

Babel

Version 3.55.2315
2021/03/18

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	8
1.4	Modifiers	8
1.5	Troubleshooting	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	34
1.18	Dates	35
1.19	Accessing language info	36
1.20	Hyphenation and line breaking	37
1.21	Selection based on BCP 47 tags	40
1.22	Selecting scripts	41
1.23	Selecting directions	41
1.24	Language attributes	46
1.25	Hooks	46
1.26	Languages supported by babel with ldf files	47
1.27	Unicode character properties in luatex	48
1.28	Tweaking some features	49
1.29	Tips, workarounds, known issues and notes	49
1.30	Current and future work	50
1.31	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	51
3	The interface between the core of babel and the language definition files	52
3.1	Guidelines for contributed languages	53
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	56
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	57
4	Changes	61
4.1	Changes in babel version 3.9	61
II	Source code	62

5	Identification and loading of required files	62
6	locale directory	62
7	Tools	63
7.1	Multiple languages	67
7.2	The Package File (\LaTeX , babel.sty)	67
7.3	base	69
7.4	Conditional loading of shorthands	71
7.5	Cross referencing macros	73
7.6	Marks	75
7.7	Preventing clashes with other packages	76
7.7.1	ifthen	76
7.7.2	varioref	77
7.7.3	hhline	77
7.7.4	hyperref	78
7.7.5	fancyhdr	78
7.8	Encoding and fonts	79
7.9	Basic bidi support	80
7.10	Local Language Configuration	85
8	The kernel of Babel (babel.def, common)	89
8.1	Tools	90
9	Multiple languages	90
9.1	Selecting the language	93
9.2	Errors	101
9.3	Hooks	104
9.4	Setting up language files	106
9.5	Shorthands	108
9.6	Language attributes	117
9.7	Support for saving macro definitions	119
9.8	Short tags	120
9.9	Hyphens	120
9.10	Multiencoding strings	122
9.11	Macros common to a number of languages	128
9.12	Making glyphs available	129
9.12.1	Quotation marks	129
9.12.2	Letters	130
9.12.3	Shorthands for quotation marks	131
9.12.4	Umlauts and tremas	132
9.13	Layout	133
9.14	Load engine specific macros	134
9.15	Creating and modifying languages	134
10	Adjusting the Babel bahavior	154
11	Loading hyphenation patterns	155
12	Font handling with fontspec	160

13	Hooks for XeTeX and LuaTeX	164
13.1	XeTeX	164
13.2	Layout	166
13.3	LuaTeX	168
13.4	Southeast Asian scripts	174
13.5	CJK line breaking	177
13.6	Automatic fonts and ids switching	178
13.7	Layout	190
13.8	Auto bidi with basic and basic-r	193
14	Data for CJK	204
15	The ‘nil’ language	204
16	Support for Plain T_EX (plain.def)	205
16.1	Not renaming hyphen.tex	205
16.2	Emulating some L ^A T _E X features	206
16.3	General tools	206
16.4	Encoding related macros	210
17	Acknowledgements	212

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 repository](#). The most recent features can be still unstable.

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

TROUBLESHOOTING A common source of trouble is a wrong setting of the input encoding. Depending on the \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 with pdfTeX follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDFTEX

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

EXAMPLE With 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 with the default font in English and Spanish, and FreeSerif in Russian is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

NOTE Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or 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

```
\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.

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

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:

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.

```
\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}**

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

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

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

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

Spaces after the environment are ignored.

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

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

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented behavior and it is just a version as environment of `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while `otherlanguage*` 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 deprecated and `otherlanguage*` (the starred version) is preferred, because the former does not take into account possible changes in encodings of characters like, say, ‘ done by some languages (eg, italian, french, ukraineb).

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

1.9 More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with the option `fontenc`.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 four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

`\shorthandon` `{\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` `*{\char}`

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

New 3.9a User shorthands are not always alive, as they may be deactivated by languages (for example, if you use `"` for your user shorthands and switch from german to french, they

stop working). Therefore, a starred version `\usesshorthands*{⟨char⟩}` is provided, which makes sure shorthands are always activated.

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

`\defineshorthand` [`⟨language⟩`, `⟨language⟩`, ...]{`⟨shorthand⟩`}{`⟨code⟩`}

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

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

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

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

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

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

```
\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` {`⟨language⟩`}

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 `ngerman` with

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

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

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.

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]{\{\language shorthands{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

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

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

⁷Thanks to Enrico Gregorio

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

\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 ease 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 `\ldf` name strings).

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

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

LUATEX/XETEX

```

\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

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

Or also:

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

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

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

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

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

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

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

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

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

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

en	English ^{ul}	kkj	Kako
eo	Esperanto ^{ul}	kl	Kalaallisut
es-MX	Spanish ^{ul}	kln	Kalenjin
es	Spanish ^{ul}	km	Khmer
et	Estonian ^{ul}	kn	Kannada ^{ul}
eu	Basque ^{ul}	ko	Korean
ewo	Ewondo	kok	Konkani
fa	Persian ^{ul}	ks	Kashmiri
ff	Fulah	ksb	Shambala
fi	Finnish ^{ul}	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French ^{ul}	ky	Kyrgyz
fr-BE	French ^{ul}	lag	Langi
fr-CA	French ^{ul}	lb	Luxembourgish
fr-CH	French ^{ul}	lg	Ganda
fr-LU	French ^{ul}	lkt	Lakota
fur	Friulian ^{ul}	ln	Lingala
fy	Western Frisian	lo	Lao ^{ul}
ga	Irish ^{ul}	lrc	Northern Luri
gd	Scottish Gaelic ^{ul}	lt	Lithuanian ^{ul}
gl	Galician ^{ul}	lu	Luba-Katanga
grc	Ancient Greek ^{ul}	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian ^{ul}
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa ¹	mg	Malagasy
ha	Hausa	mgf	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ¹
hy	Armenian ^u	ms-SG	Malay ¹
ia	Interlingua ^{ul}	ms	Malay ^{ul}
id	Indonesian ^{ul}	mt	Maltese
ig	Igbo	mua	Mundang
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

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

aghem	chinese-hans-mo
akan	chinese-hans-sg
albanian	chinese-hans
american	chinese-hant-hk
amharic	chinese-hant-mo
ancientgreek	chinese-hant
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
bena	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

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

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

lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese

polish	serbian-latn
polytonicgreek	serbian
portuguese-br	shambala
portuguese-brazil	shona
portuguese-portugal	sichuanyi
portuguese-pt	sinhala
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

vietnamese	yangben
vunjo	yiddish
walser	yoruba
welsh	zarma
westernfrisian	zulu afrikaans

Modifying and adding values to ini files

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

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

1.14 Selecting fonts

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

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

```

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}

```

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

LUATEX/XETEX

```

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

```

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

```

\babelfont{kai}{FandolKai}

```

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

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

LUATEX/XETEX

```

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

```

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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 import'ed from ini files, you can modify the values of specific keys, like:

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

```
\addto\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. Without the optional argument it just loads some additional tools if provided by the ini file, like extra counters.

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

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

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

`import=` *⟨language-tag⟩*

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

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

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

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


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

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

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

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

captions= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

A special value is `+`, which allocates a new language (in the \TeX 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}
```

Remember there is an alternative syntax for the latter:

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

script= $\langle script-name \rangle$

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

language= $\langle language-name \rangle$

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

alph= $\langle counter-name \rangle$

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

Alph= $\langle counter-name \rangle$

Same for `\Alph`.

