cancel rules in ini files (see Amharic).
2911 \ifx\l@babelnohyphens\@undefined
2912   \newlanguage\l@babelnohyphens
2913 \fi

```

9.13 Layout

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

```

2914 \bbl@trace{Bidi layout}
2915 \providecommand\IfBabelLayout[3]{#3}%
2916 \newcommand\BabelPatchSection[1]{%
2917   \@ifundefined{#1}{}{%
2918     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2919     \@namedef{#1}{%
2920       \@ifstar{\bbl@presec@#1}%
2921       {\@dblarg{\bbl@presec@#1}}}%
2922 \def\bbl@presec@#1[#2]#3{%
2923   \bbl@exp{%
2924     \\\select@language@x{\bbl@main@language}%
2925     \\\bbl@cs{sspre@#1}%
2926     \\\bbl@cs{ss@#1}%
2927     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2928     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2929     \\\select@language@x{\language}}}%
2930 \def\bbl@presec@#1#2{%
2931   \bbl@exp{%
2932     \\\select@language@x{\bbl@main@language}%
2933     \\\bbl@cs{sspre@#1}%
2934     \\\bbl@cs{ss@#1}*%
2935     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2936     \\\select@language@x{\language}}}%
2937 \IfBabelLayout{sectioning}%
2938   {\BabelPatchSection{part}%
2939    \BabelPatchSection{chapter}%
2940    \BabelPatchSection{section}%
2941    \BabelPatchSection{subsection}%
2942    \BabelPatchSection{subsubsection}%
2943    \BabelPatchSection{paragraph}%
2944    \BabelPatchSection{subparagraph}%
2945    \def\babel@toc#1{%
2946      \select@language@x{\bbl@main@language}}}%
2947 \IfBabelLayout{captions}%
2948   {\BabelPatchSection{caption}}%

```

9.14 Load engine specific macros

```

2949 \bbl@trace{Input engine specific macros}
2950 \ifcase\bbl@engine
2951   \input txtbabel.def
2952 \or
2953   \input luababel.def
2954 \or
2955   \input xebabel.def
2956 \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.

```
2957 \bbl@trace{Creating languages and reading ini files}
2958 \newcommand\babelprovide[2][{}]{%
2959   \let\bbl@savelangname\language
2960   \edef\bbl@savelocaleid{\the\localeid}%
2961   % Set name and locale id
2962   \edef\language{#2}%
2963   % \global\@namedef\bbl@lcname{#2}{#2}%
2964   \bbl@id@assign
2965   \let\bbl@KVP@captions\@nil
2966   \let\bbl@KVP@date\@nil
2967   \let\bbl@KVP@import\@nil
2968   \let\bbl@KVP@main\@nil
2969   \let\bbl@KVP@script\@nil
2970   \let\bbl@KVP@language\@nil
2971   \let\bbl@KVP@hyphenrules\@nil
2972   \let\bbl@KVP@mapfont\@nil
2973   \let\bbl@KVP@maparabic\@nil
2974   \let\bbl@KVP@mapdigits\@nil
2975   \let\bbl@KVP@intraspace\@nil
2976   \let\bbl@KVP@intrapenalty\@nil
2977   \let\bbl@KVP@onchar\@nil
2978   \let\bbl@KVP@alph\@nil
2979   \let\bbl@KVP@Alph\@nil
2980   \let\bbl@KVP@labels\@nil
2981   \bbl@csarg\let{KVP@labels*}\@nil
2982   \bbl@forkv{#1}{% TODO - error handling
2983     \in@{/}{##1}%
2984     \ifin@
2985       \bbl@renewinikey##1\@{##2}%
2986     \else
2987       \bbl@csarg\def{KVP@##1}{##2}%
2988     \fi}%
2989   % == import, captions ==
2990   \ifx\bbl@KVP@import\@nil\else
2991     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2992     {\ifx\bbl@initoload\relax
2993       \begingroup
2994         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2995         \bbl@input@texini{##2}%
2996       \endgroup
2997     \else
2998       \xdef\bbl@KVP@import{\bbl@initoload}%
2999     \fi}%
3000   }%
3001 \fi
3002 \ifx\bbl@KVP@captions\@nil
3003   \let\bbl@KVP@captions\bbl@KVP@import
3004 \fi
3005 % Load ini
3006 \bbl@ifunset{date#2}%
3007   {\bbl@provide@new{#2}}%
3008   {\bbl@ifblank{#1}%
3009     {\bbl@error
```

```

3010         {If you want to modify `#2' you must tell how in\\%
3011         the optional argument. See the manual for the\\%
3012         available options.}%
3013         {Use this macro as documented}}%
3014         {\bbl@provide@renew{#2}}}%
3015 % Post tasks
3016 \bbl@ifunset{\bbl@extracaps@#2}%
3017     {\bbl@exp{\babelensure[exclude=\\today]{#2}}}%
3018     {\toks@\\expandafter\\expandafter\\expandafter
3019     {\csname bbl@extracaps@#2\endcsname}%
3020     \bbl@exp{\babelensure[exclude=\\today,include=\\the\toks@]{#2}}}%
3021 \bbl@ifunset{\bbl@ensure@\\language}%
3022     {\bbl@exp{%
3023         \\DeclareRobustCommand<\bbl@ensure@\\language>[1]{%
3024             \\foreignlanguage{\\language}%
3025             {###1}}}%
3026     }%
3027 \bbl@exp{%
3028     \\bbl@toglobal<\bbl@ensure@\\language>%
3029     \\bbl@toglobal<\bbl@ensure@\\language\\space>%
3030 % At this point all parameters are defined if 'import'. Now we
3031 % execute some code depending on them. But what about if nothing was
3032 % imported? We just load the very basic parameters.
3033 \bbl@load@basic{#2}%
3034 % == script, language ==
3035 % Override the values from ini or defines them
3036 \ifx\bbl@KVP@script\\nil\\else
3037     \bbl@csarg\\edef{sname@#2}{\bbl@KVP@script}%
3038 \fi
3039 \ifx\bbl@KVP@language\\nil\\else
3040     \bbl@csarg\\edef{lname@#2}{\bbl@KVP@language}%
3041 \fi
3042 % == onchar ==
3043 \ifx\bbl@KVP@onchar\\nil\\else
3044     \bbl@luahyphenate
3045     \directlua{
3046         if Babel.locale_mapped == nil then
3047             Babel.locale_mapped = true
3048             Babel.linebreaking.add_before(Babel.locale_map)
3049             Babel.loc_to_scr = {}
3050             Babel.chr_to_loc = Babel.chr_to_loc or {}
3051         end}%
3052 \bbl@xin@{ ids }{ \bbl@KVP@onchar\\space}%
3053 \ifin@
3054     \ifx\bbl@starthyphens\\undefined % Needed if no explicit selection
3055         \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
3056     \fi
3057     \bbl@exp{\bbl@add\bbl@starthyphens
3058     {\bbl@patterns@lua{\\language}}}%
3059 % TODO - error/warning if no script
3060     \directlua{
3061         if Babel.script_blocks['\bbl@cl{sbcp}'] then
3062             Babel.loc_to_scr[\the\\localeid] =
3063             Babel.script_blocks['\bbl@cl{sbcp}']
3064             Babel.locale_props[\the\\localeid].lc = \the\\localeid\\space
3065             Babel.locale_props[\the\\localeid].lg = \the\\nameuse{1@\\language}\\space
3066         end
3067     }%
3068 \fi

```

```

3069 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3070 \ifin@
3071 \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{}%
3072 \bbl@ifunset{bbl@wdir@\languageName}{\bbl@provide@dirs{\languageName}}{}%
3073 \directlua{
3074   if Babel.script_blocks['\bbl@cl{sbc}'] then
3075     Babel.loc_to_scr[\the\localeid] =
3076       Babel.script_blocks['\bbl@cl{sbc}']
3077   end}%
3078 \ifx\bbl@mapselect\undefined
3079   \AtBeginDocument{%
3080     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3081     {\selectfont}}%
3082   \def\bbl@mapselect{%
3083     \let\bbl@mapselect\relax
3084     \edef\bbl@prefontid{\fontid\font}}%
3085   \def\bbl@mapdir##1{%
3086     {\def\languageName{##1}%
3087       \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3088       \bbl@switchfont
3089       \directlua{
3090         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
3091           ['\bbl@prefontid'] = \fontid\font\space}}}%
3092   \fi
3093   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languageName}}}%
3094   \fi
3095   % TODO - catch non-valid values
3096 \fi
3097 % == mapfont ==
3098 % For bidi texts, to switch the font based on direction
3099 \ifx\bbl@KVP@mapfont\@nil\else
3100   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
3101   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\the
3102     mapfont. Use 'direction'.%
3103     {See the manual for details.}}}
3104   \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{}%
3105   \bbl@ifunset{bbl@wdir@\languageName}{\bbl@provide@dirs{\languageName}}{}%
3106   \ifx\bbl@mapselect\undefined
3107     \AtBeginDocument{%
3108       \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3109       {\selectfont}}%
3110     \def\bbl@mapselect{%
3111       \let\bbl@mapselect\relax
3112       \edef\bbl@prefontid{\fontid\font}}%
3113     \def\bbl@mapdir##1{%
3114       {\def\languageName{##1}%
3115         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3116         \bbl@switchfont
3117         \directlua{Babel.fontmap
3118           [\the\csname bbl@wdir@##1\endcsname]%
3119           [\bbl@prefontid]=\fontid\font}}}%
3120     \fi
3121     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languageName}}}%
3122   \fi
3123   % == Line breaking: intraspace, intrapenalty ==
3124   % For CJK, East Asian, Southeast Asian, if interspace in ini
3125   \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3126     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3127   \fi

```

```

3128 \bbl@provide@intraspace
3129 % == Line breaking: hyphenate.other.locale ==
3130 \bbl@ifunset{bbl@hyotl@languagename}{}%
3131 {\bbl@csarg\bbl@replace{hyotl@languagename}{ }{,}%
3132 \bbl@startcommands*{languagename}{}%
3133 \bbl@csarg\bbl@foreach{hyotl@languagename}{%
3134 \ifcase\bbl@engine
3135 \ifnum##1<257
3136 \SetHyphenMap{\BabelLower{##1}{##1}}%
3137 \fi
3138 \else
3139 \SetHyphenMap{\BabelLower{##1}{##1}}%
3140 \fi}%
3141 \bbl@endcommands}%
3142 % == Line breaking: hyphenate.other.script ==
3143 \bbl@ifunset{bbl@hyots@languagename}{}%
3144 {\bbl@csarg\bbl@replace{hyots@languagename}{ }{,}%
3145 \bbl@csarg\bbl@foreach{hyots@languagename}{%
3146 \ifcase\bbl@engine
3147 \ifnum##1<257
3148 \global\lccode##1=##1\relax
3149 \fi
3150 \else
3151 \global\lccode##1=##1\relax
3152 \fi}}%
3153 % == Counters: maparabic ==
3154 % Native digits, if provided in ini (TeX level, xe and lua)
3155 \ifcase\bbl@engine\else
3156 \bbl@ifunset{bbl@dgnat@languagename}{}%
3157 {\expandafter\ifx\csname bbl@dgnat@languagename\endcsname\@empty\else
3158 \expandafter\expandafter\expandafter
3159 \bbl@setdigits\csname bbl@dgnat@languagename\endcsname
3160 \ifx\bbl@KVP@maparabic\@nil\else
3161 \ifx\bbl@latinarabic\@undefined
3162 \expandafter\let\expandafter\@arabic
3163 \csname bbl@counter@languagename\endcsname
3164 \else % ie, if layout=counters, which redefines \@arabic
3165 \expandafter\let\expandafter\bbl@latinarabic
3166 \csname bbl@counter@languagename\endcsname
3167 \fi
3168 \fi
3169 \fi}%
3170 \fi
3171 % == Counters: mapdigits ==
3172 % Native digits (lua level).
3173 \ifodd\bbl@engine
3174 \ifx\bbl@KVP@mapdigits\@nil\else
3175 \bbl@ifunset{bbl@dgnat@languagename}{}%
3176 {\RequirePackage{luatexbase}%
3177 \bbl@activate@preotf
3178 \directlua{
3179 Babel = Babel or {} %% -> presets in luababel
3180 Babel.digits_mapped = true
3181 Babel.digits = Babel.digits or {}
3182 Babel.digits[\the\localeid] =
3183 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3184 if not Babel.numbers then
3185 function Babel.numbers(head)
3186 local LOCALE = luatexbase.registernumber'bbl@attr@locale'

```

```

3187         local GLYPH = node.id'glyph'
3188         local inmath = false
3189         for item in node.traverse(head) do
3190             if not inmath and item.id == GLYPH then
3191                 local temp = node.get_attribute(item, LOCALE)
3192                 if Babel.digits[temp] then
3193                     local chr = item.char
3194                     if chr > 47 and chr < 58 then
3195                         item.char = Babel.digits[temp][chr-47]
3196                     end
3197                 end
3198             elseif item.id == node.id'math' then
3199                 inmath = (item.subtype == 0)
3200             end
3201         end
3202         return head
3203     end
3204 end
3205 }}%
3206 \fi
3207 \fi
3208 % == Counters: alph, Alph ==
3209 % What if extras<lang> contains a \babel@save\@alph? It won't be
3210 % restored correctly when exiting the language, so we ignore
3211 % this change with the \bbl@alph@saved trick.
3212 \ifx\bbl@KVP@alph\@nil\else
3213     \toks@\expandafter\expandafter\expandafter{%
3214         \csname extras\language\endcsname}%
3215     \bbl@exp{%
3216         \def\<extras\language>{%
3217             \let\\bbl@alph@saved\\@alph
3218             \the\toks@
3219             \let\\@alph\\bbl@alph@saved
3220             \\babel@save\\@alph
3221             \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
3222 \fi
3223 \ifx\bbl@KVP@Alph\@nil\else
3224     \toks@\expandafter\expandafter\expandafter{%
3225         \csname extras\language\endcsname}%
3226     \bbl@exp{%
3227         \def\<extras\language>{%
3228             \let\\bbl@Alph@saved\\@Alph
3229             \the\toks@
3230             \let\\@Alph\\bbl@Alph@saved
3231             \\babel@save\\@Alph
3232             \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3233 \fi
3234 % == require.babel in ini ==
3235 % To load or reload the babel-*.tex, if require.babel in ini
3236 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3237     \bbl@ifunset{\bbl@rtex@\language}%
3238     {\expandafter\ifx\csname bbl@rtex@\language\endcsname\@empty\else
3239         \let\BabelBeforeIni\@gobbletwo
3240         \chardef\atcatcode=\catcode`\@
3241         \catcode`\@=11\relax
3242         \bbl@input@texini{\bbl@cs{rtex@\language}}%
3243         \catcode`\@=\atcatcode
3244         \let\atcatcode\relax
3245     \fi}%

```

```

3246 \fi
3247 % == main ==
3248 \ifx\bbbl@KVP@main\@nil % Restore only if not 'main'
3249 \let\language\bbbl@savelangname
3250 \chardef\localeid\bbbl@savelocaleid\relax
3251 \fi}

```

Depending on whether or not the language exists, we define two macros.

```

3252 \def\bbbl@provide@new#1{%
3253 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3254 \namedef{extras#1}{}%
3255 \namedef{noextras#1}{}%
3256 \bbbl@startcommands*{#1}{captions}%
3257 \ifx\bbbl@KVP@captions\@nil % and also if import, implicit
3258 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
3259 \ifx##1\@empty\else
3260 \bbbl@exp{%
3261 \SetString\##1{%
3262 \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
3263 \expandafter\bbbl@tempb
3264 \fi}%
3265 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
3266 \else
3267 \ifx\bbbl@initoload\relax
3268 \bbbl@read@ini{\bbbl@KVP@captions}0% Here letters cat = 11
3269 \else
3270 \bbbl@read@ini{\bbbl@initoload}0% Here all letters cat = 11
3271 \fi
3272 \bbbl@after@ini
3273 \bbbl@savestrings
3274 \fi
3275 \StartBabelCommands*{#1}{date}%
3276 \ifx\bbbl@KVP@import\@nil
3277 \bbbl@exp{%
3278 \SetString\today{\bbbl@nocaption{today}{#1today}}}%
3279 \else
3280 \bbbl@savetoday
3281 \bbbl@savedate
3282 \fi
3283 \bbbl@endcommands
3284 \bbbl@load@basic{#1}%
3285 % == hyphenmins == (only if new)
3286 \bbbl@exp{%
3287 \gdef\<#1hyphenmins>{%
3288 {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
3289 {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}}}%
3290 % == hyphenrules ==
3291 \bbbl@provide@hyphens{#1}%
3292 % == frenchspacing == (only if new)
3293 \bbbl@ifunset{\bbbl@frspc@#1}{}%
3294 {\edef\bbbl@tempa{\bbbl@cl{frspc}}%
3295 \edef\bbbl@tempa{\expandafter\@car\bbbl@tempa\@nil}%
3296 \if u\bbbl@tempa % do nothing
3297 \else\if n\bbbl@tempa % non french
3298 \expandafter\bbbl@add\csname extras#1\endcsname{%
3299 \let\bbbl@elt\bbbl@fs@elt@i
3300 \bbbl@fs@chars}%
3301 \else\if y\bbbl@tempa % french
3302 \expandafter\bbbl@add\csname extras#1\endcsname{%

```

```

3303         \let\bbl@elt\bbl@fs@elt@ii
3304         \bbl@fs@chars}%
3305     \fi\fi\fi}%
3306 %
3307 \ifx\bbl@KVP@main\@nil\else
3308     \expandafter\main@language\expandafter{#1}%
3309 \fi}
3310 % A couple of macros used above, to avoid hashes #####...
3311 \def\bbl@fs@elt@i#1#2#3{%
3312     \ifnum\sfcode`#1=#2\relax
3313         \babel@savevariable{\sfcode`#1}%
3314         \sfcode`#1=#3\relax
3315     \fi}%
3316 \def\bbl@fs@elt@ii#1#2#3{%
3317     \ifnum\sfcode`#1=#3\relax
3318         \babel@savevariable{\sfcode`#1}%
3319         \sfcode`#1=#2\relax
3320     \fi}%
3321 %
3322 \def\bbl@provide@renew#1{%
3323     \ifx\bbl@KVP@captions\@nil\else
3324         \StartBabelCommands*{#1}{captions}%
3325         \bbl@read@ini{\bbl@KVP@captions}0%   Here all letters cat = 11
3326         \bbl@after@ini
3327         \bbl@savestrings
3328         \EndBabelCommands
3329     \fi
3330 \ifx\bbl@KVP@import\@nil\else
3331     \StartBabelCommands*{#1}{date}%
3332     \bbl@savetoday
3333     \bbl@savestate
3334     \EndBabelCommands
3335 \fi
3336 % == hyphenrules ==
3337 \bbl@provide@hyphens{#1}}
3338 % Load the basic parameters (ids, typography, counters, and a few
3339 % more), while captions and dates are left out. But it may happen some
3340 % data has been loaded before automatically, so we first discard the
3341 % saved values.
3342 \def\bbl@linebreak@export{%
3343     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3344     \bbl@exportkey{hyphr}{typography.hyphenrules}{h}%
3345     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3346     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3347     \bbl@exportkey{prehc}{typography.prehyphenchar}{h}%
3348     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{h}%
3349     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{h}%
3350     \bbl@exportkey{intsp}{typography.intraspace}{h}%
3351     \bbl@exportkey{chrng}{characters.ranges}{h}}
3352 \def\bbl@load@basic#1{%
3353     \bbl@ifunset{bbl@inidata@\language}\@nil{
3354         \getlocaleproperty\bbl@tempa{\language}{identification/load.level}%
3355         \ifcase\bbl@tempa\else
3356             \bbl@csarg\let{lname@\language}\relax
3357         \fi}%
3358     \bbl@ifunset{bbl@lname@#1}%
3359     {\def\BabelBeforeIni##1##2{%
3360         \begingroup
3361         \let\bbl@ini@captions@aux\@gobbletwo