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

onchar= ids | fonts

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

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

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

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

intrapenalty= $\langle penalty \rangle$

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

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

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

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

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

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

The styles are:

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

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

1.18 Dates

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

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

By default the calendar is the Gregorian, but an `ini` file 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. Çîleya Pêşîn 2019*, but with `variant=iza fa` it prints *31'ê Çîleya 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` $\langle\textit{language}\rangle\{\langle\textit{true}\rangle\}\{\langle\textit{false}\rangle\}$

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 $\text{T}_{\text{E}}\text{X}$ 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` $\langle\textit{field}\rangle$

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` $\ast\{\langle\textit{macro}\rangle\}\{\langle\textit{locale}\rangle\}\{\langle\textit{property}\rangle\}$

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 patters (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are store in an internal macro named `\bbl@languages` (see the code for further details), but note several locales may share a single `\language`, so they are separated concepts. In `luatex`, the `\localeid` is saved in each node (where it makes sense) as an attribute, too.

1.20 Hyphenation and line breaking

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

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

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

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

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

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

There are also some differences with \LaTeX : (1) the character used is that set for the current font, while in \LaTeX it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in \LaTeX , but it can be changed to another value by redefining `\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` [*`\langle language \rangle`*, *`\langle language \rangle`*, ...]{*`\langle exceptions \rangle`*}

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` [*`\langle language \rangle`*, *`\langle language \rangle`*, ...]{*`\langle patterns \rangle`*}

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`).

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.

\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* → *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 *([íû])*, the replacement could be *{1|íû|íû}*, which maps *í* to *í*, and *û* to *û*, so that the diaeresis is removed.

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

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

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

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

New 3.44-3.52 This command is not strictly about hyphenation, but it is included here because it is a clear counterpart of `\babelposthyphenation`. It is similar to the latter, 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, while *|* stands for an ordinary space; (3) in the replacement, discretionaries are not accepted.

It handles glyphs and spaces (but you can not insert spaces).

Performance is still somewhat poor in some cases, but it is fast in the typical ones.

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

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

```
\babelprovide[hyphenrules=+]{russian-latin} % Create locale
\babelprehyphenation{russian-latin}{([sz])h} % Create rule
{
  string = {1|sz|šž},
  remove
}
```

EXAMPLE The following rule prevent the word “a” from being at the end of a line:


```

\babelprehyphenation{english}{|a|}
  {}, {}, % Keep first space and a
  {insert, penalty = 10000}, % Insert penalty
  {} % Keep last space
}

```

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: $\text{fr-Latn-FR} \rightarrow \text{fr-Latn} \rightarrow \text{fr-FR} \rightarrow \text{fr}$. Languages with the same resolved name are considered the same. Case is normalized before, so that $\text{fr-latn-fr} \rightarrow \text{fr-Latn-FR}$. 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` `{⟨text⟩}`

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

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

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.

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 الهليني (الاجريقي) بـ
    Arabia أو Aravia (بالاغريقية Αραβία)، استخدم الرومان ثلاث
    بادئات بـ“Arabia” على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
    حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection{.section}`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for

numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it can depend on the counter format.

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

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr `{\lr-text}`

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

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

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

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

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

`\BabelPatchSection` `{⟨section-name⟩}`

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.24 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 `TEX` parameters (`#1`, `#2`, `#3`), with the meaning given:

adddialect (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 uppsorbian
Welsh welsh

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

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

```

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

```

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

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).

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

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

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

```
\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` *{(key-value-list)}*

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

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`), \TeX will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.
- Both `lATEX` 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 (`lATEX`); (2) make it unactive (your settings); (3) make `babel` shorthands active (`babel`); (4) reload `hline` (`babel`, now with the correct catcodes for `|` and `:`).

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²⁰ So, if you write a chunk of French text with `\foreignlanguage`, the

²⁰This explains why \TeX 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.

apostrophes might not be taken into account. This is a limitation of \TeX , not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.30 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`).

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.

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.

2 Loading languages with `language.dat`

\TeX and most engines based on it (`pdfTeX`, `xetex`, $\epsilon\text{-}\TeX$, the main exception being `luatex`) require hyphenation patterns to be preloaded when a format is created (eg, \LaTeX , \XeLaTeX , `pdfLaTeX`). `babel` provides a tool which has become standard in many distributions and based on a “configuration file” named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

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

2.1 Format

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

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

```
% 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
```

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.

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

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

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

A typical error when using `babel` is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain $\text{T}_{\text{E}}\text{X}$ users, so the files have to be coded so that they can be read by both $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and plain $\text{T}_{\text{E}}\text{X}$. The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the `babel` system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date<lang>` but not `\captions<lang>` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.
- Language names must be all lowercase. If an unknown language is selected, `babel` will attempt setting it after lowercasing its name.

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

- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is /).

Some recommendations:

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

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

3.1 Guidelines for contributed languages

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

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

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

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

²⁶But not removed, for backward compatibility.

- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

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

\bbl@declare@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 L^AT_EX command `\ProvidesPackage`.

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

3.3 Skeleton

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

```

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

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

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

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

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

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

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

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

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

```



```

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

`\bbl@activate`
`\bbl@deactivate`

The command `\bbl@activate` is used to change the way an active character expands. `\bbl@activate` ‘switches on’ the active behavior of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

`\declare@shorthand`

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

`\bbl@add@special`
`\bbl@remove@special`

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

3.5 Support for saving macro definitions

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

`\babel@save`

To save the current meaning of any control sequence, the macro `\babel@save` is provided.

²⁷This mechanism was introduced by Bernd Raichle.

It takes one argument, $\langle csname \rangle$, the control sequence for which the meaning has to be saved.

`\babel@savevariable`

A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the $\langle variable \rangle$.

The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6 Support for extending macros

`\addto`

The macro `\addto{ $\langle control sequence \rangle$ { $\langle \TeX code \rangle$ }}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`.

Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

3.7 Macros common to a number of languages

`\bbl@allowhyphens`

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

`\allowhyphens`

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

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

`\set@low@box`

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

`\save@sf@q`

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

`\bbl@frenchspacing`

`\bbl@nonfrenchspacing`

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

3.8 Encoding-dependent strings

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

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

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

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

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

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 a encoded way).

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

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

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

\EndBabelCommands
```

A real example is:

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

²⁸In future releases further categories may be added.

```

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

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

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

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

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

\EndBabelCommands

```

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

\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

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

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 `\@uc1clist` (eg, `\bb\BB\cc\CC`). The mandatory arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \LaTeX , we can set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

ini files contain the actual data; tex files are currently just proxies to the corresponding ini files. Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.55.2315>>
2 <<date=2021/03/18>>
```

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

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

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

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

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

`\bbl@afterelse` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement³⁰. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

`\bbl@afterfi`

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

\bbl@replace

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

161 \def\bbl@cased{%
162   \ifx\oe\OE
163     \expandafter\in@\expandafter
164       {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166       \bbl@afterelse\expandafter\MakeUppercase
167     \else
168       \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170   \else
171     \expandafter\@firstofone
172   \fi}
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\bbbl@trace[1]{\message{^^J[ #1 ]}}%
196    \let\bbbl@debug\@firstofone
197    \ifx\directlua\@undefined\else
198      \directlua{ Babel = Babel or {} }
```

```

199     Babel.debug = true }%
200   \fi}
201   {\providecommand\bbl@trace[1]{}%
202     \let\bbl@debug\@gobble
203     \ifx\directlua\@undefined\else
204       \directlua{ Babel = Babel or {}
205         Babel.debug = false }%
206     \fi}
207 <<Basic macros>>
208 % Temporarily repeat here the code for errors. TODO.
209 \def\bbl@error#1#2{%
210   \begingroup
211     \def\{\MessageBreak}%
212     \PackageError{babel}{#1}{#2}%
213   \endgroup}
214 \def\bbl@warning#1{%
215   \begingroup
216     \def\{\MessageBreak}%
217     \PackageWarning{babel}{#1}%
218   \endgroup}
219 \def\bbl@infowarn#1{%
220   \begingroup
221     \def\{\MessageBreak}%
222     \GenericWarning
223       {(babel) \@spaces\@spaces\@spaces}%
224       {Package babel Info: #1}%
225   \endgroup}
226 \def\bbl@info#1{%
227   \begingroup
228     \def\{\MessageBreak}%
229     \PackageInfo{babel}{#1}%
230   \endgroup}
231 \def\bbl@nocaption{\protect\bbl@nocaption@i}
232 % TODO - Wrong for \today !!! Must be a separate macro.
233 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
234   \global\@namedef{#2}{\textbf{?#1?}}%
235   \@nameuse{#2}%
236   \edef\bbl@tempa{#1}%
237   \bbl@sreplace\bbl@tempa{name}{}}%
238   \bbl@warning{%
239     \@backslashchar#1 not set for '\language'. Please,\%
240     define it after the language has been loaded\%
241     (typically in the preamble) with\%
242     \string\setlocalecaption{\language}{\bbl@tempa}{..}\%
243     Reported}}
244 \def\bbl@tentative{\protect\bbl@tentative@i}
245 \def\bbl@tentative@i#1{%
246   \bbl@warning{%
247     Some functions for '#1' are tentative.\%
248     They might not work as expected and their behavior\%
249     may change in the future.\%
250     Reported}}
251 \def\@nolanerr#1{%
252   \bbl@error
253     {You haven't defined the language #1\space yet.\%
254     Perhaps you misspelled it or your installation\%
255     is not complete}%
256     {Your command will be ignored, type <return> to proceed}}
257 \def\@nopatterns#1{%

```

```

258 \bbl@warning
259 {No hyphenation patterns were preloaded for\\%
260 the language `#1' into the format.\\%
261 Please, configure your TeX system to add them and\\%
262 rebuild the format. Now I will use the patterns\\%
263 preloaded for \bbl@nulllanguage\space instead}}
264 % End of errors
265 \@ifpackagewith{babel}{silent}
266 {\let\bbl@info@gobble
267 \let\bbl@infowarn@gobble
268 \let\bbl@warning@gobble}
269 {}
270 %
271 \def\AfterBabelLanguage#1{%
272 \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

```

If the format created a list of loaded languages (in `\bbl@languages`), get the name of the 0-th to show the actual language used. Also available with `base`, because it just shows info.

```

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

```

7.3 base

The first ‘real’ option to be processed is `base`, which set the hyphenation patterns then resets `ver@babel.sty` so that \TeX forgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.

Now the `base` option. With it we can define (and load, with `luatex`) hyphenation patterns, even if we are not interested in the rest of `babel`.

```

291 \bbl@trace{Defining option 'base'}
292 \@ifpackagewith{babel}{base}{%
293 \let\bbl@onlyswitch\@empty
294 \let\bbl@provide@locale\relax
295 \input babel.def
296 \let\bbl@onlyswitch\@undefined
297 \ifx\directlua\@undefined
298 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
299 \else
300 \input luababel.def
301 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
302 \fi
303 \DeclareOption{base}{}%
304 \DeclareOption{showlanguages}{}%

```

```

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

```

The next option tells babel to leave shorthand characters active at the end of processing the package. This is *not* the default as it can cause problems with other packages, but for those who want to use the shorthand characters in the preamble of their documents this can help.

```

348 \DeclareOption{KeepShorthandsActive}{}
349 \DeclareOption{activeacute}{}
350 \DeclareOption{activegrave}{}
351 \DeclareOption{debug}{}
352 \DeclareOption{noconfigs}{}
353 \DeclareOption{showlanguages}{}
354 \DeclareOption{silent}{}
355 \DeclareOption{mono}{}
356 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
357 \chardef\bbl@iniflag\z@
358 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
359 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2

```

```

360 \DeclareOption{provide*={}}{\chardef\bbl@iniflag\thr@@} % add + main
361 % A separate option
362 \let\bbl@autoload@options\@empty
363 \DeclareOption{provide*={}}{\def\bbl@autoload@options{import}}
364 % Don't use. Experimental. TODO.
365 \newif\ifbbl@single
366 \DeclareOption{selectors=off}{\bbl@singletrue}
367 <<More package options>>

```

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax <key>=<value>, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```

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

```

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

```

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

```

Now the option list is processed, taking into account only currently declared options (including those declared with a =), and <key>=<value> options (the former take precedence). Unrecognized options are saved in \bbl@language@opts, because they are language options.

```

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

```

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

```

392 \ProcessOptions*

```

7.4 Conditional loading of shorthands

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

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

```

393 \bbl@trace{Conditional loading of shorthands}
394 \def\bbl@sh@string#1{%
395   \ifx#1\@empty\else
396     \ifx#1t\string~%
397     \else\ifx#1c\string,%
398     \else\string#1%

```



```

399   \fi\fi
400   \expandafter\bb1@sh@string
401   \fi}
402 \ifx\bb1@opt@shorthands\@nnil
403   \def\bb1@ifshorthand#1#2#3{#2}%
404 \else\ifx\bb1@opt@shorthands\@empty
405   \def\bb1@ifshorthand#1#2#3{#3}%
406 \else

```

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

```

407   \def\bb1@ifshorthand#1{%
408     \bb1@xin@{\string#1}{\bb1@opt@shorthands}%
409     \ifin@
410       \expandafter\@firstoftwo
411     \else
412       \expandafter\@secondoftwo
413   \fi}

```

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

```

414   \edef\bb1@opt@shorthands{%
415     \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.