```



```

3362 \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
3363 \bbl@read@ini{##1}0%
3364 \bbl@linebreak@export
3365 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3366 \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3367 \ifx\bbl@initoload\relax\endinput\fi
3368 \endgroup}%
3369 \begingroup % boxed, to avoid extra spaces:
3370 \ifx\bbl@initoload\relax
3371 \bbl@input@texini{#1}%
3372 \else
3373 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3374 \fi
3375 \endgroup}%
3376 {}

```

The hyphenrules option is handled with an auxiliary macro.

```

3377 \def\bbl@provide@hyphens#1{%
3378 \let\bbl@tempa\relax
3379 \ifx\bbl@KVP@hyphenrules\@nil\else
3380 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3381 \bbl@foreach\bbl@KVP@hyphenrules{%
3382 \ifx\bbl@tempa\relax % if not yet found
3383 \bbl@ifsamestring{##1}{+}%
3384 {\bbl@exp{\addlanguage<l@##1>}}}%
3385 }%
3386 \bbl@ifunset{l@##1}%
3387 }%
3388 {\bbl@exp{\let\bbl@tempa<l@##1>}}}%
3389 \fi}%
3390 \fi
3391 \ifx\bbl@tempa\relax % if no opt or no language in opt found
3392 \ifx\bbl@KVP@import\@nil
3393 \ifx\bbl@initoload\relax\else
3394 \bbl@exp{% and hyphenrules is not empty
3395 \bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3396 }%
3397 {\let\bbl@tempa<l@bbl@cl{hyphr}>}}}%
3398 \fi
3399 \else % if importing
3400 \bbl@exp{% and hyphenrules is not empty
3401 \bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3402 }%
3403 {\let\bbl@tempa<l@bbl@cl{hyphr}>}}}%
3404 \fi
3405 \fi
3406 \bbl@ifunset{bbl@tempa}% ie, relax or undefined
3407 {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
3408 {\bbl@exp{\adddialect<l@#1>\language}}}%
3409 }% so, l@<lang> is ok - nothing to do
3410 {\bbl@exp{\adddialect<l@#1>\bbl@tempa}}}% found in opt list or ini
3411

```

The reader of ini files. There are 3 possible cases: a section name (in the form [. . .]), a comment (starting with ;) and a key/value pair.

```

3412 \ifx\bbl@readstream\@undefined
3413 \csname newread\endcsname\bbl@readstream
3414 \fi
3415 \def\bbl@input@texini#1{%

```

```

3416 \bbl@bsphack
3417 \bbl@exp{%
3418 \catcode`\\%=14 \catcode`\\|=0
3419 \catcode`\\={1 \catcode`\\}=2
3420 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
3421 \catcode`\\%=the\catcode`\%\relax
3422 \catcode`\\|=the\catcode`\|\relax
3423 \catcode`\\={the\catcode`\{\relax
3424 \catcode`\\}=the\catcode`\}\relax}%
3425 \bbl@esphack}
3426 \def\bbl@inipreread#1=#2\@{%
3427 \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3428 \bbl@trim\toks@{#2}%
3429 % Move trims here ??
3430 \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
3431 {\bbl@exp{%
3432 \\\g@addto@macro\\bbl@inidata{%
3433 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3434 \expandafter\bbl@inireader\bbl@tempa=#2\@}%
3435 }%
3436 \def\bbl@fetch@ini#1#2{%
3437 \bbl@exp{\def\\bbl@inidata{%
3438 \\\bbl@elt{identification}{tag.ini}{#1}%
3439 \\\bbl@elt{identification}{load.level}{#2}}}%
3440 \openin\bbl@readstream=babel-#1.ini
3441 \ifeof\bbl@readstream
3442 \bbl@error
3443 {There is no ini file for the requested language\\%
3444 (#1). Perhaps you misspelled it or your installation\\%
3445 is not complete.}%
3446 {Fix the name or reinstall babel.}%
3447 \else
3448 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3449 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3450 \bbl@info{Importing
3451 \ifcase#2 \or font and identification \or basic \fi
3452 data for \language\name\\%
3453 from babel-#1.ini. Reported}%
3454 \loop
3455 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3456 \endlinechar\m@ne
3457 \read\bbl@readstream to \bbl@line
3458 \endlinechar`^^M
3459 \ifx\bbl@line\empty\else
3460 \expandafter\bbl@inline\bbl@line\bbl@inline
3461 \fi
3462 \repeat
3463 \fi}
3464 \def\bbl@read@ini#1#2{%
3465 \bbl@csarg\xdef{lini@\language\name}{#1}%
3466 \let\bbl@section\empty
3467 \let\bbl@savestrings\empty
3468 \let\bbl@savetoday\empty
3469 \let\bbl@savestate\empty
3470 \let\bbl@inireader\bbl@iniskip
3471 \bbl@fetch@ini{#1}{#2}%
3472 \bbl@foreach\bbl@renewlist{%
3473 \bbl@ifunset{bbl@renew@##1}{\bbl@inisec[##1]\@}}%
3474 \global\let\bbl@renewlist\empty

```

```

3475 % Ends last section. See \bbl@inisec
3476 \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
3477 \bbl@cs{renew@\bbl@section}%
3478 \global\bbl@csarg\let{renew@\bbl@section}\relax
3479 \bbl@cs{secpost@\bbl@section}%
3480 \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3481 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
3482 \bbl@to\global\bbl@ini@loaded}
3483 \def\bbl@iniline#1\bbl@iniline{%
3484 \ifnextchar[\bbl@inisec{\ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@@}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start. By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

3485 \def\bbl@iniskip#1\@@{% if starts with ;
3486 \def\bbl@inisec[#1]#2\@@{% if starts with opening bracket
3487 \def\bbl@elt##1##2{%
3488 \expandafter\toks@\expandafter{%
3489 \expandafter{\bbl@section}{##1}{##2}}%
3490 \bbl@exp{%
3491 \g@addto@macro\bbl@inidata{\bbl@elt\the\toks@}}%
3492 \bbl@inireader##1=##2\@@}%
3493 \bbl@cs{renew@\bbl@section}%
3494 \global\bbl@csarg\let{renew@\bbl@section}\relax
3495 \bbl@cs{secpost@\bbl@section}%
3496 % The previous code belongs to the previous section.
3497 % -----
3498 % Now start the current one.
3499 \in@{=date.}{#1}%
3500 \ifin@
3501 \lowercase{\def\bbl@tempa{#1=}}%
3502 \bbl@replace\bbl@tempa{=date.gregorian.}{}%
3503 \bbl@replace\bbl@tempa{=date.}{}%
3504 \in@{.licr=}{#1}%
3505 \ifin@
3506 \ifcase\bbl@engine
3507 \bbl@replace\bbl@tempa{.licr=}{}%
3508 \else
3509 \let\bbl@tempa\relax
3510 \fi
3511 \fi
3512 \ifx\bbl@tempa\relax\else
3513 \bbl@replace\bbl@tempa{=}{}%
3514 \bbl@exp{%
3515 \def<\bbl@inikv@#1>####1=####2\@@{%
3516 \bbl@inidate####1...\relax{####2}{\bbl@tempa}}%
3517 \fi
3518 \fi
3519 \def\bbl@section{#1}%
3520 \def\bbl@elt##1##2{%
3521 \@namedef{\bbl@KVP@#1/#1}{}}%
3522 \bbl@cs{renew@#1}%
3523 \bbl@cs{secpre@#1}% pre-section `hook'
3524 \bbl@ifunset{\bbl@inikv@#1}%
3525 {\let\bbl@inireader\bbl@iniskip}%
3526 {\bbl@exp{\let\bbl@inireader<\bbl@inikv@#1>}}
3527 \let\bbl@renewlist\@empty
3528 \def\bbl@renewinikey#1/#2\@@#3{%

```

```

3529 \bbl@ifunset{\bbl@renew@#1}%
3530 {\bbl@add@list\bbl@renewlist{#1}}%
3531 {}%
3532 \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}

```

Reads a key=val line and stores the trimmed val in \bbl@kv@<section>.<key>.

```

3533 \def\bbl@inikv#1=#2\@{\%      key=value
3534 \bbl@trim@def\bbl@tempa{#1}%
3535 \bbl@trim\toks@{#2}%
3536 \bbl@csarg\edef{\kv@\bbl@section.\bbl@tempa}{\the\toks@}}

```

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

```

3537 \def\bbl@exportkey#1#2#3{%
3538 \bbl@ifunset{\bbl@kv@#2}%
3539 {\bbl@csarg\gdef{#1@\language}\{#3}}%
3540 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
3541 \bbl@csarg\gdef{#1@\language}\{#3}}%
3542 \else
3543 \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3544 \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@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3545 \def\bbl@iniwarning#1{%
3546 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3547 {\bbl@warning{%
3548 From babel-\bbl@cs{lini@\language}.ini:\%
3549 \bbl@cs{@kv@identification.warning#1}\%
3550 Reported }}}
3551 %
3552 \let\bbl@inikv@identification\bbl@inikv
3553 \def\bbl@secpost@identification{%
3554 \bbl@iniwarning}%
3555 \ifcase\bbl@engine
3556 \bbl@iniwarning{.pdflatex}%
3557 \or
3558 \bbl@iniwarning{.lualatex}%
3559 \or
3560 \bbl@iniwarning{.xelatex}%
3561 \fi%
3562 \bbl@exportkey{elname}{identification.name.english}{}%
3563 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3564 {\csname\bbl@elname@\language\endcsname}}%
3565 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
3566 \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
3567 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3568 \bbl@exportkey{esname}{identification.script.name}{}%
3569 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3570 {\csname\bbl@esname@\language\endcsname}}%
3571 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
3572 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3573 \ifbbl@bcptoname
3574 \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
3575 \fi}

```

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

```

3576 \let\bbl@inikv@typography\bbl@inikv
3577 \let\bbl@inikv@characters\bbl@inikv
3578 \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’.

```

3579 \def\bbl@inikv@counters#1=#2\@@{%
3580   \bbl@ifsamestring{#1}{digits}%
3581   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3582     decimal digits}%
3583     {Use another name.}}%
3584   }%
3585 \def\bbl@tempc{#1}%
3586 \bbl@trim@def{\bbl@tempb*}{#2}%
3587 \in@{.1$}{#1$}%
3588 \ifin@
3589   \bbl@replace\bbl@tempc{.1}{}%
3590   \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}\bbl@tempc @\language}%
3591   \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3592 \fi
3593 \in@{.F.}{#1}%
3594 \ifin@else\in@{.S.}{#1}\fi
3595 \ifin@
3596   \bbl@csarg\protected@xdef{cntr@#1@\language}\bbl@tempb*}%
3597 \else
3598   \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3599   \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3600   \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3601   \fi}
3602 \def\bbl@after@ini{%
3603   \bbl@linebreak@export
3604   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3605   \bbl@exportkey{rqtex}{identification.require.babel}{}%
3606   \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3607   \bbl@toglobal\bbl@savetoday
3608   \bbl@toglobal\bbl@savestate}

```

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.

```

3609 \ifcase\bbl@engine
3610   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3611     \bbl@ini@captions@aux{#1}{#2}}
3612 \else
3613   \def\bbl@inikv@captions#1=#2\@@{%
3614     \bbl@ini@captions@aux{#1}{#2}}
3615 \fi

```

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

```

3616 \def\bbl@ini@captions@aux#1#2{%
3617   \bbl@trim@def\bbl@tempa{#1}%
3618   \bbl@xin@{.template}{\bbl@tempa}%
3619   \ifin@
3620     \bbl@replace\bbl@tempa{.template}{}%
3621     \def\bbl@toreplace{#2}%
3622     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3623     \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3624     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%

```

```

3625 \bbl@replace\bbl@toreplace{}}{name\endcsname{}}}%
3626 \bbl@replace\bbl@toreplace{}}{\endcsname{}}}%
3627 \bbl@xin@{,\bbl@tempa,}{,chapter,}%
3628 \ifin@
3629 \bbl@patchchapter
3630 \global\bbl@csarg\let{chapfmt@\language}\bbl@toreplace
3631 \fi
3632 \bbl@xin@{,\bbl@tempa,}{,appendix,}%
3633 \ifin@
3634 \bbl@patchchapter
3635 \global\bbl@csarg\let{appxfmt@\language}\bbl@toreplace
3636 \fi
3637 \bbl@xin@{,\bbl@tempa,}{,part,}%
3638 \ifin@
3639 \bbl@patchpart
3640 \global\bbl@csarg\let{partfmt@\language}\bbl@toreplace
3641 \fi
3642 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3643 \ifin@
3644 \toks@\expandafter{\bbl@toreplace}%
3645 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3646 \fi
3647 \else
3648 \bbl@ifblank{#2}%
3649 {\bbl@exp{%
3650 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3651 {\bbl@trim\toks@{#2}}}%
3652 \bbl@exp{%
3653 \bbl@add\bbl@savestrings{%
3654 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3655 \toks@\expandafter{\bbl@captionslist}%
3656 \bbl@exp{\in{\<\bbl@tempa name>}{\the\toks@}}}%
3657 \ifin@else
3658 \bbl@exp{%
3659 \bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3660 \bbl@to\global\<\bbl@extracaps@\language>}}%
3661 \fi
3662 \fi}

```

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

```

3663 \def\bbl@list@the{%
3664 part,chapter,section,subsection,subsubsection,paragraph,%
3665 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3666 table,page,footnote,mpfootnote,mpfn}
3667 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3668 \bbl@ifunset{\bbl@map@#1@\language}%
3669 {\nameuse{#1}}%
3670 {\nameuse{\bbl@map@#1@\language}}}
3671 \def\bbl@inikv@labels#1=#2\@@{%
3672 \in@{.map}{#1}}%
3673 \ifin@
3674 \ifx\bbl@KVP@labels\@nil\else
3675 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3676 \ifin@
3677 \def\bbl@tempc{#1}%
3678 \bbl@replace\bbl@tempc{.map}{}%
3679 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3680 \bbl@exp{%

```

```

3681 \gdef\<bbl@map@bbl@tempc @\language>%
3682 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3683 \bbl@foreach\bbl@list@the{%
3684 \bbl@ifunset{the##1}{}%
3685 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3686 \bbl@exp{%
3687 \\bbl@sreplace\<the##1>%
3688 {\<bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}%
3689 \\bbl@sreplace\<the##1>%
3690 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3691 \expandafter\ifx\cname the##1\endcsname\bbl@tempd\else
3692 \toks@ \expandafter\expandafter\expandafter{%
3693 \cname the##1\endcsname}%
3694 \expandafter\xdef\cname the##1\endcsname{{\the\toks@}}%
3695 \fi}}%
3696 \fi
3697 \fi
3698 %
3699 \else
3700 %
3701 % The following code is still under study. You can test it and make
3702 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3703 % language dependent.
3704 \in@{enumerate.}{#1}%
3705 \ifin@
3706 \def\bbl@tempa{#1}%
3707 \bbl@replace\bbl@tempa{enumerate.}{}%
3708 \def\bbl@toreplace{#2}%
3709 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3710 \bbl@replace\bbl@toreplace{[ ]}{\cname the}%
3711 \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}%
3712 \toks@ \expandafter{\bbl@toreplace}%
3713 \bbl@exp{%
3714 \\bbl@add\<extras\language>{%
3715 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3716 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3717 \\bbl@toglobal\<extras\language>}%
3718 \fi
3719 \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.

```

3720 \def\bbl@chapttype{chap}
3721 \ifx\@makechapterhead\@undefined
3722 \let\bbl@patchchapter\relax
3723 \else\ifx\thechapter\@undefined
3724 \let\bbl@patchchapter\relax
3725 \else\ifx\ps@headings\@undefined
3726 \let\bbl@patchchapter\relax
3727 \else
3728 \def\bbl@patchchapter{%
3729 \global\let\bbl@patchchapter\relax
3730 \bbl@add\appendix{\def\bbl@chapttype{appx}}% Not harmful, I hope
3731 \bbl@toglobal\appendix
3732 \bbl@sreplace\ps@headings
3733 {\@chapapp\ thechapter}%
3734 {\bbl@chapterformat}%

```

```

3735 \bbl@toglobal\ps@headings
3736 \bbl@sreplace\chaptermark
3737 {\@chapapp\ thechapter}%
3738 {\bbl@chapterformat}%
3739 \bbl@toglobal\chaptermark
3740 \bbl@sreplace\makechapterhead
3741 {\@chapapp\space\thechapter}%
3742 {\bbl@chapterformat}%
3743 \bbl@toglobal\@makechapterhead
3744 \gdef\bbl@chapterformat{%
3745 \bbl@ifunset{\bbl\bbl@chapttype fmt@\language}%
3746 {\@chapapp\space\thechapter}
3747 {\@nameuse{\bbl\bbl@chapttype fmt@\language}}}}
3748 \fi\fi\fi
3749 \ifx\@part\undefined
3750 \let\bbl@patchpart\relax
3751 \else
3752 \def\bbl@patchpart{%
3753 \global\let\bbl@patchpart\relax
3754 \bbl@sreplace\@part
3755 {\partname\nobreakspace\thepart}%
3756 {\bbl@partformat}%
3757 \bbl@toglobal\@part
3758 \gdef\bbl@partformat{%
3759 \bbl@ifunset{\bbl@partfmt@\language}%
3760 {\partname\nobreakspace\thepart}
3761 {\@nameuse{\bbl@partfmt@\language}}}}
3762 \fi

Date. TODO. Document

3763 % Arguments are _not_ protected.
3764 \let\bbl@calendar\@empty
3765 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3766 \def\bbl@localedate#1#2#3#4{%
3767 \begingroup
3768 \ifx\@empty#1\@empty\else
3769 \let\bbl@ld@calendar\@empty
3770 \let\bbl@ld@variant\@empty
3771 \edef\bbl@tempa{\zap@space#1 \@empty}%
3772 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3773 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3774 \edef\bbl@calendar{%
3775 \bbl@ld@calendar
3776 \ifx\bbl@ld@variant\@empty\else
3777 .\bbl@ld@variant
3778 \fi}%
3779 \bbl@replace\bbl@calendar{gregorian}{}}%
3780 \fi
3781 \bbl@cased
3782 {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3783 \endgroup}
3784 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3785 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3786 \bbl@trim@def\bbl@tempa{#1.#2}%
3787 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3788 {\bbl@trim@def\bbl@tempa{#3}%
3789 \bbl@trim\toks@{#5}%
3790 \@temptokena\expandafter{\bbl@savedate}%
3791 \bbl@exp{% Reverse order - in ini last wins

```



```

3792 \def\\bbl@savestate{%
3793   \\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}%
3794   \the\temptokena}}}%
3795 {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3796   {\lowercase{\def\bbl@tempb{#6}}}%
3797   \bbl@trim@def\bbl@toreplace{#5}%
3798   \bbl@TG@date
3799   \bbl@ifunset{\bbl@date@\language @}%
3800   {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3801   % TODO. Move to a better place.
3802   \bbl@exp{%
3803     \gdef\<\language date>\protect\<\language date >}%
3804     \gdef\<\language date >####1####2####3{%
3805       \\bbl@usedategroupttrue
3806       \<bbl@ensure@\language >{%
3807         \\localedate{####1}{####2}{####3}}}%
3808       \\bbl@add\\bbl@savetoday{%
3809         \\SetString\\today{%
3810           \<\language date>%
3811           {\the\year}{\the\month}{\the\day}}}%
3812       }%
3813   \ifx\bbl@tempb\@empty\else
3814     \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3815   \fi}%
3816   {}}}%

```

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.

```

3817 \let\bbl@calendar\@empty
3818 \newcommand\BabelDateSpace{\nobreakspace}
3819 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3820 \newcommand\BabelDated[1]{\number#1}
3821 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3822 \newcommand\BabelDateM[1]{\number#1}
3823 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3824 \newcommand\BabelDateMMM[1]{%
3825   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3826 \newcommand\BabelDatey[1]{\number#1}%
3827 \newcommand\BabelDateyy[1]{%
3828   \ifnum#1<10 0\number#1 %
3829   \else\ifnum#1<100 \number#1 %
3830   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3831   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3832   \else
3833     \bbl@error
3834     {Currently two-digit years are restricted to the\
3835     range 0-9999.}%
3836     {There is little you can do. Sorry.}%
3837   \fi\fi\fi\fi}}
3838 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3839 \def\bbl@replace@finish@iii#1{%
3840   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3841 \def\bbl@TG@date{%
3842   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3843   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3844   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3845   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3846   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%

```

```

3847 \bbl@replace\bbl@toreplace{[MM]}\BabelDateMM{####2}%
3848 \bbl@replace\bbl@toreplace{[MMMM]}\BabelDateMMMM{####2}%
3849 \bbl@replace\bbl@toreplace{[y]}\BabelDatey{####1}%
3850 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{####1}%
3851 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}%
3852 \bbl@replace\bbl@toreplace{[y]}\bbl@datecncr[####1|}%
3853 \bbl@replace\bbl@toreplace{[m]}\bbl@datecncr[####2|}%
3854 \bbl@replace\bbl@toreplace{[d]}\bbl@datecncr[####3|}%
3855 % Note after \bbl@replace \toks@ contains the resulting string.
3856 % TODO - Using this implicit behavior doesn't seem a good idea.
3857 \bbl@replace@finish@iii\bbl@toreplace}
3858 \def\bbl@datecncr{\expandafter\bbl@xdatecncr\expandafter}
3859 \def\bbl@xdatecncr[#1|#2]{\localenumeral{#2}{#1}}

```

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

```

3860 \def\bbl@provide@lsys#1{%
3861   \bbl@ifunset{bbl@lname@#1}%
3862     {\bbl@ini@basic{#1}}%
3863   }%
3864   \bbl@csarg\let{lsys@#1}\empty
3865   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3866   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3867   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3868   \bbl@ifunset{bbl@lname@#1}{}%
3869   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3870   \ifcase\bbl@engine\or\or
3871     \bbl@ifunset{bbl@prehc@#1}{}%
3872     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3873       }%
3874     {\ifx\bbl@xenohyph\undefined
3875       \let\bbl@xenohyph\bbl@xenohyph@d
3876       \ifx\AtBeginDocument\@notprerr
3877         \expandafter\@secondoftwo % to execute right now
3878       \fi
3879       \AtBeginDocument{%
3880         \expandafter\bbl@add
3881         \csname selectfont \endcsname{\bbl@xenohyph}%
3882         \expandafter\selectlanguage\expandafter{\languagename}%
3883         \expandafter\bbl@tglobal\csname selectfont \endcsname}%
3884       \fi}%
3885   \fi
3886   \bbl@csarg\bbl@tglobal{lsys@#1}}
3887 \def\bbl@xenohyph@d{%
3888   \bbl@ifset{bbl@prehc@languagename}%
3889     {\ifnum\hyphenchar\font=\defaultshyphenchar
3890       \iffontchar\font\bbl@c1{prehc}\relax
3891       \hyphenchar\font\bbl@c1{prehc}\relax
3892     \else\iffontchar\font"200B
3893       \hyphenchar\font"200B
3894     \else
3895       \bbl@warning
3896       {Neither 0 nor ZERO WIDTH SPACE are available\\%
3897        in the current font, and therefore the hyphen\\%
3898        will be printed. Try changing the fontspec's\\%
3899        'HyphenChar' to another value, but be aware\\%
3900        this setting is not safe (see the manual)}%
3901       \hyphenchar\font\defaultshyphenchar
3902     \fi\fi

```



```

3949 \<ifcase>###1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3950 \else
3951 \toks@\expandafter{\the\toks@\or #1}%
3952 \expandafter\bbl@buildifcase
3953 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as a special case, for a fixed form (see babel-he.ini, for example).

```

3954 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\language}{#2}}
3955 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3956 \newcommand\localecounter[2]{%
3957 \expandafter\bbl@localecntr
3958 \expandafter{\number\csname c@#2\endcsname}{#1}}
3959 \def\bbl@alphnumeral#1#2{%
3960 \expandafter\bbl@alphnumeral@i\number#2 76543210\@{#1}}
3961 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@#9{%
3962 \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
3963 \bbl@alphnumeral@ii{#9}000000#1\or
3964 \bbl@alphnumeral@ii{#9}00000#1#2\or
3965 \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3966 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3967 \bbl@alphnum@invalid{>9999}%
3968 \fi}
3969 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3970 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3971 {\bbl@cs{cntr@#1.4@\language}{#5}
3972 \bbl@cs{cntr@#1.3@\language}{#6}
3973 \bbl@cs{cntr@#1.2@\language}{#7}
3974 \bbl@cs{cntr@#1.1@\language}{#8}
3975 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3976 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3977 {\bbl@cs{cntr@#1.S.321@\language}}%
3978 \fi}%
3979 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}
3980 \def\bbl@alphnum@invalid#1{%
3981 \bbl@error{Alphabetic numeral too large (#1)}%
3982 {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.

```

3983 \newcommand\localeinfo[1]{%
3984 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
3985 {\bbl@error{I've found no info for the current locale.\%
3986 The corresponding ini file has not been loaded\%
3987 Perhaps it doesn't exist}%
3988 {See the manual for details.}}%
3989 {\bbl@cs{\csname bbl@info@#1\endcsname @\language}}}
3990 % \@namedef{bbl@info@name.locale}{lname}
3991 \@namedef{bbl@info@tag.ini}{lini}
3992 \@namedef{bbl@info@name.english}{elname}
3993 \@namedef{bbl@info@name.opentype}{lname}
3994 \@namedef{bbl@info@tag.bcp47}{tbc}
3995 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3996 \@namedef{bbl@info@tag.opentype}{lotf}

```

```

3997 \@namedef{bbl@info@script.name}{esname}
3998 \@namedef{bbl@info@script.name.opentype}{sname}
3999 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4000 \@namedef{bbl@info@script.tag.opentype}{sotf}
4001 \let\bbl@ensureinfo\@gobble
4002 \newcommand\BabelEnsureInfo{%
4003   \ifx\InputIfFileExists\undefined\else
4004     \def\bbl@ensureinfo##1{%
4005       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}}%
4006   \fi
4007   \bbl@foreach\bbl@loaded{%
4008     \def\language{##1}%
4009     \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`.

```

4010 \newcommand\getlocaleproperty{%
4011   \@ifstar\bbl@getproperty@s\bbl@getproperty@x%
4012   \def\bbl@getproperty@s#1#2#3{%
4013     \let#1\relax
4014     \def\bbl@elt##1##2##3{%
4015       \bbl@ifsamestring{##1/##2}{#3}%
4016       {\providecommand#1{##3}%
4017       \def\bbl@elt####1####2####3{}}}%
4018     {}}%
4019     \bbl@cs{inidata@#2}}%
4020   \def\bbl@getproperty@x#1#2#3{%
4021     \bbl@getproperty@s{#1}{#2}{#3}%
4022     \ifx#1\relax
4023       \bbl@error
4024         {Unknown key for locale '#2':\%
4025         #3\%
4026         \string#1 will be set to \relax}%
4027       {Perhaps you misspelled it.}%
4028     \fi}
4029   \let\bbl@ini@loaded\@empty
4030   \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel behavior

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

```

4031 \newcommand\babeladjust[1]{% TODO. Error handling.
4032   \bbl@forkv{#1}{%
4033     \bbl@ifunset{bbl@ADJ@##1@##2}%
4034     {\bbl@cs{ADJ@##1}{##2}}%
4035     {\bbl@cs{ADJ@##1@##2}}}
4036 %
4037 \def\bbl@adjust@lua#1#2{%
4038   \ifvmode
4039     \ifnum\currentgrouplevel=\z@
4040       \directlua{ Babel.#2 }%
4041       \expandafter\expandafter\expandafter\@gobble
4042     \fi
4043   \fi
4044   {\bbl@error % The error is gobbled if everything went ok.
4045     {Currently, #1 related features can be adjusted only\%

```

```

4046         in the main vertical list.}%
4047         {Maybe things change in the future, but this is what it is.}}
4048 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4049   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4050 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4051   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4052 \@namedef{bbl@ADJ@bidi.text@on}{%
4053   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4054 \@namedef{bbl@ADJ@bidi.text@off}{%
4055   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4056 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4057   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4058 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4059   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4060 %
4061 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4062   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4063 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4064   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4065 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4066   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4067 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4068   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4069 %
4070 \def\bbl@adjust@layout#1{%
4071   \ifvmode
4072     #1%
4073     \expandafter\@gobble
4074   \fi
4075   {\bbl@error   % The error is gobbled if everything went ok.
4076     {Currently, layout related features can be adjusted only\\%
4077       in vertical mode.}%
4078     {Maybe things change in the future, but this is what it is.}}
4079 \@namedef{bbl@ADJ@layout.tabular@on}{%
4080   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4081 \@namedef{bbl@ADJ@layout.tabular@off}{%
4082   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4083 \@namedef{bbl@ADJ@layout.lists@on}{%
4084   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4085 \@namedef{bbl@ADJ@layout.lists@off}{%
4086   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4087 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4088   \bbl@activateposthyphen}
4089 %
4090 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4091   \bbl@bcpallowedtrue}
4092 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4093   \bbl@bcpallowedfalse}
4094 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4095   \def\bbl@bcp@prefix{#1}}
4096 \def\bbl@bcp@prefix{bcp47-}
4097 \@namedef{bbl@ADJ@autoload.options}#1{%
4098   \def\bbl@autoload@options{#1}}
4099 \let\bbl@autoload@bcptoptions\@empty
4100 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4101   \def\bbl@autoload@bcptoptions{#1}}
4102 \newif\ifbbl@bcptoname
4103 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4104   \bbl@bcptonametrue

```

```

4105 \BabelEnsureInfo}
4106 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4107 \bbl@bcptonamefalse}
4108% TODO: use babel name, override
4109%
4110% As the final task, load the code for lua.
4111%
4112 \ifx\directlua\@undefined\else
4113 \ifx\bbl@luapatterns\@undefined
4114 \input luababel.def
4115 \fi
4116 \fi
4117 </core>

```

A proxy file for switch.def

```

4118 <*kernel>
4119 \let\bbl@onlyswitch\@empty
4120 \input babel.def
4121 \let\bbl@onlyswitch\@undefined
4122 </kernel>
4123 <*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.

```

4124 <<Make sure ProvidesFile is defined>>
4125 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
4126 \xdef\bbl@format{\jobname}
4127 \def\bbl@version{<<version>>}
4128 \def\bbl@date{<<date>>}
4129 \ifx\AtBeginDocument\@undefined
4130 \def\@empty{}
4131 \let\orig@dump\dump
4132 \def\dump{%
4133 \ifx\@ztryfc\@undefined
4134 \else
4135 \toks0=\expandafter{\@preamblecmds}%
4136 \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4137 \def\@begindocumenthook{}%
4138 \fi
4139 \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4140 \fi
4141 <<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.

```

4142 \def\process@line#1#2 #3 #4 {%
4143   \ifx=#1%
4144     \process@synonym{#2}%
4145   \else
4146     \process@language{#1#2}{#3}{#4}%
4147   \fi
4148   \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.

```

4149 \toks@{}
4150 \def\bbl@languages{}

```

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

Otherwise the name will be a synonym for the language loaded last.

We also need to copy the hyphenmin parameters for the synonym.

```

4151 \def\process@synonym#1{%
4152   \ifnum\last@language=\m@ne
4153     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4154   \else
4155     \expandafter\chardef\csname l@#1\endcsname\last@language
4156     \wlog{\string\l@#1=\string\language\the\last@language}%
4157     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4158       \csname\language\hyphenmins\endcsname
4159     \let\bbl@elt\relax
4160     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4161   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

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

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

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

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

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not

empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```

4162 \def\process@language#1#2#3{%
4163   \expandafter\addlanguage\csname l@#1\endcsname
4164   \expandafter\language\csname l@#1\endcsname
4165   \edef\language#1{%
4166     \bbl@hook@everylanguage{#1}%
4167     % > luatex
4168     \bbl@get@enc#1::\@@@
4169     \begingroup
4170       \lefthyphenmin\m@ne
4171       \bbl@hook@loadpatterns{#2}%
4172       % > luatex
4173       \ifnum\lefthyphenmin=\m@ne
4174       \else
4175         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4176           \the\lefthyphenmin\the\righthyphenmin}%
4177       \fi
4178     \endgroup
4179     \def\bbl@tempa{#3}%
4180     \ifx\bbl@tempa\@empty\else
4181       \bbl@hook@loadexceptions{#3}%
4182       % > luatex
4183     \fi
4184     \let\bbl@elt\relax
4185     \edef\bbl@languages{%
4186       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4187     \ifnum\the\language=\z@
4188       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4189         \set@hyphenmins\tw@\thr@@\relax
4190       \else
4191         \expandafter\expandafter\expandafter\set@hyphenmins
4192         \csname #1hyphenmins\endcsname
4193       \fi
4194       \the\toks@
4195       \toks@{}%
4196     \fi}

```

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

```

4197 \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.

```

4198 \def\bbl@hook@everylanguage#1{}
4199 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4200 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4201 \def\bbl@hook@loadkernel#1{%
4202   \def\addlanguage{\csname newlanguage\endcsname}%
4203   \def\adddialect##1##2{%
4204     \global\chardef##1##2\relax

```

```

4205 \wlog{\string##1 = a dialect from \string\language##2}%
4206 \def\iflanguage##1{%
4207 \expandafter\ifx\csname l@##1\endcsname\relax
4208 \@nolanerr{##1}%
4209 \else
4210 \ifnum\csname l@##1\endcsname=\language
4211 \expandafter\expandafter\expandafter\@firstoftwo
4212 \else
4213 \expandafter\expandafter\expandafter\@secondoftwo
4214 \fi
4215 \fi}%
4216 \def\providehyphenmins##1##2{%
4217 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4218 \@namedef{##1hyphenmins}{##2}%
4219 \fi}%
4220 \def\set@hyphenmins##1##2{%
4221 \lefthyphenmin##1\relax
4222 \righthyphenmin##2\relax}%
4223 \def\selectlanguage{%
4224 \errhelp{Selecting a language requires a package supporting it}%
4225 \errmessage{Not loaded}}%
4226 \let\foreignlanguage\selectlanguage
4227 \let\otherlanguage\selectlanguage
4228 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4229 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4230 \def\setlocale{%
4231 \errhelp{Find an armchair, sit down and wait}%
4232 \errmessage{Not yet available}}%
4233 \let\uselocale\setlocale
4234 \let\locale\setlocale
4235 \let\selectlocale\setlocale
4236 \let\localename\setlocale
4237 \let\textlocale\setlocale
4238 \let\textlanguage\setlocale
4239 \let\languagetext\setlocale}
4240 \begingroup
4241 \def\AddBabelHook#1#2{%
4242 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4243 \def\next{\toks1}%
4244 \else
4245 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4246 \fi
4247 \next}
4248 \ifx\directlua\undefined
4249 \ifx\XeTeXinputencoding\undefined\else
4250 \input xebabel.def
4251 \fi
4252 \else
4253 \input luababel.def
4254 \fi
4255 \openin1 = babel-\bbl@format.cfg
4256 \ifeof1
4257 \else
4258 \input babel-\bbl@format.cfg\relax
4259 \fi
4260 \closein1
4261 \endgroup
4262 \bbl@hook@loadkernel{switch.def}

```

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

```
4263 \openin1 = language.dat
```

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

```
4264 \def\languagename{english}%
4265 \ifeof1
4266 \message{I couldn't find the file language.dat,\space
4267         I will try the file hyphen.tex}
4268 \input hyphen.tex\relax
4269 \chardef\l@english\z@
4270 \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`.

```
4271 \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.

```
4272 \loop
4273 \endlinechar\m@ne
4274 \read1 to \bbl@line
4275 \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.

```
4276 \if T\ifeof1F\fi T\relax
4277 \ifx\bbl@line\@empty\else
4278 \edef\bbl@line{\bbl@line\space\space\space}%
4279 \expandafter\process@line\bbl@line\relax
4280 \fi
4281 \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.

```
4282 \begingroup
4283 \def\bbl@elt#1#2#3#4{%
4284 \global\language=#2\relax
4285 \gdef\languagename{#1}%
4286 \def\bbl@elt##1##2##3##4{}}%
4287 \bbl@languages
4288 \endgroup
4289 \fi
4290 \closein1
```

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

```
4291 \if/\the\toks@/\else
4292 \errhelp{language.dat loads no language, only synonyms}
4293 \errmessage{Orphan language synonym}
4294 \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.