```

416   \bb1@ifshorthand{'}%
417     {\PassOptionsToPackage{activeacute}{babel}}{}
418   \bb1@ifshorthand{`}%
419     {\PassOptionsToPackage{activegrave}{babel}}{}
420 \fi\fi

```

With headfoot=lang we can set the language used in heads/foots. For example, in babel/3796 just adds headfoot=english. It misuses \@resetactivechars but seems to work.

```

421 \ifx\bb1@opt@headfoot\@nnil\else
422   \g@addto@macro\@resetactivechars{%
423     \set@typeset@protect
424     \expandafter\select@language@x\expandafter{\bb1@opt@headfoot}%
425     \let\protect\noexpand}
426 \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.

```

427 \ifx\bb1@opt@safe\@undefined
428   \def\bb1@opt@safe{BR}
429 \fi
430 \ifx\bb1@opt@main\@nnil\else
431   \edef\bb1@language@opts{%
432     \ifx\bb1@language@opts\@empty\else\bb1@language@opts,\fi
433     \bb1@opt@main}
434 \fi

```

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

```

435 \bb1@trace{Defining IfBabelLayout}
436 \ifx\bb1@opt@layout\@nnil
437   \newcommand\IfBabelLayout[3]{#3}%
438 \else
439   \newcommand\IfBabelLayout[1]{%
440     \@expandtwoargs\in@{.#1.}{.\bb1@opt@layout.}%
441     \ifin@

```

```

442     \expandafter\@firstoftwo
443     \else
444     \expandafter\@secondoftwo
445     \fi}
446 \fi

```

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

```

447 \input babel.def

```

7.5 Cross referencing macros

The \TeX book states:

The *key* argument is any sequence of letters, digits, and punctuation symbols; upper- and lowercase letters are regarded as different.

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.

When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category ‘letter’ or ‘other’.

The following package options control which macros are to be redefined.

```

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

```

`\@newl@bel` First we open a new group to keep the changed setting of `\protect` local and then we set the `@safe@actives` switch to true to make sure that any shorthand that appears in any of the arguments immediately expands to its non-active self.

```

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

```

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

```

463 \CheckCommand*\@testdef[3]{%
464   \def\reserved@a{#3}%
465   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
466   \else
467     \@tempwattrue
468   \fi}

```

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

```

469 \def\@testdef#1#2#3{% TODO. With @samestring?
470   \@safe@activetrue
471   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname

```

```

472 \def\bbl@tempb{#3}%
473 \@safe@activesfalse
474 \ifx\bbl@tempa\relax
475 \else
476 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
477 \fi
478 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
479 \ifx\bbl@tempa\bbl@tempb
480 \else
481 \@tempswatrue
482 \fi}
483 \fi

```

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

```

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

```

`\@citex` The macro used to cite from a bibliography, `\cite`, uses an internal macro, `\@citex`. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave `\cite` alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.

```

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

```

Unfortunately, the packages `natbib` and `cite` need a different definition of `\@citex`... To begin with, `natbib` has a definition for `\@citex` with *three* arguments... We only know that a package is loaded when `\begin{document}` is executed, so we need to postpone the different redefinition.

```

499 \AtBeginDocument{%
500 \@ifpackageloaded{natbib}{%

```

Notice that we use `\def` here instead of `\bbl@redefine` because `\org@@citex` is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).

(Recent versions of `natbib` change dynamically `\@citex`, so PR4087 doesn't seem fixable in a simple way. Just load `natbib` before.)

```

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

```

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

```

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

```

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

```

510 \bbl@redefine\nocite#1{%
511   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

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

```

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

```

515 \def\bbl@bibcite#1#2{%
516   \org@bibcite{#1}\@safe@activesfalse#2}

```

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

```

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

```

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

```

522 \AtBeginDocument{\bbl@cite@choice}

```

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

```

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

```

7.6 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used.

We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

```

531 \bbl@trace{Marks}
532 \IfBabelLayout{sectioning}
533   {\ifx\bbl@opt@headfoot\@nnil
534     \g@addto@macro\@resetactivechars{%
535       \set@typeset@protect
536       \expandafter\select@language@x\expandafter{\bbl@main@language}%
537       \let\protect\noexpand

```

```

538     \ifcase\bbbl@bidimode\else % Only with bidi. See also above
539     \edef\thepage{%
540         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
541     \fi}%
542 \fi}
543 {\ifbbbl@single\else
544     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
545     \markright#1{%
546         \bbl@ifblank{#1}%
547         {\org@markright{}}}%
548         {\toks@{#1}}%
549         \bbl@exp{%
550             \\org@markright{\\protect\\foreignlanguage{\language}%
551                 {\\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

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

```

7.7 Preventing clashes with other packages

7.7.1 ifthen

`\ifthenelse` Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

```

\ifthenelse{\isodd{\pageref{some:label}}}
{code for odd pages}
{code for even pages}

```

In order for this to work the argument of `\isodd` needs to be fully expandable. With the above redefinition of `\pageref` it is not in the case of this example. To overcome that, we add some code to the definition of `\ifthenelse` to make things work.

We want to revert the definition of `\pageref` and `\ref` to their original definition for the first argument of `\ifthenelse`, so we first need to store their current meanings.

Then we can set the `\@safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments.

```

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

```

7.7.2 varioref

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

```

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

```

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

```

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

```

7.7.3 hhline

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the `‘` character which is made active by the french support in

babel. Therefore we need to *reload* the package when the ‘.’ is an active character. Note that this happens *after* the category code of the @-sign has been changed to other, so we need to temporarily change it to letter again.

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

7.7.4 hyperref

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

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

7.7.5 fancyhdr

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

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

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

```
624 \def\substitutefontfamily#1#2#3{%
625   \lowercase{\immediate\openout15=#1#2.fd\relax}%
626   \immediate\write15{%
627     \string\ProvidesFile{#1#2.fd}%
628     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
629     \space generated font description file]^^J
630     \string\DeclareFontFamily{#1}{#2}{^^J
631     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
632     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
633     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
634     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
635     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
636     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
637     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
638     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
639     }%
640   \closeout15
641   }
642 \@onlypreamble\substitutefontfamily
```

7.8 Encoding and fonts

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

\ensureascii

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

Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at $\text{\begin{document}}$, which latin fontencoding to use.