```

4295 \let\bbl@line\@undefined
4296 \let\process@line\@undefined
4297 \let\process@synonym\@undefined
4298 \let\process@language\@undefined
4299 \let\bbl@get@enc\@undefined
4300 \let\bbl@hyph@enc\@undefined
4301 \let\bbl@tempa\@undefined
4302 \let\bbl@hook@loadkernel\@undefined
4303 \let\bbl@hook@everylanguage\@undefined
4304 \let\bbl@hook@loadpatterns\@undefined
4305 \let\bbl@hook@loadexceptions\@undefined
4306 \let\patterns\@undefined

```

Here the code for `iniTEX` ends.

12 Font handling with fontspec

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

```

4307 <<*More package options>> ≡
4308 \chardef\bbl@bidimode\z@
4309 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4310 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4311 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4312 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4313 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4314 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4315 <</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.

```

4316 <<*Font selection>> ≡
4317 \bbl@trace{Font handling with fontspec}
4318 \ifx\ExplSyntaxOn\@undefined\else
4319   \ExplSyntaxOn
4320   \catcode`\ =10
4321   \def\bbl@loadfontspec{%
4322     \usepackage{fontspec}%
4323     \expandafter
4324     \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4325       Font '\l_fontspec_fontname_tl' is using the\\%
4326       default features for language '##1'.\\%
4327       That's usually fine, because many languages\\%
4328       require no specific features, but if the output is\\%
4329       not as expected, consider selecting another font.}
4330     \expandafter
4331     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%

```

```

4332     Font '\l_fontspec_fontname_tl' is using the\\%
4333     default features for script '##2'.\\%
4334     That's not always wrong, but if the output is\\%
4335     not as expected, consider selecting another font.}}
4336 \ExplSyntaxOff
4337 \fi
4338 \@onlypreamble\babelfont
4339 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4340 \bbl@foreach{#1}{\%
4341 \expandafter\ifx\csname date##1\endcsname\relax
4342 \IfFileExists{babel-##1.tex}{\%
4343 {\babelprovide{##1}}\%
4344 }{\%
4345 \fi}\%
4346 \edef\bbl@tempa{#1}\%
4347 \def\bbl@tempb{#2}\% Used by \bbl@bblfont
4348 \ifx\fontspec\@undefined
4349 \bbl@loadfontspec
4350 \fi
4351 \EnableBabelHook{babel-fontspec}\% Just calls \bbl@switchfont
4352 \bbl@bblfont}
4353 \newcommand\bbl@bblfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
4354 \bbl@ifunset{\bbl@tempb family}\%
4355 {\bbl@providefam{\bbl@tempb}}\%
4356 {\bbl@exp{\%
4357 \\\bbl@sreplace\<\bbl@tempb family >\%
4358 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}\%
4359 % For the default font, just in case:
4360 \bbl@ifunset{\bbl@lsys\language\name}{\bbl@provide@lsys{\language\name}}{\%
4361 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}\%
4362 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}\% save bbl@rmdflt@
4363 \bbl@exp{\%
4364 \let\bbl@\bbl@tempb dflt@\language\name>\<\bbl@\bbl@tempb dflt@>\%
4365 \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\language\name>\%
4366 \<\bbl@tempb default>\<\bbl@tempb family>}}\%
4367 {\bbl@foreach\bbl@tempa{\% ie bbl@rmdflt@lang / *scrt
4368 \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:

```

4369 \def\bbl@providefam#1{\%
4370 \bbl@exp{\%
4371 \\\newcommand\<#1default>{\% Just define it
4372 \\\bbl@add@list\\\bbl@font@fams{#1}\%
4373 \\\DeclareRobustCommand\<#1family>{\%
4374 \\\not@math@alphabet\<#1family>\relax
4375 \\\fontfamily\<#1default>\selectfont}\%
4376 \\\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.

```

4377 \def\bbl@nostdfont#1{\%
4378 \bbl@ifunset{\bbl@WFF@\f@family}\%
4379 {\bbl@csarg\gdef{WFF@\f@family}}{\% Flag, to avoid dupl warns
4380 \bbl@infowarn{The current font is not a babel standard family:\\%
4381 #1\%
4382 \fontname\font\\%
4383 There is nothing intrinsically wrong with this warning, and\\%
4384 you can ignore it altogether if you do not need these\\%
4385 families. But if they are used in the document, you should be\\%

```

```

4386     aware 'babel' will no set Script and Language for them, so\\%
4387     you may consider defining a new family with \string\babelfont.\\%
4388     See the manual for further details about \string\babelfont.\\%
4389     Reported}}
4390     }%
4391 \gdef\bbl@switchfont{%
4392   \bbl@ifunset{\bbl@sys@\language}\bbl@provide@sys{\language}}}%
4393   \bbl@exp{%   eg Arabic -> arabic
4394     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}%
4395   \bbl@foreach\bbl@font@fams{%
4396     \bbl@ifunset{\bbl@##1dflt@\language}%      (1) language?
4397     {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}%    (2) from script?
4398       {\bbl@ifunset{\bbl@##1dflt@}%             2=F - (3) from generic?
4399         }%                                         123=F - nothing!
4400         {\bbl@exp{%                             3=T - from generic
4401           \global\let<\bbl@##1dflt@\language>%
4402             \<\bbl@##1dflt@>}}}%
4403         {\bbl@exp{%                             2=T - from script
4404           \global\let<\bbl@##1dflt@\language>%
4405             \<\bbl@##1dflt@*\bbl@tempa>}}}%
4406         }%                                         1=T - language, already defined
4407   \def\bbl@tempa{\bbl@nostdfont}}}%
4408   \bbl@foreach\bbl@font@fams{%   don't gather with prev for
4409     \bbl@ifunset{\bbl@##1dflt@\language}%
4410     {\bbl@cs{famrst@##1}%
4411       \global\bbl@csarg\let{famrst@##1}\relax}%
4412     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4413       \\bbl@add\\originalTeX{%
4414         \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4415         \<##1default>\<##1family>{##1}}}%
4416       \\bbl@font@set<\bbl@##1dflt@\language>% the main part!
4417       \<##1default>\<##1family>}}}%
4418   \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.

```

4419 \ifx\f@family\undefined\else   % if latex
4420   \ifcase\bbl@engine           % if pdftex
4421     \let\bbl@ckeckstdfonts\relax
4422   \else
4423     \def\bbl@ckeckstdfonts{%
4424       \begingroup
4425       \global\let\bbl@ckeckstdfonts\relax
4426       \let\bbl@tempa\@empty
4427       \bbl@foreach\bbl@font@fams{%
4428         \bbl@ifunset{\bbl@##1dflt@}%
4429         {\nameuse{##1family}%
4430           \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4431         \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4432           \space\space\fontname\font\\}%
4433         \bbl@csarg\xdef{##1dflt@}{\f@family}%
4434         \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4435       }%
4436     \ifx\bbl@tempa\@empty\else
4437       \bbl@infowarn{The following font families will use the default\\%
4438         settings for all or some languages:\\%
4439         \bbl@tempa
4440         There is nothing intrinsically wrong with it, but\\%
4441         'babel' will no set Script and Language, which could\\%

```

```

4442         be relevant in some languages. If your document uses\\%
4443         these families, consider redefining them with \string\babelfont.\\%
4444         Reported}%
4445     \fi
4446 \endgroup}
4447 \fi
4448 \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.

```

4449 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4450 \bbl@xin@{<>}{#1}%
4451 \ifin@
4452 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4453 \fi
4454 \bbl@exp{%          'Unprotected' macros return prev values
4455 \def\\#2{#1}%      eg, \rmdefault{\bbl@rmdflt@lang}
4456 \\bbl@ifsamestring{#2}{\f@family}%
4457 {\\#3%
4458 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4459 \let\\bbl@tempa\relax}%
4460 {}}}
4461 % TODO - next should be global?, but even local does its job. I'm
4462 % still not sure -- must investigate:
4463 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4464 \let\bbl@tempe\bbl@mapselect
4465 \let\bbl@mapselect\relax
4466 \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4467 \let#4\@empty %      Make sure \renewfontfamily is valid
4468 \bbl@exp{%
4469 \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4470 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4471 {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4472 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4473 {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4474 \\renewfontfamily\\#4%
4475 [\bbl@cs{lsys@\language},#2]{#3}% ie \bbl@exp{..}{#3}
4476 \begingroup
4477 #4%
4478 \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4479 \endgroup
4480 \let#4\bbl@temp@fam
4481 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4482 \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.

```

4483 \def\bbl@font@rst#1#2#3#4{%
4484 \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.

```

4485 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but

essentially – that was not the way to go :-).

```
4486 \newcommand\babelFSstore[2][{}]{%
4487   \bbl@ifblank{#1}%
4488   {\bbl@csarg\def{sname@#2}{Latin}}%
4489   {\bbl@csarg\def{sname@#2}{#1}}%
4490   \bbl@provide@dirs{#2}%
4491   \bbl@csarg\ifnum{wdir@#2}>\z@
4492     \let\bbl@beforeforeign\leavevmode
4493     \EnableBabelHook{babel-bidi}%
4494   \fi
4495   \bbl@foreach{#2}{%
4496     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4497     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4498     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4499 \def\bbl@FSstore#1#2#3#4{%
4500   \bbl@csarg\edef{#2default#1}{#3}%
4501   \expandafter\addto\csname extras#1\endcsname{%
4502     \let#4#3%
4503     \ifx#3\f@family
4504       \edef#3{\csname bbl@#2default#1\endcsname}%
4505       \fontfamily{#3}\selectfont
4506     \else
4507       \edef#3{\csname bbl@#2default#1\endcsname}%
4508     \fi}%
4509   \expandafter\addto\csname noextras#1\endcsname{%
4510     \ifx#3\f@family
4511       \fontfamily{#4}\selectfont
4512     \fi
4513     \let#3#4}}
4514 \let\bbl@langfeatures\@empty
4515 \def\babelFSfeatures{% make sure \fontspec is redefined once
4516   \let\bbl@ori@fontspec\fontspec
4517   \renewcommand\fontspec[1][{}]{%
4518     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4519   \let\babelFSfeatures\bbl@FSfeatures
4520   \babelFSfeatures}
4521 \def\bbl@FSfeatures#1#2{%
4522   \expandafter\addto\csname extras#1\endcsname{%
4523     \babel@save\bbl@langfeatures
4524     \edef\bbl@langfeatures{#2,}}
4525 \</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.

```
4526 <<{*Footnote changes}>> ≡
4527 \bbl@trace{Bidi footnotes}
4528 \ifnum\bbl@bidimode>\z@
4529   \def\bbl@footnote#1#2#3{%
4530     \@ifnextchar[%
4531       {\bbl@footnote@o{#1}{#2}{#3}}%
4532       {\bbl@footnote@x{#1}{#2}{#3}}}
4533   \long\def\bbl@footnote@x#1#2#3#4{%
4534     \bgroup
```



```

4535 \select@language@x{\bbl@main@language}%
4536 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4537 \egroup}
4538 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4539 \bgroup
4540 \select@language@x{\bbl@main@language}%
4541 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4542 \egroup}
4543 \def\bbl@footnotetext#1#2#3{%
4544 \ifnextchar[%
4545 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4546 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4547 \long\def\bbl@footnotetext@x#1#2#3#4{%
4548 \bgroup
4549 \select@language@x{\bbl@main@language}%
4550 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4551 \egroup}
4552 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4553 \bgroup
4554 \select@language@x{\bbl@main@language}%
4555 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4556 \egroup}
4557 \def\BabelFootnote#1#2#3#4{%
4558 \ifx\bbl@fn@footnote\undefined
4559 \let\bbl@fn@footnote\footnote
4560 \fi
4561 \ifx\bbl@fn@footnotetext\undefined
4562 \let\bbl@fn@footnotetext\footnotetext
4563 \fi
4564 \bbl@ifblank{#2}%
4565 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4566 \@namedef{\bbl@stripslash#1text}%
4567 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4568 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4569 \@namedef{\bbl@stripslash#1text}%
4570 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4571 \fi
4572 <</Footnote changes>>

```

Now, the code.

```

4573 (*xetex)
4574 \def\BabelStringsDefault{unicode}
4575 \let\xebbl@stop\relax
4576 \AddBabelHook{xetex}{encodedcommands}{%
4577 \def\bbl@tempa{#1}%
4578 \ifx\bbl@tempa\empty
4579 \XeTeXinputencoding"bytes"%
4580 \else
4581 \XeTeXinputencoding"#1"%
4582 \fi
4583 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4584 \AddBabelHook{xetex}{stopcommands}{%
4585 \xebbl@stop
4586 \let\xebbl@stop\relax}
4587 \def\bbl@intraspace#1 #2 #3\@@{%
4588 \bbl@csarg\gdef{\xeisp@languagename}%
4589 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4590 \def\bbl@intrapenalty#1\@@{%
4591 \bbl@csarg\gdef{\xeipn@languagename}%

```

```

4592 {\XeTeXlinebreakpenalty #1\relax}}
4593 \def\bbbl@provide@intraspace{%
4594 \bbbl@xin@{\bbbl@cl{lnbrk}}{s}%
4595 \ifin@else\bbbl@xin@{\bbbl@cl{lnbrk}}{c}\fi
4596 \ifin@
4597 \bbbl@ifunset{\bbbl@intsp@{\languagename}}{s}%
4598 {\expandafter\ifx\cename \bbbl@intsp@{\languagename}\endcsname\empty\else
4599 \ifx\bbbl@KVP@intraspace\@nil
4600 \bbbl@exp{%
4601 \\\bbbl@intraspace\bbbl@cl{intsp}}\@@}%
4602 \fi
4603 \ifx\bbbl@KVP@intrapenalty\@nil
4604 \bbbl@intrapenalty0\@@
4605 \fi
4606 \fi
4607 \ifx\bbbl@KVP@intraspace\@nil\else % We may override the ini
4608 \expandafter\bbbl@intraspace\bbbl@KVP@intraspace\@@
4609 \fi
4610 \ifx\bbbl@KVP@intrapenalty\@nil\else
4611 \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
4612 \fi
4613 \bbbl@exp{%
4614 \\\bbbl@add\<extras\languagename>{%
4615 \XeTeXlinebreaklocale "\bbbl@cl{tbcpr}"%
4616 \<\bbbl@xeisp@\languagename>%
4617 \<\bbbl@xeipn@\languagename>}%
4618 \\\bbbl@toglobal\<extras\languagename>%
4619 \\\bbbl@add\<noextras\languagename>{%
4620 \XeTeXlinebreaklocale "en"%
4621 \\\bbbl@toglobal\<noextras\languagename>}%
4622 \ifx\bbbl@ispace\@undefined
4623 \gdef\bbbl@ispace{\bbbl@cl{xeisp}}%
4624 \ifx\AtBeginDocument\@notprerr
4625 \expandafter\@secondoftwo % to execute right now
4626 \fi
4627 \AtBeginDocument{%
4628 \expandafter\bbbl@add
4629 \cselectfont \endcsname{\bbbl@ispace}%
4630 \expandafter\bbbl@toglobal\cselectfont \endcsname}%
4631 \fi}%
4632 \fi}
4633 \ifx\DisableBabelHook\@undefined\endinput\fi
4634 \AddBabelHook{babel-fontspec}{afterextras}{\bbbl@switchfont}
4635 \AddBabelHook{babel-fontspec}{beforestart}{\bbbl@ckeckstdfonts}
4636 \DisableBabelHook{babel-fontspec}
4637 <<Font selection>>
4638 \input txtbabel.def
4639 </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.

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

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdf_{tex}

and xetex.

```
4640 (*texxet)
4641 \providecommand\bbl@provide@intraspace{}
4642 \bbl@trace{Redefinitions for bidi layout}
4643 \def\bbl@sspre@caption{%
4644   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir\bbl@main@language}}}}
4645 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4646 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4647 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4648 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4649   \def\hangfrom#1{%
4650     \setbox\@tempboxa\hbox{#1}%
4651     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4652     \noindent\box\@tempboxa}
4653 \def\raggedright{%
4654   \let\@centercr
4655   \bbl@startskip\z@skip
4656   \@rightskip\@flushglue
4657   \bbl@endskip\@rightskip
4658   \parindent\z@
4659   \parfillskip\bbl@startskip}
4660 \def\raggedleft{%
4661   \let\@centercr
4662   \bbl@startskip\@flushglue
4663   \bbl@endskip\z@skip
4664   \parindent\z@
4665   \parfillskip\bbl@endskip}
4666 \fi
4667 \IfBabelLayout{lists}
4668   {\bbl@sreplace\list
4669     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4670     \def\bbl@listleftmargin{%
4671       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4672     \ifcase\bbl@engine
4673       \def\labelenumii{\theenumii}% pdfTeX doesn't reverse ()
4674       \def\p@enumiii{\p@enumii}\theenumii}%
4675     \fi
4676     \bbl@sreplace\@verbatim
4677       {\leftskip\@totalleftmargin}%
4678       {\bbl@startskip\textwidth
4679         \advance\bbl@startskip-\linewidth}%
4680     \bbl@sreplace\@verbatim
4681       {\rightskip\z@skip}%
4682       {\bbl@endskip\z@skip}}%
4683   {}
4684 \IfBabelLayout{contents}
4685   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4686     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4687   {}
4688 \IfBabelLayout{columns}
4689   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4690     \def\bbl@outputbox#1{%
4691       \hb@xt@\textwidth{%
4692         \hskip\columnwidth
4693         \hfil
4694         {\normalcolor\vrule \@width\columnseprule}%
4695         \hfil
4696         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4697       }
4698     }
4699 }
```

```

4697      \hskip-\textwidth
4698      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4699      \hskip\columnsep
4700      \hskip\columnwidth}}}%
4701  {}
4702  <<Footnote changes>>
4703  \IfBabelLayout{footnotes}%
4704  {\BabelFootnote\footnote\language\language{}{}}%
4705  \BabelFootnote\localfootnote\language\language{}{}}%
4706  \BabelFootnote\mainfootnote{}{}}{}
4707  {}

```

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.

```

4708 \IfBabelLayout{counters}%
4709  {\let\bbbl@latinarabic=\@arabic
4710   \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}}%
4711   \let\bbbl@asciroman=\@roman
4712   \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
4713   \let\bbbl@asciiRoman=\@Roman
4714   \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}}{}
4715 </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, \bbbl@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).

```

4716 (*luatex)
4717 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4718 \bbl@trace{Read language.dat}
4719 \ifx\bbl@readstream\undefined
4720 \csname newread\endcsname\bbl@readstream
4721 \fi
4722 \begingroup
4723 \toks@{}
4724 \count@ \z@ % 0=start, 1=0th, 2=normal
4725 \def\bbl@process@line#1#2 #3 #4 {%
4726   \ifx=#1%
4727     \bbl@process@synonym{#2}%
4728   \else
4729     \bbl@process@language{#1#2}{#3}{#4}%
4730   \fi
4731   \ignorespaces}
4732 \def\bbl@manylang{%
4733   \ifnum\bbl@last>\@ne
4734     \bbl@info{Non-standard hyphenation setup}%
4735   \fi
4736   \let\bbl@manylang\relax}
4737 \def\bbl@process@language#1#2#3{%
4738   \ifcase\count@
4739     \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4740   \or
4741     \count@\tw@
4742   \fi
4743   \ifnum\count@=\tw@
4744     \expandafter\addlanguage\csname l@#1\endcsname
4745     \language\allocationnumber
4746     \chardef\bbl@last\allocationnumber
4747     \bbl@manylang
4748     \let\bbl@elt\relax
4749     \xdef\bbl@languages{%
4750       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4751   \fi
4752   \the\toks@
4753   \toks@{}}
4754 \def\bbl@process@synonym@aux#1#2{%
4755   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4756   \let\bbl@elt\relax
4757   \xdef\bbl@languages{%
4758     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4759 \def\bbl@process@synonym#1{%
4760   \ifcase\count@
4761     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4762   \or
4763     \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4764   \else
4765     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4766   \fi}
4767 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4768   \chardef\l@english\z@
4769   \chardef\l@USenglish\z@

```

```

4770 \chardef\bbl@last\z@
4771 \global\@namedef\bbl@hyphendata@0{{hyphen.tex}{}}
4772 \gdef\bbl@languages{%
4773   \bbl@elt{english}{0}{hyphen.tex}{}%
4774   \bbl@elt{USenglish}{0}{}}
4775 \else
4776   \global\let\bbl@languages@format\bbl@languages
4777   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4778     \ifnum#2>\z@\else
4779       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4780     \fi}%
4781   \xdef\bbl@languages{\bbl@languages}%
4782 \fi
4783 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4784 \bbl@languages
4785 \openin\bbl@readstream=language.dat
4786 \ifeof\bbl@readstream
4787   \bbl@warning{I couldn't find language.dat. No additional\\%
4788     patterns loaded. Reported}%
4789 \else
4790   \loop
4791     \endlinechar\m@ne
4792     \read\bbl@readstream to \bbl@line
4793     \endlinechar\^^M
4794     \if T\ifeof\bbl@readstream F\fi T\relax
4795     \ifx\bbl@line\empty\else
4796       \edef\bbl@line{\bbl@line\space\space\space}%
4797       \expandafter\bbl@process@line\bbl@line\relax
4798     \fi
4799   \repeat
4800 \fi
4801 \endgroup
4802 \bbl@trace{Macros for reading patterns files}
4803 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4804 \ifx\babelcatcodetablenum\undefined
4805   \ifx\newcatcodetable\undefined
4806     \def\babelcatcodetablenum{5211}
4807     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4808   \else
4809     \newcatcodetable\babelcatcodetablenum
4810     \newcatcodetable\bbl@pattcodes
4811   \fi
4812 \else
4813   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4814 \fi
4815 \def\bbl@luapatterns#1#2{%
4816   \bbl@get@enc#1::\@@@
4817   \setbox\z@\hbox\bgroup
4818   \begingroup
4819     \savecatcodetable\babelcatcodetablenum\relax
4820     \initcatcodetable\bbl@pattcodes\relax
4821     \catcodetable\bbl@pattcodes\relax
4822     \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4823     \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4824     \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4825     \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4826     \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4827     \catcode`\'=12 \catcode`\'=12 \catcode`\`=12
4828     \input #1\relax

```

```

4829 \catcodetable\babelcatcodetablenum\relax
4830 \endgroup
4831 \def\bbbl@tempa{#2}%
4832 \ifx\bbbl@tempa\@empty\else
4833 \input #2\relax
4834 \fi
4835 \egroup}%
4836 \def\bbbl@patterns@lua#1{%
4837 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4838 \csname l@#1\endcsname
4839 \edef\bbbl@tempa{#1}%
4840 \else
4841 \csname l@#1:\f@encoding\endcsname
4842 \edef\bbbl@tempa{#1:\f@encoding}%
4843 \fi\relax
4844 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4845 \@ifundefined{bbbl@hyphendata@the\language}%
4846 {\def\bbbl@elt##1##2##3##4{%
4847 \ifnum##2=\csname l@bbbl@tempa\endcsname % #2=spanish, dutch:OT1...
4848 \def\bbbl@tempb{##3}%
4849 \ifx\bbbl@tempb\@empty\else % if not a synonymous
4850 \def\bbbl@tempc{##3}{##4}%
4851 \fi
4852 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
4853 \fi}%
4854 \bbbl@languages
4855 \@ifundefined{bbbl@hyphendata@the\language}%
4856 {\bbbl@info{No hyphenation patterns were set for\%
4857 language '\bbbl@tempa'. Reported}}%
4858 {\expandafter\expandafter\expandafter\bbbl@luapatterns
4859 \csname bbl@hyphendata@the\language\endcsname}}}%
4860 \endinput\fi
4861 % Here ends \ifx\AddBabelHook\@undefined
4862 % A few lines are only read by hyphen.cfg
4863 \ifx\DisableBabelHook\@undefined
4864 \AddBabelHook{luatex}{everylanguage}{%
4865 \def\process@language##1##2##3{%
4866 \def\process@line####1####2 ####3 ####4 {}}}
4867 \AddBabelHook{luatex}{loadpatterns}{%
4868 \input #1\relax
4869 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4870 {{#1}}}%
4871 \AddBabelHook{luatex}{loadexceptions}{%
4872 \input #1\relax
4873 \def\bbbl@tempb##1##2{{##1}{#1}}%
4874 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4875 {\expandafter\expandafter\expandafter\bbbl@tempb
4876 \csname bbl@hyphendata@the\language\endcsname}}
4877 \endinput\fi
4878 % Here stops reading code for hyphen.cfg
4879 % The following is read the 2nd time it's loaded
4880 \begingroup % TODO - to a lua file
4881 \catcode`\%=12
4882 \catcode`\'=12
4883 \catcode`\%=12
4884 \catcode`\:=12
4885 \directlua{
4886 Babel = Babel or {}
4887 function Babel.bytes(line)

```

```

4888     return line:gsub("(.)",
4889         function (chr) return unicode.utf8.char(string.byte(chr)) end)
4890 end
4891 function Babel.begin_process_input()
4892     if luatexbase and luatexbase.add_to_callback then
4893         luatexbase.add_to_callback('process_input_buffer',
4894             Babel.bytes,'Babel.bytes')
4895     else
4896         Babel.callback = callback.find('process_input_buffer')
4897         callback.register('process_input_buffer',Babel.bytes)
4898     end
4899 end
4900 function Babel.end_process_input ()
4901     if luatexbase and luatexbase.remove_from_callback then
4902         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4903     else
4904         callback.register('process_input_buffer',Babel.callback)
4905     end
4906 end
4907 function Babel.addpatterns(pp, lg)
4908     local lg = lang.new(lg)
4909     local pats = lang.patterns(lg) or ''
4910     lang.clear_patterns(lg)
4911     for p in pp:gmatch('[^%s]+') do
4912         ss = ''
4913         for i in string.utfcharacters(p:gsub('%d', '')) do
4914             ss = ss .. '%d?' .. i
4915         end
4916         ss = ss:gsub('^%%d%?%.','%%%.') .. '%d?'
4917         ss = ss:gsub('%.%%d%?$','%%%.')
4918         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4919         if n == 0 then
4920             tex.sprint(
4921                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4922                 .. p .. [[{}]])
4923             pats = pats .. ' ' .. p
4924         else
4925             tex.sprint(
4926                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4927                 .. p .. [[{}]])
4928         end
4929     end
4930     lang.patterns(lg, pats)
4931 end
4932 }
4933 \endgroup
4934 \ifx\newattribute\@undefined\else
4935     \newattribute\bbl@attr@locale
4936     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4937     \AddBabelHook{luatex}{beforeextras}{%
4938         \setattribute\bbl@attr@locale\localeid}
4939 \fi
4940 \def\BabelStringsDefault{unicode}
4941 \let\luabbl@stop\relax
4942 \AddBabelHook{luatex}{encodedcommands}{%
4943     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4944     \ifx\bbl@tempa\bbl@tempb\else
4945         \directlua{Babel.begin_process_input()}%
4946         \def\luabbl@stop{%

```



```

4947 \directlua{Babel.end_process_input()}}%
4948 \fi}%
4949 \AddBabelHook{luatex}{stopcommands}{%
4950 \luabbbl@stop
4951 \let\luabbbl@stop\relax}
4952 \AddBabelHook{luatex}{patterns}{%
4953 \@ifundefined{bbl@hyphendata@the\language}%
4954 {\def\bbl@elt##1##2##3##4{%
4955 \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4956 \def\bbl@tempb{##3}%
4957 \ifx\bbl@tempb\empty\else % if not a synonymous
4958 \def\bbl@tempc{##3}{##4}}%
4959 \fi
4960 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4961 \fi}%
4962 \bbl@languages
4963 \@ifundefined{bbl@hyphendata@the\language}%
4964 {\bbl@info{No hyphenation patterns were set for\%
4965 language '#2'. Reported}}%
4966 {\expandafter\expandafter\expandafter\bbl@luapatterns
4967 \csname bbl@hyphendata@the\language\endcsname}}}%
4968 \@ifundefined{bbl@patterns@}{}%
4969 \begingroup
4970 \bbl@xin@{\, \number\language,}{\, \bbl@pttnlist}%
4971 \ifin@ \else
4972 \ifx\bbl@patterns@\empty \else
4973 \directlua{ Babel.addpatterns(
4974 [[\bbl@patterns@]], \number\language) }%
4975 \fi
4976 \@ifundefined{bbl@patterns@#1}%
4977 \empty
4978 {\directlua{ Babel.addpatterns(
4979 [[\space\csname bbl@patterns@#1\endcsname]],
4980 \number\language) }}%
4981 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4982 \fi
4983 \endgroup}%
4984 \bbl@exp{%
4985 \bbl@ifunset{bbl@prehc@language\name}{}%
4986 {\bbl@i fblank{\bbl@cs{prehc@language\name}}}%
4987 {\prehyphenchar=\bbl@cl{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.

```

4988 \@onlypreamble\babelpatterns
4989 \AtEndOfPackage{%
4990 \newcommand\babelpatterns[2][\empty]{%
4991 \ifx\bbl@patterns@\relax
4992 \let\bbl@patterns@\empty
4993 \fi
4994 \ifx\bbl@pttnlist\empty \else
4995 \bbl@warning{%
4996 You must not intermingle \string\selectlanguage\space and\%
4997 \string\babelpatterns\space or some patterns will not\%
4998 be taken into account. Reported}%
4999 \fi
5000 \ifx\@empty#1%

```

```

5001     \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5002   \else
5003     \edef\bbl@tempb{\zap@space#1 \@empty}%
5004     \bbl@for\bbl@tempa\bbl@tempb{%
5005       \bbl@fixname\bbl@tempa
5006       \bbl@iflanguage\bbl@tempa{%
5007         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5008           \@ifundefined{bbl@patterns@\bbl@tempa}%
5009           \@empty
5010           {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5011           #2}}}%
5012   \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. *In progress*. Replace regular (ie, implicit) discretionary by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary are not touched. See Unicode UAX 14.

```

5013% TODO - to a lua file
5014 \directlua{
5015   Babel = Babel or {}
5016   Babel.linebreaking = Babel.linebreaking or {}
5017   Babel.linebreaking.before = {}
5018   Babel.linebreaking.after = {}
5019   Babel.locale = {} % Free to use, indexed with \localeid
5020   function Babel.linebreaking.add_before(func)
5021     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5022     table.insert(Babel.linebreaking.before , func)
5023   end
5024   function Babel.linebreaking.add_after(func)
5025     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5026     table.insert(Babel.linebreaking.after, func)
5027   end
5028 }
5029 \def\bbl@intraspace#1 #2 #3\@@{%
5030   \directlua{
5031     Babel = Babel or {}
5032     Babel.intraspaces = Babel.intraspaces or {}
5033     Babel.intraspaces['\csname bbl@sbcpr@\languagename\endcsname'] = %
5034       {b = #1, p = #2, m = #3}
5035     Babel.locale_props[\the\localeid].intraspace = %
5036       {b = #1, p = #2, m = #3}
5037   }}
5038 \def\bbl@intrapenalty#1\@@{%
5039   \directlua{
5040     Babel = Babel or {}
5041     Babel.intrapenalties = Babel.intrapenalties or {}
5042     Babel.intrapenalties['\csname bbl@sbcpr@\languagename\endcsname'] = #1
5043     Babel.locale_props[\the\localeid].intrapenalty = #1
5044   }}
5045 \begingroup
5046 \catcode`\%=12
5047 \catcode`\^=14
5048 \catcode`\'=12
5049 \catcode`\~=12
5050 \gdef\bbl@seaintraspace{^
5051   \let\bbl@seaintraspace\relax

```

```

5052 \directlua{
5053   Babel = Babel or {}
5054   Babel.sea_enabled = true
5055   Babel.sea_ranges = Babel.sea_ranges or {}
5056   function Babel.set_chranges (script, chrng)
5057     local c = 0
5058     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5059       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5060       c = c + 1
5061     end
5062   end
5063   function Babel.sea_disc_to_space (head)
5064     local sea_ranges = Babel.sea_ranges
5065     local last_char = nil
5066     local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
5067     for item in node.traverse(head) do
5068       local i = item.id
5069       if i == node.id'glyph' then
5070         last_char = item
5071       elseif i == 7 and item.subtype == 3 and last_char
5072         and last_char.char > 0x0C99 then
5073         quad = font.getfont(last_char.font).size
5074         for lg, rg in pairs(sea_ranges) do
5075           if last_char.char > rg[1] and last_char.char < rg[2] then
5076             lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
5077             local intraspace = Babel.intraspaces[lg]
5078             local intrapenalty = Babel.intrapenalties[lg]
5079             local n
5080             if intrapenalty ~= 0 then
5081               n = node.new(14, 0) ^^ penalty
5082               n.penalty = intrapenalty
5083               node.insert_before(head, item, n)
5084             end
5085             n = node.new(12, 13) ^^ (glue, spaceskip)
5086             node.setglue(n, intraspace.b * quad,
5087               intraspace.p * quad,
5088               intraspace.m * quad)
5089             node.insert_before(head, item, n)
5090             node.remove(head, item)
5091           end
5092         end
5093       end
5094     end
5095   end
5096 }^^
5097 \bbl@luahyphenate}
5098 \catcode`\%=14
5099 \gdef\bbl@cjkintraspaces{%
5100   \let\bbl@cjkintraspaces\relax
5101   \directlua{
5102     Babel = Babel or {}
5103     require'babel-data-cjk.lua'
5104     Babel.cjk_enabled = true
5105     function Babel.cjk_linebreak(head)
5106       local GLYPH = node.id'glyph'
5107       local last_char = nil
5108       local quad = 655360    % 10 pt = 655360 = 10 * 65536
5109       local last_class = nil
5110       local last_lang = nil

```

```

5111
5112     for item in node.traverse(head) do
5113         if item.id == GLYPH then
5114
5115             local lang = item.lang
5116
5117             local LOCALE = node.get_attribute(item,
5118                 luatexbase.registernumber'bbl@attr@locale')
5119             local props = Babel.locale_props[LOCALE]
5120
5121             local class = Babel.cjk_class[item.char].c
5122
5123             if class == 'cp' then class = 'cl' end % ]) as CL
5124             if class == 'id' then class = 'I' end
5125
5126             local br = 0
5127             if class and last_class and Babel.cjk_breaks[last_class][class] then
5128                 br = Babel.cjk_breaks[last_class][class]
5129             end
5130
5131             if br == 1 and props.linebreak == 'c' and
5132                 lang ~= \the\l@nohyphenation\space and
5133                 last_lang ~= \the\l@nohyphenation then
5134                 local intrapenalty = props.intrapenalty
5135                 if intrapenalty ~= 0 then
5136                     local n = node.new(14, 0)    % penalty
5137                     n.penalty = intrapenalty
5138                     node.insert_before(head, item, n)
5139                 end
5140                 local intraspace = props.intraspace
5141                 local n = node.new(12, 13)    % (glue, spaceskip)
5142                 node.setglue(n, intraspace.b * quad,
5143                     intraspace.p * quad,
5144                     intraspace.m * quad)
5145                 node.insert_before(head, item, n)
5146             end
5147
5148             if font.getfont(item.font) then
5149                 quad = font.getfont(item.font).size
5150             end
5151             last_class = class
5152             last_lang = lang
5153         else % if penalty, glue or anything else
5154             last_class = nil
5155         end
5156     end
5157     lang.hyphenate(head)
5158 end
5159 }%
5160 \bbl@luahyphenate}
5161 \gdef\bbl@luahyphenate{%
5162     \let\bbl@luahyphenate\relax
5163     \directlua{
5164         luatexbase.add_to_callback('hyphenate',
5165             function (head, tail)
5166                 if Babel.linebreaking.before then
5167                     for k, func in ipairs(Babel.linebreaking.before) do
5168                         func(head)
5169                     end

```

```

5170 end
5171 if Babel.cjk_enabled then
5172     Babel.cjk_linebreak(head)
5173 end
5174 lang.hyphenate(head)
5175 if Babel.linebreaking.after then
5176     for k, func in ipairs(Babel.linebreaking.after) do
5177         func(head)
5178     end
5179 end
5180 if Babel.sea_enabled then
5181     Babel.sea_disc_to_space(head)
5182 end
5183 end,
5184 'Babel.hyphenate')
5185 }
5186 }
5187 \endgroup
5188 \def\bbl@provide@intraspace{%
5189   \bbl@ifunset{\bbl@intsp@\language\name}{}%
5190   {\expandafter\ifx\csname bbl@intsp@\language\name\endcsname\@empty\else
5191     \bbl@xin{\bbl@cl{lncrk}}{c}%
5192     \bin@ % cjk
5193     \bbl@cjk_intraspace
5194     \directlua{
5195       Babel = Babel or {}
5196       Babel.locale_props = Babel.locale_props or {}
5197       Babel.locale_props[\the\localeid].linebreak = 'c'
5198     }%
5199     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@}%
5200     \ifx\bbl@KVP@intrapenalty\nil
5201       \bbl@intrapenalty0\@@
5202     \fi
5203   \else % sea
5204     \bbl@sea_intraspace
5205     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@}%
5206     \directlua{
5207       Babel = Babel or {}
5208       Babel.sea_ranges = Babel.sea_ranges or {}
5209       Babel.set_chranges('\bbl@cl{sbcpr}',
5210                          '\bbl@cl{chrng}')
5211     }%
5212     \ifx\bbl@KVP@intrapenalty\nil
5213       \bbl@intrapenalty0\@@
5214     \fi
5215   \fi
5216 \fi
5217 \ifx\bbl@KVP@intrapenalty\nil\else
5218   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5219 \fi}

```

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.

Work in progress.