\latinencoding

When text is being typeset in an encoding other than ‘latin’ (OT1 or T1), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

```
679 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}
```

But this might be overruled with a later loading of the package fontenc. Therefore we check at the execution of $\text{\begin{document}}$ whether it was loaded with the T1 option. The normal way to do this (using \@ifpackageloaded) is disabled for this package. Now we have to revert to parsing the internal macro \@filelist which contains all the filenames loaded.


```

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

```

`\latintext` Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

```

704 \DeclareRobustCommand{\latintext}{%
705   \fontencoding{\latinencoding}\selectfont
706   \def\encodingdefault{\latinencoding}}

```

`\textlatin` This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

```

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

```

7.9 Basic bidi support

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

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

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

- `pdfTeX` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.

- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but `bidi` text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `LuaTeX-jā` shows, vertical typesetting is possible, too.

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

```

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

```

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

```

751 \bbl@trace{Loading basic (internal) bidi support}
752 \ifodd\bbl@engine
753   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
754     \let\bbl@beforeforeign\leavevmode
755     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
756     \RequirePackage{luatexbase}
757     \bbl@activate@preotf
758     \directlua{
759       require('babel-data-bidi.lua')

```

```

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

```

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

```

815 \bbbl@trace{Macros to switch the text direction}

```

```

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

```

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

```

857 \ifodd\bbl@engine % luatex=1
858   \chardef\bbl@thetexmdir\z@
859   \chardef\bbl@thepardir\z@
860   \def\bbl@getluadir#1{%
861     \directlua{
862       if tex.#1dir == 'TLT' then
863         tex.sprint('0')
864       elseif tex.#1dir == 'TRT' then
865         tex.sprint('1')
866       end}}
867   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
868     \ifcase#3\relax
869       \ifcase\bbl@getluadir{#1}\relax\else
870         #2 TLT\relax
871       \fi
872     \else

```

```

873     \ifcase\bb1@getluadir{#1}\relax
874     #2 TRT\relax
875     \fi
876   \fi}
877 \def\bb1@textdir#1{%
878   \bb1@setluadir{text}\textdir{#1}%
879   \chardef\bb1@thetextdir#1\relax
880   \setattribute\bb1@attr@dir{\numexpr\bb1@thepardir*3+#1}}
881 \def\bb1@pardir#1{%
882   \bb1@setluadir{par}\pardir{#1}%
883   \chardef\bb1@thepardir#1\relax}
884 \def\bb1@bodydir{\bb1@setluadir{body}\bodydir}
885 \def\bb1@pagedir{\bb1@setluadir{page}\pagedir}
886 \def\bb1@dirparastext{\pardir\the\textdir\relax}% %%%
887 % Sadly, we have to deal with boxes in math with basic.
888 % Activated every math with the package option bidi=:
889 \def\bb1@mathboxdir{%
890   \ifcase\bb1@thetextdir\relax
891     \everyhbox{\textdir TLT\relax}%
892   \else
893     \everyhbox{\textdir TRT\relax}%
894   \fi}
895 \frozen@everymath\expandafter{%
896   \expandafter\bb1@mathboxdir\the\frozen@everymath}
897 \frozen@everydisplay\expandafter{%
898   \expandafter\bb1@mathboxdir\the\frozen@everydisplay}
899 \else % pdftex=0, xetex=2
900   \newcount\bb1@dirlevel
901   \chardef\bb1@thetextdir\z@
902   \chardef\bb1@thepardir\z@
903   \def\bb1@textdir#1{%
904     \ifcase#1\relax
905       \chardef\bb1@thetextdir\z@
906       \bb1@textdir@i\beginL\endL
907     \else
908       \chardef\bb1@thetextdir\@ne
909       \bb1@textdir@i\beginR\endR
910     \fi}
911   \def\bb1@textdir@i#1#2{%
912     \ifhmode
913       \ifnum\currentgrouplevel>\z@
914         \ifnum\currentgrouplevel=\bb1@dirlevel
915           \bb1@error{Multiple bidi settings inside a group}%
916             {I'll insert a new group, but expect wrong results.}%
917           \bgroup\aftergroup#2\aftergroup\egroup
918         \else
919           \ifcase\currentgrouptype\or % 0 bottom
920             \aftergroup#2% 1 simple {}
921           \or
922             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
923           \or
924             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
925           \or\or\or % vbox vtop align
926           \or
927             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
928           \or\or\or\or\or\or % output math disc insert vcent mathchoice
929           \or
930             \aftergroup#2% 14 \begingroup
931           \else

```

```

932      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
933    \fi
934  \fi
935  \bbl@dirlevel\currentgrouplevel
936  \fi
937  #1%
938  \fi}
939  \def\bbl@pdir#1{\chardef\bbl@thepardir#1\relax}
940  \let\bbl@bodydir@gobble
941  \let\bbl@pagedir@gobble
942  \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).

```

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

```

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

```

969 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
970 \AtBeginDocument{%
971   \ifx\pdfstringdefDisableCommands\undefined\else
972     \ifx\pdfstringdefDisableCommands\relax\else
973       \pdfstringdefDisableCommands{\let\babelsublr@firstofone}%
974     \fi
975   \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`.

```

976 \bbl@trace{Local Language Configuration}

```

```

977 \ifx\loadlocalcfg\undefined
978 \ifpackagewith{babel}{noconfigs}%
979 {\let\loadlocalcfg@gobble}%
980 {\def\loadlocalcfg#1{%
981   \InputIfFileExists{#1.cfg}%
982   {\typeout{*****^J%
983             * Local config file #1.cfg used^^J%
984             *}}}%
985   \@empty}}
986 \fi

```

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

```

987 \ifx\@unexpandable@protect\undefined
988 \def\@unexpandable@protect{\noexpand\protect\noexpand}
989 \long\def\protected@write#1#2#3{%
990   \begingroup
991     \let\thepage\relax
992     #2%
993     \let\protect\@unexpandable@protect
994     \edef\reserved@a{\write#1{#3}}%
995     \reserved@a
996   \endgroup
997   \if@nobreak\ifvmode\nobreak\fi\fi}
998 \fi
999 %
1000 % \subsection{Language options}
1001 %
1002 % Languages are loaded when processing the corresponding option
1003 % \textit{except} if a |main| language has been set. In such a
1004 % case, it is not loaded until all options has been processed.
1005 % The following macro inputs the ldf file and does some additional
1006 % checks (|\input| works, too, but possible errors are not caught).
1007 %
1008 % \begin{macrocode}
1009 \bbl@trace{Language options}
1010 \let\bbl@afterlang\relax
1011 \let\BabelModifiers\relax
1012 \let\bbl@loaded\@empty
1013 \def\bbl@load@language#1{%
1014   \InputIfFileExists{#1.ldf}%
1015   {\edef\bbl@loaded{\CurrentOption
1016     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1017     \expandafter\let\expandafter\bbl@afterlang
1018       \csname\CurrentOption.ldf-h@@k\endcsname
1019     \expandafter\let\expandafter\BabelModifiers
1020       \csname bbl@mod@\CurrentOption\endcsname}%
1021   {\bbl@error{%
1022     Unknown option '\CurrentOption'. Either you misspelled it\\%
1023     or the language definition file \CurrentOption.ldf was not found}}%
1024     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1025     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1026     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

```

1027 \def\bbl@try@load@lang#1#2#3{%
1028   \IfFileExists{\CurrentOption.ldf}%

```

```

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

```

Another way to extend the list of ‘known’ options for babel was to create the file `bblopts.cfg` in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new `.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.