Common stuff.

```
5220 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5221 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5222 \DisableBabelHook{babel-fontspec}
5223 <<Font selection>>
```

13.6 Automatic fonts and ids switching

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

```
5224% TODO - to a lua file
5225 \directlua{
5226 Babel.script_blocks = {
5227   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5228              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5229   ['Armn'] = {{0x0530, 0x058F}},
5230   ['Beng'] = {{0x0980, 0x09FF}},
5231   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5232   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5233   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5234              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5235   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5236   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5237              {0xAB00, 0xAB2F}},
5238   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5239   % Don't follow strictly Unicode, which places some Coptic letters in
5240   % the 'Greek and Coptic' block
5241   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5242   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5243              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5244              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5245              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5246              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5247              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5248   ['Hebr'] = {{0x0590, 0x05FF}},
5249   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5250              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5251   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5252   ['Knda'] = {{0x0C80, 0x0CFF}},
5253   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5254              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5255              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5256   ['Lao'] = {{0x0E80, 0x0EFF}},
5257   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5258              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5259              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5260   ['Mahj'] = {{0x11150, 0x1117F}},
5261   ['Mlym'] = {{0x0D00, 0x0D7F}},
```

```

5262 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5263 ['Orya'] = {{0x0B00, 0x0B7F}},
5264 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5265 ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5266 ['Taml'] = {{0x0B80, 0x0BFF}},
5267 ['Telu'] = {{0x0C00, 0x0C7F}},
5268 ['Tfng'] = {{0x2D30, 0x2D7F}},
5269 ['Thai'] = {{0x0E00, 0x0E7F}},
5270 ['Tibt'] = {{0x0F00, 0x0FFF}},
5271 ['Vaii'] = {{0xA500, 0xA63F}},
5272 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5273 }
5274
5275 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5276 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5277 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5278
5279 function Babel.locale_map(head)
5280   if not Babel.locale_mapped then return head end
5281
5282   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5283   local GLYPH = node.id('glyph')
5284   local inmath = false
5285   local toloc_save
5286   for item in node.traverse(head) do
5287     local toloc
5288     if not inmath and item.id == GLYPH then
5289       % Optimization: build a table with the chars found
5290       if Babel.chr_to_loc[item.char] then
5291         toloc = Babel.chr_to_loc[item.char]
5292       else
5293         for lc, maps in pairs(Babel.loc_to_scr) do
5294           for _, rg in pairs(maps) do
5295             if item.char >= rg[1] and item.char <= rg[2] then
5296               Babel.chr_to_loc[item.char] = lc
5297               toloc = lc
5298               break
5299             end
5300           end
5301         end
5302       end
5303       % Now, take action, but treat composite chars in a different
5304       % fashion, because they 'inherit' the previous locale. Not yet
5305       % optimized.
5306       if not toloc and
5307         (item.char >= 0x0300 and item.char <= 0x036F) or
5308         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5309         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5310         toloc = toloc_save
5311       end
5312       if toloc and toloc > -1 then
5313         if Babel.locale_props[toloc].lg then
5314           item.lang = Babel.locale_props[toloc].lg
5315           node.set_attribute(item, LOCALE, toloc)
5316         end
5317         if Babel.locale_props[toloc]['/'..item.font] then
5318           item.font = Babel.locale_props[toloc]['/'..item.font]
5319         end
5320         toloc_save = toloc

```

```

5321     end
5322     elseif not inmath and item.id == 7 then
5323         item.replace = item.replace and Babel.locale_map(item.replace)
5324         item.pre      = item.pre and Babel.locale_map(item.pre)
5325         item.post     = item.post and Babel.locale_map(item.post)
5326     elseif item.id == node.id'math' then
5327         inmath = (item.subtype == 0)
5328     end
5329 end
5330 return head
5331 end
5332 }

```

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

```

5333 \newcommand\babelcharproperty[1]{%
5334   \count@=#1\relax
5335   \ifvmode
5336     \expandafter\bbl@chprop
5337   \else
5338     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5339               vertical mode (preamble or between paragraphs)}%
5340     {See the manual for futher info}%
5341   \fi}
5342 \newcommand\bbl@chprop[3][\the\count@]{%
5343   \@tempcnta=#1\relax
5344   \bbl@ifunset{bbl@chprop@#2}%
5345   {\bbl@error{No property named '#2'. Allowed values are\\%
5346             direction (bc), mirror (bmg), and linebreak (lb)}%
5347    {See the manual for futher info}}%
5348   }%
5349   \loop
5350     \bbl@cs{chprop@#2}{#3}%
5351   \ifnum\count@<\@tempcnta
5352     \advance\count@\@ne
5353   \repeat}
5354 \def\bbl@chprop@direction#1{%
5355   \directlua{
5356     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5357     Babel.characters[\the\count@]['d'] = '#1'
5358   }}
5359 \let\bbl@chprop@bc\bbl@chprop@direction
5360 \def\bbl@chprop@mirror#1{%
5361   \directlua{
5362     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5363     Babel.characters[\the\count@]['m'] = '\number#1'
5364   }}
5365 \let\bbl@chprop@bmg\bbl@chprop@mirror
5366 \def\bbl@chprop@linebreak#1{%
5367   \directlua{
5368     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5369     Babel.cjk_characters[\the\count@]['c'] = '#1'
5370   }}
5371 \let\bbl@chprop@lb\bbl@chprop@linebreak
5372 \def\bbl@chprop@locale#1{%
5373   \directlua{
5374     Babel.chr_to_loc = Babel.chr_to_loc or {}
5375     Babel.chr_to_loc[\the\count@] =
5376       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space

```



```
5377  }}
```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the luatex manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```
5378 \begingroup % TODO - to a lua file
5379 \catcode`\#=12
5380 \catcode`\%=12
5381 \catcode`\&=14
5382 \directlua{
5383   Babel.linebreaking.post_replacements = {}
5384   Babel.linebreaking.pre_replacements = {}
5385
5386   function Babel.str_to_nodes(fn, matches, base)
5387     local n, head, last
5388     if fn == nil then return nil end
5389     for s in string.utfvalues(fn(matches)) do
5390       if base.id == 7 then
5391         base = base.replace
5392       end
5393       n = node.copy(base)
5394       n.char = s
5395       if not head then
5396         head = n
5397       else
5398         last.next = n
5399       end
5400       last = n
5401     end
5402     return head
5403   end
5404
5405   function Babel.fetch_word(head, mode)
5406     local word_string = ''
5407     local word_nodes = {}
5408     local lang
5409     local item = head
5410     local inmath = false
5411     local mode = 0 && 'word' -- first steps in merging with subtext
5412
5413     while item do
5414
5415       if item.id == 29
5416         and not(item.char == 124) && ie, not |
5417         and not(item.char == 61) && ie, not =
5418         and not inmath
5419         and (item.lang == lang or lang == nil) then
5420         lang = lang or item.lang
```

```

5421     word_string = word_string .. unicode.utf8.char(item.char)
5422     word_nodes[#word_nodes+1] = item
5423
5424     elseif item.id == 7 and item.subtype == 2
5425         and not inmath and mode == 0 then
5426         word_string = word_string .. '='
5427         word_nodes[#word_nodes+1] = item
5428
5429     elseif item.id == 7 and item.subtype == 3
5430         and not inmath and mode == 0 then
5431         word_string = word_string .. '|'
5432         word_nodes[#word_nodes+1] = item
5433
5434     elseif item.id == 11 and item.subtype == 0 then
5435         inmath = true
5436
5437     elseif mode > 0 and item.id == 12 and item.subtype == 13 then
5438         word_string = word_string .. '|'
5439         word_nodes[#word_nodes+1] = item
5440
5441     elseif word_string == '' then
5442         && pass
5443
5444     else
5445         return word_string, word_nodes, item, lang
5446     end
5447
5448     item = item.next
5449 end
5450 end
5451
5452 function Babel.post_hyphenate_replace(head)
5453     local u = unicode.utf8
5454     local lbkr = Babel.linebreaking.post_replacements
5455     local word_head = head
5456
5457     while true do
5458         local w, wn, nw, lang = Babel.fetch_word(word_head)
5459         if not lang then return head end
5460
5461         if not lbkr[lang] then
5462             break
5463         end
5464
5465         && For every pattern
5466         for k=1, #lbkr[lang] do
5467             local p = lbkr[lang][k].pattern
5468             local r = lbkr[lang][k].replace
5469
5470             && For every match.
5471             while true do
5472                 local matches = { u.match(w, p) }
5473                 if #matches < 2 then break end
5474
5475                 && Get and remove empty captures (with ()), which return a
5476                 && number with the position), and keep actual captures
5477                 && (from (...)), if any, in matches.
5478                 local first = table.remove(matches, 1)
5479                 local last = table.remove(matches, #matches)

```

```

5480
5481      %% Fix offsets, from bytes to unicode. Explained above.
5482      first = u.len(w:sub(1, first-1)) + 1
5483      last  = u.len(w:sub(1, last-1))
5484
5485      local new  %% used when inserting and removing nodes
5486      local changed = 0
5487
5488      %% This loop traverses the replace list and takes the
5489      %% corresponding actions
5490      for q = first, last do
5491          local crep = r[q-first+1]
5492          local char_node = wn[q]
5493          local char_base = char_node
5494
5495          if crep and crep.data then
5496              char_base = wn[crep.data+first-1]
5497          end
5498
5499          if crep == {} then
5500              break
5501          elseif crep == nil then
5502              changed = changed + 1
5503              node.remove(head, char_node)
5504          elseif crep and (crep.pre or crep.no or crep.post) then
5505              changed = changed + 1
5506              d = node.new(7, 0)  %% (disc, discretionary)
5507              d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5508              d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5509              d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5510              d.attr = char_base.attr
5511              if crep.pre == nil then  %% TeXbook p96
5512                  d.penalty = crep.penalty or tex.hyphenpenalty
5513              else
5514                  d.penalty = crep.penalty or tex.exhyphenpenalty
5515              end
5516              head, new = node.insert_before(head, char_node, d)
5517              node.remove(head, char_node)
5518              if q == 1 then
5519                  word_head = new
5520              end
5521          elseif crep and crep.string then
5522              changed = changed + 1
5523              local str = crep.string(matches)
5524              if str == '' then
5525                  if q == 1 then
5526                      word_head = char_node.next
5527                  end
5528                  head, new = node.remove(head, char_node)
5529              elseif char_node.id == 29 and u.len(str) == 1 then
5530                  char_node.char = string.utfvalue(str)
5531              else
5532                  local n
5533                  for s in string.utfvalues(str) do
5534                      if char_node.id == 7 then
5535                          log('Automatic hyphens cannot be replaced, just removed.')
5536                      else
5537                          n = node.copy(char_base)
5538                      end

```

```

5539         n.char = s
5540         if q == 1 then
5541             head, new = node.insert_before(head, char_node, n)
5542             word_head = new
5543         else
5544             node.insert_before(head, char_node, n)
5545         end
5546     end
5547
5548     node.remove(head, char_node)
5549     end %% string length
5550     end %% if char and char.string
5551     end %% for char in match
5552     if changed > 20 then
5553         texio.write('Too many changes. Ignoring the rest.')
5554     elseif changed > 0 then
5555         w, wn, nw = Babel.fetch_word(word_head)
5556     end
5557
5558     end %% for match
5559     end %% for patterns
5560     word_head = nw
5561     end %% for words
5562     return head
5563 end
5564
5565 &%%&
5566 &%% Preliminary code for \babelprehyphenation
5567 &%% TODO. Copypaste pattern. Merge with fetch_word
5568 function Babel.fetch_subtext(head)
5569     local word_string = ''
5570     local word_nodes = {}
5571     local lang
5572     local item = head
5573     local inmath = false
5574
5575     while item do
5576
5577         if item.id == 29 then
5578             local locale = node.get_attribute(item, Babel.attr_locale)
5579
5580             if not(item.char == 124) &%% ie, not | = space
5581                 and not inmath
5582                 and (locale == lang or lang == nil) then
5583                 lang = lang or locale
5584                 word_string = word_string .. unicode.utf8.char(item.char)
5585                 word_nodes[#word_nodes+1] = item
5586             end
5587
5588             if item == node.tail(head) then
5589                 item = nil
5590                 return word_string, word_nodes, item, lang
5591             end
5592
5593             elseif item.id == 12 and item.subtype == 13 and not inmath then
5594                 word_string = word_string .. '|'
5595                 word_nodes[#word_nodes+1] = item
5596
5597                 if item == node.tail(head) then

```

```

5598         item = nil
5599         return word_string, word_nodes, item, lang
5600     end
5601
5602     elseif item.id == 11 and item.subtype == 0 then
5603         inmath = true
5604
5605     elseif word_string == '' then
5606         &% pass
5607
5608     else
5609         return word_string, word_nodes, item, lang
5610
5611     end
5612
5613     item = item.next
5614 end
5615 end
5616
5617 &% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5618 function Babel.pre_hyphenate_replace(head)
5619     local u = unicode.utf8
5620     local lbkr = Babel.linebreaking.pre_replacements
5621     local word_head = head
5622
5623     while true do &% for each subtext block
5624         local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5625         if not lang then return head end
5626
5627         if not lbkr[lang] then
5628             break
5629         end
5630
5631         for k=1, #lbkr[lang] do &% for each saved posthyphen
5632             local p = lbkr[lang][k].pattern
5633             local r = lbkr[lang][k].replace
5634
5635             while true do
5636                 local matches = { u.match(w, p) }
5637                 local reparse = true
5638                 if #matches < 2 then break end
5639
5640                 local first = table.remove(matches, 1)
5641                 local last = table.remove(matches, #matches)
5642
5643                 &% Fix offsets, from bytes to unicode.
5644                 first = u.len(w:sub(1, first-1)) + 1
5645                 last = u.len(w:sub(1, last-1))
5646
5647                 local new &% used when inserting and removing nodes
5648                 local changed = 0
5649
5650                 &% This loop traverses the matched substring and takes the
5651                 &% corresponding action stored in the replacement list.
5652                 &% sc is the position in substr nodes / string
5653                 &% rc is the replacement table index
5654                 sc = first-1
5655                 rc = 0
5656                 while rc < last-first+1 do

```

```

5657         sc = sc + 1
5658         rc = rc + 1
5659         local crep = r[rc]
5660         local char_node = wn[sc]
5661         local char_base = char_node
5662
5663         if crep and crep.data then
5664             char_base = wn[crep.data+first-1]
5665         end
5666
5667         if crep and next(crep) == nil then && {}
5668             reparsed = false
5669         elseif crep == nil then && remove
5670             changed = changed + 1
5671             node.remove(head, char_node)
5672             table.remove(wn, sc)
5673             reparsed = false
5674             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5675             sc = sc - 1
5676         elseif crep and crep.insert then
5677             && print(crep.insert)
5678         elseif crep and crep.string then
5679             changed = changed + 1
5680             local str = crep.string(matches)
5681             if str == '' then
5682                 if sc == 1 then
5683                     word_head = char_node.next
5684                 end
5685                 head, new = node.remove(head, char_node)
5686             elseif char_node.id == 29 and u.len(str) == 1 then
5687                 && For one-to-one can we modify directly the
5688                 && values without re-fetching.
5689                 char_node.char = string.utfvalue(str)
5690                 reparsed = false
5691                 w = u.sub(w, 1, sc-1) .. str .. u.sub(w, sc+1)
5692             else
5693                 local n
5694                 for s in string.utfvalues(str) do
5695                     if char_node.id == 7 then
5696                         log('Automatic hyphens cannot be replaced, just removed.')
5697                     else
5698                         n = node.copy(char_base)
5699                     end
5700                     n.char = s
5701                     if sc == 1 then
5702                         head, new = node.insert_before(head, char_node, n)
5703                         word_head = new
5704                     else
5705                         node.insert_before(head, char_node, n)
5706                     end
5707                 end
5708
5709                 node.remove(head, char_node)
5710             end && string length
5711         end && if char and char.string
5712     end && while char in match
5713
5714     if changed > 20 then
5715         texio.write('Too many changes. Ignoring the rest.')

```

```

5716         elseif changed > 0 then
5717             if reparse then
5718                 w, wn, nw = Babel.fetch_subtext(word_head)
5719             else
5720                 reparse = true
5721             end
5722         end
5723     end && for match
5724 end && for patterns
5725     word_head = nw
5726 end && for subtext
5727     return head
5728 end
5729 &&%% end of preliminary code for \babelprehyphenation
5730
5731 && The following functions belong to the next macro
5732
5733 && This table stores capture maps, numbered consecutively
5734 Babel.capture_maps = {}
5735
5736 function Babel.capture_func(key, cap)
5737     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
5738     ret = ret:gsub('{{[0-9]}|([^|]+)|(.-)}', Babel.capture_func_map)
5739     ret = ret:gsub("%[%[%]%.%", '')
5740     ret = ret:gsub("%.%[%[%]%.%", '')
5741     return key .. "[=function(m) return ]] .. ret .. [[ end]]
5742 end
5743
5744 function Babel.capt_map(from, mapno)
5745     return Babel.capture_maps[mapno][from] or from
5746 end
5747
5748 && Handle the {n|abc|ABC} syntax in captures
5749 function Babel.capture_func_map(capno, from, to)
5750     local froms = {}
5751     for s in string.utfcharacters(from) do
5752         table.insert(froms, s)
5753     end
5754     local cnt = 1
5755     table.insert(Babel.capture_maps, {})
5756     local mlen = table.getn(Babel.capture_maps)
5757     for s in string.utfcharacters(to) do
5758         Babel.capture_maps[mlen][froms[cnt]] = s
5759         cnt = cnt + 1
5760     end
5761     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5762         (mlen) .. ").." .. "["
5763 end
5764 }

```

Now the TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ - becomes `function(m) return m[1]..m[1]..'-' end`, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to $m[1]$. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which

explains the internal group, too).