```

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

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `\bbl@language@opts` are assumed to be languages (note this list also contains the language given with `\main`). If not declared above, the names of the option and the file are the same.

```

1058 \let\bbl@tempc\relax
1059 \bbl@foreach\bbl@language@opts{%
1060 \ifcase\bbl@iniflag % Default
1061 \bbl@ifunset{ds@#1}%
1062 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1063 {}}%
1064 \or % provide=*
1065 \@gobble % case 2 same as 1
1066 \or % provide+=*
1067 \bbl@ifunset{ds@#1}%
1068 {\IfFileExists{#1.ldf}{}%
1069 {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}}%
1070 {}}%
1071 \bbl@ifunset{ds@#1}%
1072 {\def\bbl@tempc{#1}%
1073 \DeclareOption{#1}{%
1074 \ifnum\bbl@iniflag>\@ne
1075 \bbl@ldfinit
1076 \babelprovide[import]{#1}%
1077 \bbl@afterldf}%

```



```

1078         \else
1079         \bbl@load@language{#1}%
1080         \fi}}%
1081     {}%
1082 \or      % provide*=*
1083     \def\bbl@tempc{#1}%
1084     \bbl@ifunset{ds@#1}%
1085     {\DeclareOption{#1}{%
1086         \bbl@ldfinit
1087         \babelprovide[import]{#1}%
1088         \bbl@afterldf{}}}%
1089     {}%
1090 \fi}

```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an ldf exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.

```

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

```

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

```

1108 \ifnum\bbl@iniflag=\z@\else
1109     \ifx\bbl@opt@main\@nnil
1110         \ifx\bbl@tempc\relax
1111             \let\bbl@opt@main\bbl@tempb
1112         \else
1113             \let\bbl@opt@main\bbl@tempc
1114         \fi
1115     \fi
1116 \fi
1117 \ifx\bbl@opt@main\@nnil\else
1118     \expandafter
1119     \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1120     \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1121 \fi

```

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

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

```

1122 \def\AfterBabelLanguage#1{%
1123     \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
1124     \DeclareOption*{}
1125 \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.

```

1126 \bbl@trace{Option 'main'}
1127 \ifx\bbl@opt@main\@nnil
1128   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1129   \let\bbl@tempc\@empty
1130   \bbl@for\bbl@tempb\bbl@tempa{%
1131     \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
1132     \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
1133     \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1134     \expandafter\bbl@tempa\bbl@loaded,\@nnil
1135     \ifx\bbl@tempb\bbl@tempc\else
1136       \bbl@warning{%
1137         Last declared language option is '\bbl@tempc',\%
1138         but the last processed one was '\bbl@tempb'.\%
1139         The main language cannot be set as both a global\%
1140         and a package option. Use 'main=\bbl@tempc' as\%
1141         option. Reported}%
1142       \fi
1143     \else
1144       \ifodd\bbl@iniflag % case 1,3
1145         \bbl@ldfinit
1146         \let\CurrentOption\bbl@opt@main
1147         \bbl@exp{\@babelprovide[import,main]{\bbl@opt@main}}
1148         \bbl@afterldf{}%
1149       \else % case 0,2
1150         \chardef\bbl@iniflag\z@ % Force ldf
1151         \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1152         \ExecuteOptions{\bbl@opt@main}
1153         \DeclareOption*{}%
1154         \ProcessOptions*
1155       \fi
1156     \fi
1157   \def\AfterBabelLanguage{%
1158     \bbl@error
1159     {Too late for \string\AfterBabelLanguage}%
1160     {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.

1161 \ifx\bbl@main@language\@undefined
1162   \bbl@info{%
1163     You haven't specified a language. I'll use 'nil'\%
1164     as the main language. Reported}
1165   \bbl@load@language{nil}
1166 \fi
1167 \</package>
1168 \<core>

```

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

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

Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

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

8.1 Tools

```
1169 \ifx\ldf@quit\@undefined\else
1170 \endinput\fi % Same line!
1171 <<Make sure ProvidesFile is defined>>
1172 \ProvidesFile{babel.def}[(<date>)](<version>) Babel common definitions]
```

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

```
1173 \ifx\AtBeginDocument\@undefined % TODO. change test.
1174 <<Emulate LaTeX>>
1175 \def\language{english}%
1176 \let\bbl@opt@shorthands\@nnil
1177 \def\bbl@ifshorthand#1#2#3{#2}%
1178 \let\bbl@language@opts\@empty
1179 \ifx\babeloptionstrings\@undefined
1180 \let\bbl@opt@strings\@nnil
1181 \else
1182 \let\bbl@opt@strings\babeloptionstrings
1183 \fi
1184 \def\BabelStringsDefault{generic}
1185 \def\bbl@tempa{normal}
1186 \ifx\babeloptionmath\bbl@tempa
1187 \def\bbl@mathnormal{\noexpand\textormath}
1188 \fi
1189 \def\AfterBabelLanguage#1#2{}
1190 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1191 \let\bbl@afterlang\relax
1192 \def\bbl@opt@safe{BR}
1193 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1194 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1195 \expandafter\newif\csname ifbbl@single\endcsname
1196 \chardef\bbl@bidimode\z@
1197 \fi
```

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

```
1198 \ifx\bbl@trace\@undefined
1199 \let\ldf@init\endinput
1200 \def\ProvidesLanguage#1{\endinput}
1201 \endinput\fi % Same line!
```

And continue.

9 Multiple languages

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

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.