```
5765 \catcode`\# = 6
5766 \gdef\babelposthyphenation#1#2#3{&%
5767   \bbl@activateposthyphen
5768   \beginngroup
5769     \def\babeltempa{\bbl@add@list\babeltempb}&%
5770     \let\babeltempb\@empty
5771     \bbl@foreach{#3}{&%
5772       \bbl@ifsamestring{##1}{remove}&%
5773       {\bbl@add@list\babeltempb{nil}}&%
5774       {\directlua{
5775         local rep = {[##1]}
5776         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5777         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5778         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5779         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5780       }}&%
5781     \directlua{
5782       local lbkr = Babel.linebreaking.post_replacements
5783       local u = unicode.utf8
5784       &% Convert pattern:
5785       local patt = string.gsub([=[#2]=], '%s', '')
5786       if not u.find(patt, '()', nil, true) then
5787         patt = '()' .. patt .. '()'
5788       end
5789       patt = string.gsub(patt, '%(%)%^', '^()')
5790       patt = string.gsub(patt, '%$%(%)', '()$')
5791       patt = u.gsub(patt, '{(.)}',
5792         function (n)
5793           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5794         end)
5795       lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5796       table.insert(lbkr[\the\csname l@#1\endcsname],
5797         { pattern = patt, replace = { \babeltempb } })
5798     }&%
5799   \endgroup}
5800 % TODO. Working !!! Copypaste pattern.
5801 \gdef\babelprehyphenation#1#2#3{&%
5802   \bbl@activateprehyphen
5803   \beginngroup
5804     \def\babeltempa{\bbl@add@list\babeltempb}&%
5805     \let\babeltempb\@empty
5806     \bbl@foreach{#3}{&%
5807       \bbl@ifsamestring{##1}{remove}&%
5808       {\bbl@add@list\babeltempb{nil}}&%
5809       {\directlua{
5810         local rep = {[##1]}
5811         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5812         tex.print([[\string\babeltempa{}}] .. rep .. [[}}]])
5813       }}&%
5814     \directlua{
5815       local lbkr = Babel.linebreaking.pre_replacements
5816       local u = unicode.utf8
5817       &% Convert pattern:
5818       local patt = string.gsub([=[#2]=], '%s', '')
5819       if not u.find(patt, '()', nil, true) then
5820         patt = '()' .. patt .. '()'
5821       end
```



```

5822     patt = u.gsub(patt, '{(.)}',
5823                 function (n)
5824                     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5825                 end)
5826     lbr[\the\csname bbl@id@@#1\endcsname] = lbr[\the\csname bbl@id@@#1\endcsname] or {}
5827     table.insert(lbr[\the\csname bbl@id@@#1\endcsname],
5828                 { pattern = patt, replace = { \babeltempb } })
5829 }&%
5830 \endgroup}
5831 \endgroup
5832 \def\bbl@activateposthyphen{%
5833     \let\bbl@activateposthyphen\relax
5834     \directlua{
5835         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5836     }}
5837 % TODO. Working !!!
5838 \def\bbl@activateprehyphen{%
5839     \let\bbl@activateprehyphen\relax
5840     \directlua{
5841         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5842     }}

```

13.7 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.

```

5843 \bbl@trace{Redefinitions for bidi layout}
5844 \ifx\@eqnnum\undefined\else
5845     \ifx\bbl@attr@dir\undefined\else
5846         \edef\@eqnnum{%
5847             \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5848             \unexpanded\expandafter{\@eqnnum}}%
5849     \fi
5850 \fi
5851 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5852 \ifnum\bbl@bidimode>\z@
5853     \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5854         \bbl@exp{%
5855             \mathdir\the\bodydir
5856             #1%           Once entered in math, set boxes to restore values
5857             \<ifmmode>%
5858             \everyvbox{%
5859                 \the\everyvbox
5860                 \bodydir\the\bodydir
5861                 \mathdir\the\mathdir
5862                 \everyhbox{\the\everyhbox}%
5863                 \everyvbox{\the\everyvbox}}%

```

```

5864 \everyhbox{%
5865 \the\everyhbox
5866 \bodydir\the\bodydir
5867 \mathdir\the\mathdir
5868 \everyhbox{\the\everyhbox}%
5869 \everyvbox{\the\everyvbox}%
5870 \<fi>}}%
5871 \def\@hangfrom#1{%
5872 \setbox\@tempboxa\hbox{{#1}}%
5873 \hangindent\wd\@tempboxa
5874 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5875 \shapemode\@ne
5876 \fi
5877 \noindent\box\@tempboxa}
5878 \fi
5879 \IfBabelLayout{tabular}
5880 {\let\bb1@OL@tabular\@tabular
5881 \bb1@replace\@tabular{$}\{\bb1@nextfake$\}%
5882 \let\bb1@NL@tabular\@tabular
5883 \AtBeginDocument{%
5884 \ifx\bb1@NL@tabular\@tabular\else
5885 \bb1@replace\@tabular{$}\{\bb1@nextfake$\}%
5886 \let\bb1@NL@tabular\@tabular
5887 \fi}}
5888 {}
5889 \IfBabelLayout{lists}
5890 {\let\bb1@OL@list\list
5891 \bb1@sreplace\list{\parshape}\{\bb1@listparshape}%
5892 \let\bb1@NL@list\list
5893 \def\bb1@listparshape#1#2#3{%
5894 \parshape #1 #2 #3 %
5895 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5896 \shapemode\tw@
5897 \fi}}
5898 {}
5899 \IfBabelLayout{graphics}
5900 {\let\bb1@pictresetdir\relax
5901 \def\bb1@pictsetdir{%
5902 \ifcase\bb1@thetextdir
5903 \let\bb1@pictresetdir\relax
5904 \else
5905 \textdir TLT\relax
5906 \def\bb1@pictresetdir{\textdir TRT\relax}%
5907 \fi}%
5908 \let\bb1@OL@picture\@picture
5909 \let\bb1@OL@put\put
5910 \bb1@sreplace\@picture{\hskip-}\{\bb1@pictsetdir\hskip-}%
5911 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5912 \@killglue
5913 \raise#2\unitlength
5914 \hb@xt@\z@\{\kern#1\unitlength{\bb1@pictresetdir#3}\hss}}%
5915 \AtBeginDocument
5916 {\ifx\tikz@atbegin@node\@undefined\else
5917 \let\bb1@OL@pgfpicture\pgfpicture
5918 \bb1@sreplace\pgfpicture{\pgfpicturetrue}%
5919 {\bb1@pictsetdir\pgfpicturetrue}%
5920 \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir}%
5921 \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5922 \fi}}

```

5923 {}

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.

```

5924 \IfBabelLayout{counters}%
5925   {\let\bbl@OL@@textsuperscript\textsuperscript
5926    \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5927    \let\bbl@latinarabic=\@arabic
5928    \let\bbl@OL@@arabic\@arabic
5929    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5930    \@ifpackagewith{babel}{bidi=default}%
5931      {\let\bbl@asciroman=\@roman
5932       \let\bbl@OL@@roman\@roman
5933       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5934       \let\bbl@asciiRoman=\@Roman
5935       \let\bbl@OL@@roman\@Roman
5936       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5937       \let\bbl@OL@labelenumii\labelenumii
5938       \def\labelenumii{}\theenumii}%
5939       \let\bbl@OL@p@enumiii\p@enumiii
5940       \def\p@enumiii{\p@enumii}\theenumii{}\{}\}%
5941 <<Footnote changes>>
5942 \IfBabelLayout{footnotes}%
5943   {\let\bbl@OL@footnote\footnote
5944    \BabelFootnote\footnote\language{}{}\}%
5945    \BabelFootnote\localfootnote\language{}{}\}%
5946    \BabelFootnote\mainfootnote{}\{}\}%
5947   {}

```

Some \LaTeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5948 \IfBabelLayout{extras}%
5949   {\let\bbl@OL@underline\underline
5950    \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake\$@@underline}%
5951    \let\bbl@OL@LaTeX2e\LaTeX2e
5952    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5953     \if b\expandafter\car\@series\@nil\boldmath\fi
5954     \babelsublr{%
5955       \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
5956   {}
5957 </luatex>

```

13.8 Auto bidi with basic and basic-r

The file babel-data-bidi.lua currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (`<l>`, `<r>` or `<al>`).

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

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

```
5958 (*basic-r)
5959 Babel = Babel or {}
5960
5961 Babel.bidi_enabled = true
5962
5963 require('babel-data-bidi.lua')
5964
5965 local characters = Babel.characters
5966 local ranges = Babel.ranges
5967
5968 local DIR = node.id("dir")
5969
5970 local function dir_mark(head, from, to, outer)
5971   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5972   local d = node.new(DIR)
5973   d.dir = '+' .. dir
5974   node.insert_before(head, from, d)
5975   d = node.new(DIR)
5976   d.dir = '-' .. dir
5977   node.insert_after(head, to, d)
5978 end
5979
5980 function Babel.bidi(head, ispar)
5981   local first_n, last_n          -- first and last char with nums
5982   local last_es                  -- an auxiliary 'last' used with nums
5983   local first_d, last_d          -- first and last char in L/R block
5984   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):

```

5985 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5986 local strong_lr = (strong == 'l') and 'l' or 'r'
5987 local outer = strong
5988
5989 local new_dir = false
5990 local first_dir = false
5991 local inmath = false
5992
5993 local last_lr
5994
5995 local type_n = ''
5996
5997 for item in node.traverse(head) do
5998
5999   -- three cases: glyph, dir, otherwise
6000   if item.id == node.id'glyph'
6001     or (item.id == 7 and item.subtype == 2) then
6002
6003     local itemchar
6004     if item.id == 7 and item.subtype == 2 then
6005       itemchar = item.replace.char
6006     else
6007       itemchar = item.char
6008     end
6009     local chardata = characters[itemchar]
6010     dir = chardata and chardata.d or nil
6011     if not dir then
6012       for nn, et in ipairs(ranges) do
6013         if itemchar < et[1] then
6014           break
6015         elseif itemchar <= et[2] then
6016           dir = et[3]
6017           break
6018         end
6019       end
6020     end
6021     dir = dir or 'l'
6022     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).

```

6023   if new_dir then
6024     attr_dir = 0
6025     for at in node.traverse(item.attr) do
6026       if at.number == luatexbase.registernumber'bbl@attr@dir' then
6027         attr_dir = at.value % 3
6028       end
6029     end
6030     if attr_dir == 1 then
6031       strong = 'r'
6032     elseif attr_dir == 2 then
6033       strong = 'al'

```

```

6034     else
6035         strong = 'l'
6036     end
6037     strong_lr = (strong == 'l') and 'l' or 'r'
6038     outer = strong_lr
6039     new_dir = false
6040 end
6041
6042 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

6043     dir_real = dir          -- We need dir_real to set strong below
6044     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:

```

6045     if strong == 'al' then
6046         if dir == 'en' then dir = 'an' end          -- W2
6047         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6048         strong_lr = 'r'                             -- W3
6049     end

```

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

```

6050     elseif item.id == node.id'dir' and not inmath then
6051         new_dir = true
6052         dir = nil
6053     elseif item.id == node.id'math' then
6054         inmath = (item.subtype == 0)
6055     else
6056         dir = nil          -- Not a char
6057     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 `texdir` 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>`.

```

6058     if dir == 'en' or dir == 'an' or dir == 'et' then
6059         if dir ~= 'et' then
6060             type_n = dir
6061         end
6062         first_n = first_n or item
6063         last_n = last_n or item
6064         last_es = nil
6065     elseif dir == 'es' and last_n then -- W3+W6
6066         last_es = item
6067     elseif dir == 'cs' then          -- it's right - do nothing
6068     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6069         if strong_lr == 'r' and type_n ~= '' then
6070             dir_mark(head, first_n, last_n, 'r')
6071         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6072             dir_mark(head, first_n, last_n, 'r')
6073             dir_mark(head, first_d, last_d, outer)
6074             first_d, last_d = nil, nil
6075         elseif strong_lr == 'l' and type_n ~= '' then
6076             last_d = last_n

```

```

6077     end
6078     type_n = ''
6079     first_n, last_n = nil, nil
6080 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:

```

6081   if dir == 'l' or dir == 'r' then
6082     if dir ~= outer then
6083       first_d = first_d or item
6084       last_d = item
6085     elseif first_d and dir ~= strong_lr then
6086       dir_mark(head, first_d, last_d, outer)
6087       first_d, last_d = nil, nil
6088     end
6089   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.

```

6090   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6091     item.char = characters[item.char] and
6092               characters[item.char].m or item.char
6093   elseif (dir or new_dir) and last_lr ~= item then
6094     local mir = outer .. strong_lr .. (dir or outer)
6095     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6096       for ch in node.traverse(node.next(last_lr)) do
6097         if ch == item then break end
6098         if ch.id == node.id'glyph' and characters[ch.char] then
6099           ch.char = characters[ch.char].m or ch.char
6100         end
6101       end
6102     end
6103   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).

```

6104   if dir == 'l' or dir == 'r' then
6105     last_lr = item
6106     strong = dir_real          -- Don't search back - best save now
6107     strong_lr = (strong == 'l') and 'l' or 'r'
6108   elseif new_dir then
6109     last_lr = nil
6110   end
6111 end

```

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

```

6112   if last_lr and outer == 'r' then
6113     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6114       if characters[ch.char] then
6115         ch.char = characters[ch.char].m or ch.char
6116       end
6117     end
6118   end

```

```

6119 if first_n then
6120     dir_mark(head, first_n, last_n, outer)
6121 end
6122 if first_d then
6123     dir_mark(head, first_d, last_d, outer)
6124 end

```

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

```

6125 return node.prev(head) or head
6126 end
6127 </basic-r>

```

And here the Lua code for bidi=basic:

```

6128 (*basic)
6129 Babel = Babel or {}
6130
6131 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6132
6133 Babel.fontmap = Babel.fontmap or {}
6134 Babel.fontmap[0] = {}      -- l
6135 Babel.fontmap[1] = {}      -- r
6136 Babel.fontmap[2] = {}      -- al/an
6137
6138 Babel.bidi_enabled = true
6139 Babel.mirroring_enabled = true
6140
6141 require('babel-data-bidi.lua')
6142
6143 local characters = Babel.characters
6144 local ranges = Babel.ranges
6145
6146 local DIR = node.id('dir')
6147 local GLYPH = node.id('glyph')
6148
6149 local function insert_implicit(head, state, outer)
6150     local new_state = state
6151     if state.sim and state.eim and state.sim ~= state.eim then
6152         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6153         local d = node.new(DIR)
6154         d.dir = '+' .. dir
6155         node.insert_before(head, state.sim, d)
6156         local d = node.new(DIR)
6157         d.dir = '-' .. dir
6158         node.insert_after(head, state.eim, d)
6159     end
6160     new_state.sim, new_state.eim = nil, nil
6161     return head, new_state
6162 end
6163
6164 local function insert_numeric(head, state)
6165     local new
6166     local new_state = state
6167     if state.san and state.ean and state.san ~= state.ean then
6168         local d = node.new(DIR)
6169         d.dir = '+TLT'
6170         _, new = node.insert_before(head, state.san, d)
6171         if state.san == state.sim then state.sim = new end
6172         local d = node.new(DIR)

```



```

6173     d.dir = '-TLT'
6174     _, new = node.insert_after(head, state.ean, d)
6175     if state.ean == state.eim then state.eim = new end
6176 end
6177 new_state.san, new_state.ean = nil, nil
6178 return head, new_state
6179 end
6180
6181 -- TODO - \hbox with an explicit dir can lead to wrong results
6182 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6183 -- was s made to improve the situation, but the problem is the 3-dir
6184 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6185 -- well.
6186
6187 function Babel.bidi(head, ispar, hdir)
6188     local d -- d is used mainly for computations in a loop
6189     local prev_d = ''
6190     local new_d = false
6191
6192     local nodes = {}
6193     local outer_first = nil
6194     local inmath = false
6195
6196     local glue_d = nil
6197     local glue_i = nil
6198
6199     local has_en = false
6200     local first_et = nil
6201
6202     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6203
6204     local save_outer
6205     local temp = node.get_attribute(head, ATDIR)
6206     if temp then
6207         temp = temp % 3
6208         save_outer = (temp == 0 and 'l') or
6209                     (temp == 1 and 'r') or
6210                     (temp == 2 and 'al')
6211     elseif ispar then -- Or error? Shouldn't happen
6212         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6213     else -- Or error? Shouldn't happen
6214         save_outer = ('TRT' == hdir) and 'r' or 'l'
6215     end
6216     -- when the callback is called, we are just _after_ the box,
6217     -- and the textdir is that of the surrounding text
6218     -- if not ispar and hdir ~= tex.textdir then
6219     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6220     -- end
6221     local outer = save_outer
6222     local last = outer
6223     -- 'al' is only taken into account in the first, current loop
6224     if save_outer == 'al' then save_outer = 'r' end
6225
6226     local fontmap = Babel.fontmap
6227
6228     for item in node.traverse(head) do
6229
6230         -- In what follows, #node is the last (previous) node, because the
6231         -- current one is not added until we start processing the neutrals.

```

```

6232
6233 -- three cases: glyph, dir, otherwise
6234 if item.id == GLYPH
6235     or (item.id == 7 and item.subtype == 2) then
6236
6237     local d_font = nil
6238     local item_r
6239     if item.id == 7 and item.subtype == 2 then
6240         item_r = item.replace -- automatic discs have just 1 glyph
6241     else
6242         item_r = item
6243     end
6244     local chardata = characters[item_r.char]
6245     d = chardata and chardata.d or nil
6246     if not d or d == 'nsm' then
6247         for nn, et in ipairs(ranges) do
6248             if item_r.char < et[1] then
6249                 break
6250             elseif item_r.char <= et[2] then
6251                 if not d then d = et[3]
6252                 elseif d == 'nsm' then d_font = et[3]
6253                 end
6254                 break
6255             end
6256         end
6257     end
6258     d = d or 'l'
6259
6260     -- A short 'pause' in bidi for mapfont
6261     d_font = d_font or d
6262     d_font = (d_font == 'l' and 0) or
6263             (d_font == 'nsm' and 0) or
6264             (d_font == 'r' and 1) or
6265             (d_font == 'al' and 2) or
6266             (d_font == 'an' and 2) or nil
6267     if d_font and fontmap and fontmap[d_font][item_r.font] then
6268         item_r.font = fontmap[d_font][item_r.font]
6269     end
6270
6271     if new_d then
6272         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6273         if inmath then
6274             attr_d = 0
6275         else
6276             attr_d = node.get_attribute(item, ATDIR)
6277             attr_d = attr_d % 3
6278         end
6279         if attr_d == 1 then
6280             outer_first = 'r'
6281             last = 'r'
6282         elseif attr_d == 2 then
6283             outer_first = 'r'
6284             last = 'al'
6285         else
6286             outer_first = 'l'
6287             last = 'l'
6288         end
6289         outer = last
6290         has_en = false

```

```

6291         first_et = nil
6292         new_d = false
6293     end
6294
6295     if glue_d then
6296         if (d == 'l' and 'l' or 'r') ~= glue_d then
6297             table.insert(nodes, {glue_i, 'on', nil})
6298         end
6299         glue_d = nil
6300         glue_i = nil
6301     end
6302
6303     elseif item.id == DIR then
6304         d = nil
6305         new_d = true
6306
6307     elseif item.id == node.id'glue' and item.subtype == 13 then
6308         glue_d = d
6309         glue_i = item
6310         d = nil
6311
6312     elseif item.id == node.id'math' then
6313         inmath = (item.subtype == 0)
6314
6315     else
6316         d = nil
6317     end
6318
6319     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6320     if last == 'al' and d == 'en' then
6321         d = 'an'          -- W3
6322     elseif last == 'al' and (d == 'et' or d == 'es') then
6323         d = 'on'          -- W6
6324     end
6325
6326     -- EN + CS/ES + EN      -- W4
6327     if d == 'en' and #nodes >= 2 then
6328         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6329             and nodes[#nodes-1][2] == 'en' then
6330             nodes[#nodes][2] = 'en'
6331         end
6332     end
6333
6334     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6335     if d == 'an' and #nodes >= 2 then
6336         if (nodes[#nodes][2] == 'cs')
6337             and nodes[#nodes-1][2] == 'an' then
6338             nodes[#nodes][2] = 'an'
6339         end
6340     end
6341
6342     -- ET/EN                  -- W5 + W7->1 / W6->on
6343     if d == 'et' then
6344         first_et = first_et or (#nodes + 1)
6345     elseif d == 'en' then
6346         has_en = true
6347         first_et = first_et or (#nodes + 1)
6348     elseif first_et then      -- d may be nil here !
6349         if has_en then

```

```

6350         if last == 'l' then
6351             temp = 'l'      -- W7
6352         else
6353             temp = 'en'     -- W5
6354         end
6355     else
6356         temp = 'on'        -- W6
6357     end
6358     for e = first_et, #nodes do
6359         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6360     end
6361     first_et = nil
6362     has_en = false
6363 end
6364
6365 if d then
6366     if d == 'al' then
6367         d = 'r'
6368         last = 'al'
6369     elseif d == 'l' or d == 'r' then
6370         last = d
6371     end
6372     prev_d = d
6373     table.insert(nodes, {item, d, outer_first})
6374 end
6375
6376 outer_first = nil
6377
6378 end
6379
6380 -- TODO -- repeated here in case EN/ET is the last node. Find a
6381 -- better way of doing things:
6382 if first_et then      -- dir may be nil here !
6383     if has_en then
6384         if last == 'l' then
6385             temp = 'l'      -- W7
6386         else
6387             temp = 'en'     -- W5
6388         end
6389     else
6390         temp = 'on'        -- W6
6391     end
6392     for e = first_et, #nodes do
6393         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6394     end
6395 end
6396
6397 -- dummy node, to close things
6398 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6399
6400 ----- NEUTRAL -----
6401
6402 outer = save_outer
6403 last = outer
6404
6405 local first_on = nil
6406
6407 for q = 1, #nodes do
6408     local item

```

```

6409
6410     local outer_first = nodes[q][3]
6411     outer = outer_first or outer
6412     last = outer_first or last
6413
6414     local d = nodes[q][2]
6415     if d == 'an' or d == 'en' then d = 'r' end
6416     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6417
6418     if d == 'on' then
6419         first_on = first_on or q
6420     elseif first_on then
6421         if last == d then
6422             temp = d
6423         else
6424             temp = outer
6425         end
6426         for r = first_on, q - 1 do
6427             nodes[r][2] = temp
6428             item = nodes[r][1] -- MIRRORING
6429             if Babel.mirroring_enabled and item.id == GLYPH
6430                 and temp == 'r' and characters[item.char] then
6431                 local font_mode = font.fonts[item.font].properties.mode
6432                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6433                     item.char = characters[item.char].m or item.char
6434                 end
6435             end
6436         end
6437         first_on = nil
6438     end
6439
6440     if d == 'r' or d == 'l' then last = d end
6441 end
6442
6443 ----- IMPLICIT, REORDER -----
6444
6445 outer = save_outer
6446 last = outer
6447
6448 local state = {}
6449 state.has_r = false
6450
6451 for q = 1, #nodes do
6452     local item = nodes[q][1]
6453
6454     outer = nodes[q][3] or outer
6455
6456     local d = nodes[q][2]
6457
6458     if d == 'nsm' then d = last end -- W1
6459     if d == 'en' then d = 'an' end
6460     local isdir = (d == 'r' or d == 'l')
6461
6462     if outer == 'l' and d == 'an' then
6463         state.san = state.san or item
6464         state.ean = item
6465     elseif state.san then
6466         head, state = insert_numeric(head, state)
6467     end
6468 end

```

```

6468     end
6469
6470     if outer == 'l' then
6471         if d == 'an' or d == 'r' then      -- im -> implicit
6472             if d == 'r' then state.has_r = true end
6473             state.sim = state.sim or item
6474             state.eim = item
6475         elseif d == 'l' and state.sim and state.has_r then
6476             head, state = insert_implicit(head, state, outer)
6477         elseif d == 'l' then
6478             state.sim, state.eim, state.has_r = nil, nil, false
6479         end
6480     else
6481         if d == 'an' or d == 'l' then
6482             if nodes[q][3] then -- nil except after an explicit dir
6483                 state.sim = item -- so we move sim 'inside' the group
6484             else
6485                 state.sim = state.sim or item
6486             end
6487             state.eim = item
6488         elseif d == 'r' and state.sim then
6489             head, state = insert_implicit(head, state, outer)
6490         elseif d == 'r' then
6491             state.sim, state.eim = nil, nil
6492         end
6493     end
6494
6495     if isdir then
6496         last = d      -- Don't search back - best save now
6497     elseif d == 'on' and state.san then
6498         state.san = state.san or item
6499         state.ean = item
6500     end
6501
6502 end
6503
6504 return node.prev(head) or head
6505 end
6506 </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.

```
6507 ⟨*nil⟩
6508 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
6509 \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.

```
6510 \ifx\l@nil\@undefined
6511   \newlanguage\l@nil
6512   \@namedef{bbl@hyphendata@the\l@nil}{\{}}{}% Remove warning
6513   \let\bbl@elt\relax
6514   \edef\bbl@languages{% Add it to the list of languages
6515     \bbl@languages\bbl@elt{nil}{the\l@nil}{}}{}
6516 \fi
```

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

```
6517 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
6518 \let\captionnil\@empty
6519 \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.

```
6520 \ldf@finish{nil}
6521 ⟨/nil⟩
```

16 Support for Plain \TeX (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 \TeX -format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with \TeX , you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing \TeX sees, we need to set some category codes just to be able to change the definition of `\input`.

```
6522 ⟨*bplain | blplain⟩
6523 \catcode`\{=1 % left brace is begin-group character
6524 \catcode`\}=2 % right brace is end-group character
6525 \catcode`\#=6 % hash mark is macro parameter character
```

```
6526 \openin 0 hyphen.cfg
6527 \ifeof0
6528 \else
6529   \let\input
```

```

6530 \def\input #1 {%
6531     \let\input\@
6532     \a hyphen.cfg
6533     \let\@undefined
6534 }
6535 \fi
6536 </bplain | bplain>

```

```
6537 <bplain>\a plain.tex
6538 <blplain>\a lplain.tex
```

```
6539 <bplain>\def\fmtname{babel-plain}
6540 <blplain>\def\fmtname{babel-lplain}
```

```

6541 <<{*Emulate LaTeX}> \equiv
6542 % == Code for plain ==
6543 \def\@empty{}
6544 \def\loadlocalcfg#1{%
6545   \openin0#1.cfg
6546   \ifeof0
6547     \closein0
6548   \else
6549     \closein0
6550     {\immediate\write16{*****}%}
6551     \immediate\write16{* Local config file #1.cfg used}%
6552     \immediate\write16{*}%
6553   }
6554   \input #1.cfg\relax
6555 \fi
6556 \@endoflfd}

```

6557 \long\def\@firstofone#1{#1}


```

6558 \long\def\@firstoftwo#1#2{#1}
6559 \long\def\@secondoftwo#1#2{#2}
6560 \def\@nnil{\@nil}
6561 \def\@gobbletwo#1#2{}
6562 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}{%
6563 \def\@star@or@long#1{%
6564   \@ifstar
6565   {\let\l@ngrel@x\relax#1}%
6566   {\let\l@ngrel@x\long#1}}
6567 \let\l@ngrel@x\relax
6568 \def\@car#1#2\@nil{#1}
6569 \def\@cdr#1#2\@nil{#2}
6570 \let\@typeset@protect\relax
6571 \let\protected@edef\edef
6572 \long\def\@gobble#1{}
6573 \edef\@backslashchar{\expandafter\@gobble\string\}
6574 \def\strip@prefix#1>{}
6575 \def\g@addto@macro#1#2{%
6576   \toks@\expandafter{#1#2}%
6577   \xdef#1{\the\toks@}}
6578 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6579 \def\@nameuse#1{\csname #1\endcsname}
6580 \def\@ifundefined#1{%
6581   \expandafter\ifx\csname#1\endcsname\relax
6582     \expandafter\@firstoftwo
6583   \else
6584     \expandafter\@secondoftwo
6585   \fi}
6586 \def\@expandtwoargs#1#2#3{%
6587   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6588 \def\zap@space#1 #2{%
6589   #1%
6590   \ifx#2\@empty\else\expandafter\zap@space\fi
6591   #2}
6592 \let\bb1@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}`.

```

6593 \ifx\@preamblecmds\@undefined
6594   \def\@preamblecmds{}
6595 \fi
6596 \def\@onlypreamble#1{%
6597   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6598     \@preamblecmds\do#1}}
6599 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

6600 \def\begin{document}{%
6601   \@begindocumenthook
6602   \global\let\@begindocumenthook\@undefined
6603   \def\do##1{\global\let##1\@undefined}%
6604   \@preamblecmds
6605   \global\let\do\noexpand}
6606 \ifx\@begindocumenthook\@undefined
6607   \def\@begindocumenthook{}
6608 \fi
6609 \@onlypreamble\@begindocumenthook
6610 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick L^AT_EX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```
6611 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6612 \@onlypreamble\AtEndOfPackage
6613 \def\@endofldf{}
6614 \@onlypreamble\@endofldf
6615 \let\bbl@afterlang\@empty
6616 \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.

```
6617 \catcode`\&=\z@
6618 \ifx&\if@filesw\@undefined
6619 \expandafter\let\csname if@filesw\expandafter\endcsname
6620 \csname iffalse\endcsname
6621 \fi
6622 \catcode`\&=4
```

Mimick L^AT_EX's commands to define control sequences.

```
6623 \def\newcommand{\@star@or@long\new@command}
6624 \def\new@command#1{%
6625 \testopt{\@newcommand#1}0}
6626 \def\@newcommand#1[#2]{%
6627 \@ifnextchar [{\@xargdef#1[#2]}%
6628 {\@argdef#1[#2]}}
6629 \long\def\@argdef#1[#2]#3{%
6630 \@yargdef#1\@ne{#2}{#3}}
6631 \long\def\@xargdef#1[#2][#3]#4{%
6632 \expandafter\def\expandafter#1\expandafter{%
6633 \expandafter\@protected@testopt\expandafter #1%
6634 \csname\string#1\expandafter\endcsname{#3}}%
6635 \expandafter\@yargdef \csname\string#1\endcsname
6636 \tw@{#2}{#4}}
6637 \long\def\@yargdef#1#2#3{%
6638 \@tempcnta#3\relax
6639 \advance \@tempcnta \@ne
6640 \let\@hash@\relax
6641 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6642 \@tempcntb #2%
6643 \@whilenum\@tempcntb <\@tempcnta
6644 \do{%
6645 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6646 \advance\@tempcntb \@ne}%
6647 \let\@hash@###
6648 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6649 \def\providecommand{\@star@or@long\provide@command}
6650 \def\provide@command#1{%
6651 \begingroup
6652 \escapechar\m@ne\xdef\@gtempa{\string#1}%
6653 \endgroup
6654 \expandafter\@ifundefined\@gtempa
6655 {\def\reserved@a{\new@command#1}}%
6656 {\let\reserved@a\relax
6657 \def\reserved@a{\new@command\reserved@a}}%
6658 \reserved@a}%
6659 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6660 \def\declare@robustcommand#1{%
```

```

6661 \edef\reserved@a{\string#1}%
6662 \def\reserved@b{#1}%
6663 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6664 \edef#1{%
6665     \ifx\reserved@a\reserved@b
6666         \noexpand\x@protect
6667         \noexpand#1%
6668     \fi
6669     \noexpand\protect
6670     \expandafter\noexpand\csname
6671         \expandafter\@gobble\string#1 \endcsname
6672 }%
6673 \expandafter\new@command\csname
6674     \expandafter\@gobble\string#1 \endcsname
6675 }
6676 \def\x@protect#1{%
6677     \ifx\protect\@typeset@protect\else
6678         \@x@protect#1%
6679     \fi
6680 }
6681 \catcode`\&=\z@ % Trick to hide conditionals
6682 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

6683 \def\bbl@tempa{\csname newif\endcsname&fin@}
6684 \catcode`\&=4
6685 \ifx\in@\undefined
6686     \def\in@#1#2{%
6687         \def\in@@##1#1##2##3\in@@{%
6688             \ifx\in@@##2\in@false\else\in@true\fi}%
6689         \in@@#2#1\in@\in@@}
6690 \else
6691     \let\bbl@tempa\@empty
6692 \fi
6693 \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).

```

6694 \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.

```

6695 \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.

```

6696 \ifx\@tempcnta\undefined
6697     \csname newcount\endcsname\@tempcnta\relax
6698 \fi
6699 \ifx\@tempcntb\undefined

```

```

6700 \csname newcount\endcsname\@tempcntb\relax
6701 \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).

```

6702 \ifx\bye\@undefined
6703 \advance\count10 by -2\relax
6704 \fi
6705 \ifx\@ifnextchar\@undefined
6706 \def\@ifnextchar#1#2#3{%
6707   \let\reserved@d=#1%
6708   \def\reserved@a{#2}\def\reserved@b{#3}%
6709   \futurelet\@let@token\@ifnch}
6710 \def\@ifnch{%
6711   \ifx\@let@token\@sptoken
6712     \let\reserved@c\@xifnch
6713   \else
6714     \ifx\@let@token\reserved@d
6715       \let\reserved@c\reserved@a
6716     \else
6717       \let\reserved@c\reserved@b
6718     \fi
6719   \fi
6720   \reserved@c}
6721 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6722 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6723 \fi
6724 \def\@testopt#1#2{%
6725   \@ifnextchar[#{1}{#1[#2]}}
6726 \def\@protected@testopt#1{%
6727   \ifx\protect\@typeset@protect
6728     \expandafter\@testopt
6729   \else
6730     \@x@protect#1%
6731   \fi}
6732 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6733   #2\relax}\fi}
6734 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6735   \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

6736 \def\DeclareTextCommand{%
6737   \@dec@text@cmd\providecommand
6738 }
6739 \def\ProvideTextCommand{%
6740   \@dec@text@cmd\providecommand
6741 }
6742 \def\DeclareTextSymbol#1#2#3{%
6743   \@dec@text@cmd\chardef#1{#2}#3\relax
6744 }
6745 \def\@dec@text@cmd#1#2#3{%
6746   \expandafter\def\expandafter#2%
6747     \expandafter{%
6748       \csname#3-cmd\expandafter\endcsname
6749       \expandafter#2%
6750       \csname#3\string#2\endcsname

```

```

6751     }%
6752 %    \let\@ifdefinable\@rc@ifdefinable
6753     \expandafter#1\csname#3\string#2\endcsname
6754 }
6755 \def\@current@cmd#1{%
6756     \ifx\protect\@typeset@protect\else
6757         \noexpand#1\expandafter\@gobble
6758     \fi
6759 }
6760 \def\@changed@cmd#1#2{%
6761     \ifx\protect\@typeset@protect
6762         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6763             \expandafter\ifx\csname ?\string#1\endcsname\relax
6764                 \expandafter\def\csname ?\string#1\endcsname{%
6765                     \@changed@x@err{#1}%
6766                 }%
6767             \fi
6768             \global\expandafter\let
6769                 \csname\cf@encoding \string#1\expandafter\endcsname
6770                 \csname ?\string#1\endcsname
6771             \fi
6772             \csname\cf@encoding\string#1%
6773             \expandafter\endcsname
6774         \else
6775             \noexpand#1%
6776         \fi
6777 }
6778 \def\@changed@x@err#1{%
6779     \errhelp{Your command will be ignored, type <return> to proceed}%
6780     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}%
6781 \def\DeclareTextCommandDefault#1{%
6782     \DeclareTextCommand#1?%
6783 }
6784 \def\ProvideTextCommandDefault#1{%
6785     \ProvideTextCommand#1?%
6786 }
6787 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6788 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6789 \def\DeclareTextAccent#1#2#3{%
6790     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6791 }
6792 \def\DeclareTextCompositeCommand#1#2#3#4{%
6793     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6794     \edef\reserved@b{\string##1}%
6795     \edef\reserved@c{%
6796         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6797     \ifx\reserved@b\reserved@c
6798         \expandafter\expandafter\expandafter\ifx
6799             \expandafter\@car\reserved@a\relax\relax\nil
6800             \@text@composite
6801         \else
6802             \edef\reserved@b##1{%
6803                 \def\expandafter\noexpand
6804                     \csname#2\string#1\endcsname####1{%
6805                     \noexpand\@text@composite
6806                     \expandafter\noexpand\csname#2\string#1\endcsname
6807                     ####1\noexpand\@empty\noexpand\@text@composite
6808                     {##1}%
6809                 }%

```

```

6810         }%
6811         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6812     \fi
6813     \expandafter\def\csname\expandafter\string\csname
6814     #2\endcsname\string#1-\string#3\endcsname{#4}
6815 \else
6816     \errhelp{Your command will be ignored, type <return> to proceed}%
6817     \errmessage{\string\DeclareTextCompositeCommand\space used on
6818         inappropriate command \protect#1}
6819 \fi
6820 }
6821 \def\@text@composite#1#2#3\@text@composite{%
6822     \expandafter\@text@composite@x
6823     \csname\string#1-\string#2\endcsname
6824 }
6825 \def\@text@composite@x#1#2{%
6826     \ifx#1\relax
6827         #2%
6828     \else
6829         #1%
6830     \fi
6831 }
6832 %
6833 \def\@strip@args#1:#2-#3\@strip@args{#2}
6834 \def\DeclareTextComposite#1#2#3#4{%
6835     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6836     \bgroup
6837         \lcode`\@=#4%
6838         \lowercase{%
6839     \egroup
6840         \reserved@a \@%
6841     }%
6842 }
6843 %
6844 \def\UseTextSymbol#1#2{#2}
6845 \def\UseTextAccent#1#2#3{}
6846 \def\@use@text@encoding#1{}
6847 \def\DeclareTextSymbolDefault#1#2{%
6848     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6849 }
6850 \def\DeclareTextAccentDefault#1#2{%
6851     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6852 }
6853 \def\cf@encoding{OT1}

```

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

```

6854 \DeclareTextAccent{"}{OT1}{127}
6855 \DeclareTextAccent{'}{OT1}{19}
6856 \DeclareTextAccent{^}{OT1}{94}
6857 \DeclareTextAccent`}{OT1}{18}
6858 \DeclareTextAccent{~}{OT1}{126}

```

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

```

6859 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6860 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
6861 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
6862 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
6863 \DeclareTextSymbol{\i}{OT1}{16}

```

```
6864 \DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```
6865 \ifx\scriptsize\undefined
6866   \let\scriptsize\sevenrm
6867 \fi
6868 % End of code for plain
6869 <</Emulate LaTeX>>
```

A proxy file:

```
6870 <*plain>
6871 \input babel.def
6872 </plain>
```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

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