```
1202 <<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.

```

1203 \def\bbl@version{\langle version\rangle}
1204 \def\bbl@date{\langle date\rangle}
1205 \def\adddialect#1#2{%
1206   \global\chardef#1#2\relax
1207   \bbl@usehooks{adddialect}{\{#1\}{#2\}}%
1208   \begingroup
1209     \count@#1\relax
1210     \def\bbl@elt##1##2##3##4{%
1211       \ifnum\count@=##2\relax
1212         \bbl@info{\string#1 = using hyphenrules for ##1\}%
1213         (\string\language\the\count@)}%
1214       \def\bbl@elt####1####2####3####4{%
1215         \fi}%
1216       \bbl@cs{languages}%
1217     \endgroup

```

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

```

1218 \def\bbl@fixname#1{%
1219   \begingroup
1220   \def\bbl@tempe{l@}%
1221   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1222   \bbl@tempd
1223     {\lowercase\expandafter{\bbl@tempd}%
1224     {\uppercase\expandafter{\bbl@tempd}%
1225     \@empty
1226     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1227     \uppercase\expandafter{\bbl@tempd}}}%
1228     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1229     \lowercase\expandafter{\bbl@tempd}}}%
1230   \@empty
1231   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1232   \bbl@tempd
1233   \bbl@exp{\bbl@usehooks{language}{\{#1\}}}
1234 \def\bbl@iflanguage#1{%
1235   \@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`.

```

1236 \def\bbl@bcpcase#1#2#3#4\@#5{%
1237   \ifx\@empty#3%
1238     \uppercase{\def#5{#1#2}}%
1239   \else
1240     \uppercase{\def#5{#1}}%
1241     \lowercase{\def#5{#5#2#3#4}}%
1242   \fi}
1243 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1244   \let\bbl@bcp\relax
1245   \lowercase{\def\bbl@tempa{#1}}%

```

```

1246 \ifx\@empty#2%
1247 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1248 \else\ifx\@empty#3%
1249 \bbl@bcp#2\@empty\@empty\@{\bbl@tempb
1250 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1251 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1252 {}%
1253 \ifx\bbl@bcp\relax
1254 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1255 \fi
1256 \else
1257 \bbl@bcp#2\@empty\@empty\@{\bbl@tempb
1258 \bbl@bcp#3\@empty\@empty\@{\bbl@tempc
1259 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1260 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1261 {}%
1262 \ifx\bbl@bcp\relax
1263 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1264 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1265 {}%
1266 \fi
1267 \ifx\bbl@bcp\relax
1268 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1269 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1270 {}%
1271 \fi
1272 \ifx\bbl@bcp\relax
1273 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1274 \fi
1275 \fi\fi}
1276 \let\bbl@initoload\relax
1277 \def\bbl@provide@locale{%
1278 \ifx\babelprovide\@undefined
1279 \bbl@error{For a language to be defined on the fly 'base'\\%
1280 is not enough, and the whole package must be\\%
1281 loaded. Either delete the 'base' option or\\%
1282 request the languages explicitly}%
1283 {See the manual for further details.}%
1284 \fi
1285 % TODO. Option to search if loaded, with \LocaleForEach
1286 \let\bbl@auxname\language\name % Still necessary. TODO
1287 \bbl@ifunset{\bbl@bcp@map@\language\name}{}% Move uplevel??
1288 {\edef\language\name{\@nameuse{\bbl@bcp@map@\language\name}}}%
1289 \if\bbl@bcp@allowed
1290 \expandafter\ifx\csname date\language\name\endcsname\relax
1291 \expandafter
1292 \bbl@bcp@lookup\language\name-\@empty-\@empty-\@empty\@{\
1293 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
1294 \edef\language\name{\bbl@bcp@prefix\bbl@bcp}%
1295 \edef\locale\name{\bbl@bcp@prefix\bbl@bcp}%
1296 \expandafter\ifx\csname date\language\name\endcsname\relax
1297 \let\bbl@initoload\bbl@bcp
1298 \bbl@exp{\babelprovide[\bbl@autoload@bcpoptions]{\language\name}}}%
1299 \let\bbl@initoload\relax
1300 \fi
1301 \bbl@csarg\xdef{\bcp@map@\bbl@bcp}{\locale\name}%
1302 \fi
1303 \fi
1304 \fi

```

```

1305 \expandafter\ifx\csname date\language\endcsname\relax
1306 \IfFileExists{babel-\language.tex}%
1307 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
1308 {}%
1309 \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.

```

1310 \def\iflanguage#1{%
1311 \bbl@iflanguage{#1}{%
1312 \ifnum\csname l@#1\endcsname=\language
1313 \expandafter\@firstoftwo
1314 \else
1315 \expandafter\@secondoftwo
1316 \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.

```

1317 \let\bbl@select@type\z@
1318 \edef\selectlanguage{%
1319 \noexpand\protect
1320 \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`.

```
1321 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1322 \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.

```
1323 \def\bbl@language@stack{}
```

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

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

```

1324 \def\bbl@push@language{%
1325 \ifx\language\@undefined\else
1326 \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1327 \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`.

```
1328 \def\bbl@pop@lang#1+#2\@@{%
1329   \edef\language{#1}%
1330   \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).

```
1331 \let\bbl@ifrestoring\@secondoftwo
1332 \def\bbl@pop@language{%
1333   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1334   \let\bbl@ifrestoring\@firstoftwo
1335   \expandafter\bbl@set@language\expandafter{\language}%
1336   \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).

```
1337 \chardef\localeid\z@
1338 \def\bbl@id@last{0} % No real need for a new counter
1339 \def\bbl@id@assign{%
1340   \bbl@ifunset{bbl@id@@\language}%
1341   {\count@bbl@id@last\relax
1342     \advance\count@\@ne
1343     \bbl@csarg\chardef{id@@\language}\count@
1344     \edef\bbl@id@last{\the\count@}%
1345     \ifcase\bbl@engine\or
1346       \directlua{
1347         Babel = Babel or {}
1348         Babel.locale_props = Babel.locale_props or {}
1349         Babel.locale_props[\bbl@id@last] = {}
1350         Babel.locale_props[\bbl@id@last].name = '\language'
1351       }%
1352     \fi}%
1353   }%
1354   \chardef\localeid\bbl@c{l{id@}}}
```

The unprotected part of `\selectlanguage`.

```
1355 \expandafter\def\csname selectlanguage \endcsname#1{%
1356   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@fi
1357   \bbl@push@language
1358   \aftergroup\bbl@pop@language
1359   \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.

```

1360 \def\BabelContentsFiles{toc,lof,lot}
1361 \def\bbl@set@language#1{% from selectlanguage, pop@
1362 % The old buggy way. Preserved for compatibility.
1363 \edef\language{%
1364   \ifnum\escapechar=\expandafter`\string#1\@empty
1365   \else\string#1\@empty\fi}%
1366 \ifcat\relax\noexpand#1%
1367   \expandafter\ifx\csname date\language\endcsname\relax
1368     \edef\language{#1}%
1369     \let\localname\language
1370   \else
1371     \bbl@info{Using '\string\language' instead of 'language' is\\%
1372       deprecated. If what you want is to use a\\%
1373       macro containing the actual locale, make\\%
1374       sure it does not not match any language.\\%
1375       Reported}%
1376     I'll\\%
1377     try to fix '\string\localname', but I cannot promise\\%
1378     anything. Reported}%
1379   \ifx\scantokens\@undefined
1380     \def\localname{??}%
1381   \else
1382     \scantokens\expandafter{\expandafter
1383       \def\expandafter\localname\expandafter{\language}}%
1384   \fi
1385 \fi
1386 \else
1387   \def\localname{#1}% This one has the correct catcodes
1388 \fi
1389 \select@language{\language}%
1390 % write to aux
1391 \expandafter\ifx\csname date\language\endcsname\relax\else
1392   \if@filesw
1393     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1394       % \bbl@savelastskip
1395       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1396       % \bbl@restorelastskip
1397     \fi
1398     \bbl@usehooks{write}{}%
1399   \fi
1400 \fi}
1401 % The following is used above to deal with skips before the write
1402 % whatsit. Adapted from hyperref, but it might fail, so for the moment
1403 % it's not activated. TODO.
1404 \def\bbl@savelastskip{%
1405   \let\bbl@restorelastskip\relax
1406   \ifvmode
1407     \ifdim\lastskip=\z@
1408       \let\bbl@restorelastskip\nobreak
1409     \else
1410       \bbl@exp{%
1411         \def\\bbl@restorelastskip{%
1412           \skip@=\the\lastskip
1413           \\nobreak \vskip-\skip@ \vskip\skip@}}%

```



```

1414 \fi
1415 \fi}
1416 \newif\ifbbl@bcpallowed
1417 \bbl@bcpallowedfalse
1418 \def\select@language#1{% from set@, babel@aux
1419 % set hymap
1420 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1421 % set name
1422 \edef\language#1}%
1423 \bbl@fixname\language
1424 % TODO. name@map must be here?
1425 \bbl@provide@locale
1426 \bbl@iflanguage\language{%
1427 \expandafter\ifx\csname date\language\endcsname\relax
1428 \bbl@error
1429 {Unknown language '\language'. Either you have\\%
1430 misspelled its name, it has not been installed,\\%
1431 or you requested it in a previous run. Fix its name,\\%
1432 install it or just rerun the file, respectively. In\\%
1433 some cases, you may need to remove the aux file}%
1434 {You may proceed, but expect wrong results}%
1435 \else
1436 % set type
1437 \let\bbl@select@type\z@
1438 \expandafter\bbl@switch\expandafter{\language}%
1439 \fi}}
1440 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1441 \select@language{#1}%
1442 \bbl@foreach\BabelContentsFiles{%
1443 \@writefile{##1}{\babel@toc{#1}{#2}}}% %% TODO - ok in plain?
1444 \def\babel@toc#1#2{%
1445 \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.

```

1446 \newif\ifbbl@usedategroup
1447 \def\bbl@switch#1{% from select@, foreign@
1448 % make sure there is info for the language if so requested
1449 \bbl@ensureinfo{#1}%
1450 % restore
1451 \originalTeX
1452 \expandafter\def\expandafter\originalTeX\expandafter{%
1453 \csname noextras#1\endcsname
1454 \let\originalTeX\@empty
1455 \babel@beginsave}%
1456 \bbl@usehooks{afterreset}}}%
1457 \languageshorthands{none}%
1458 % set the locale id
1459 \bbl@id@assign

```

```

1460 % switch captions, date
1461 % No text is supposed to be added here, so we remove any
1462 % spurious spaces.
1463 \bbl@bsphack
1464 \ifcase\bbl@select@type
1465   \csname captions#1\endcsname\relax
1466   \csname date#1\endcsname\relax
1467 \else
1468   \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1469   \ifin@
1470     \csname captions#1\endcsname\relax
1471   \fi
1472   \bbl@xin@{,date,}{, \bbl@select@opts,}%
1473   \ifin@ % if \foreign... within \<lang>date
1474     \csname date#1\endcsname\relax
1475   \fi
1476 \fi
1477 \bbl@esphack
1478 % switch extras
1479 \bbl@usehooks{beforeextras}{}%
1480 \csname extras#1\endcsname\relax
1481 \bbl@usehooks{afterextras}{}%
1482 % > babel-ensure
1483 % > babel-sh-<short>
1484 % > babel-bidi
1485 % > babel-fontspec
1486 % hyphenation - case mapping
1487 \ifcase\bbl@opt@hyphenmap\or
1488   \def\BabelLower##1##2{\lcode##1=##2\relax}%
1489   \ifnum\bbl@hymapsel>4\else
1490     \csname\language @bbl@hyphenmap\endcsname
1491   \fi
1492   \chardef\bbl@opt@hyphenmap\z@
1493 \else
1494   \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1495     \csname\language @bbl@hyphenmap\endcsname
1496   \fi
1497 \fi
1498 \let\bbl@hymapsel@cclv
1499 % hyphenation - select patterns
1500 \bbl@patterns{#1}%
1501 % hyphenation - allow stretching with babelnohyphens
1502 \ifnum\language=\l@babelnohyphens
1503   \babel@savevariable\emergencystretch
1504   \emergencystretch\maxdimen
1505   \babel@savevariable\hbadness
1506   \hbadness\@M
1507 \fi
1508 % hyphenation - mins
1509 \babel@savevariable\lefthyphenmin
1510 \babel@savevariable\righthyphenmin
1511 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1512   \set@hyphenmins\tw@\thr@\relax
1513 \else
1514   \expandafter\expandafter\expandafter\set@hyphenmins
1515     \csname #1hyphenmins\endcsname\relax
1516 \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.

```
1517 \long\def\otherlanguage#1{%
1518   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1519   \csname selectlanguage \endcsname{#1}%
1520   \ignorespaces}
```

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

```
1521 \long\def\endotherlanguage{%
1522   \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`.

```
1523 \expandafter\def\csname otherlanguage*\endcsname{%
1524   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1525 \def\bbl@otherlanguage@s[#1]#2{%
1526   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1527   \def\bbl@select@opts{#1}%
1528   \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”.

```
1529 \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`.

```
1530 \providecommand\bbl@beforeforeign{}
1531 \edef\foreignlanguage{%
1532   \noexpand\protect
1533   \expandafter\noexpand\csname foreignlanguage \endcsname}
1534 \expandafter\def\csname foreignlanguage \endcsname{%
1535   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1536 \providecommand\bbl@foreign@x[3][]{%
1537   \begin{group}
```

```

1538 \def\bbl@select@opts{#1}%
1539 \let\BabelText\@firstofone
1540 \bbl@beforeforeign
1541 \foreign@language{#2}%
1542 \bbl@usehooks{foreign}{}%
1543 \BabelText{#3}% Now in horizontal mode!
1544 \endgroup}
1545 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
1546 \begingroup
1547 {\par}%
1548 \let\bbl@select@opts\@empty
1549 \let\BabelText\@firstofone
1550 \foreign@language{#1}%
1551 \bbl@usehooks{foreign*}{}%
1552 \bbl@dirparastext
1553 \BabelText{#2}% Still in vertical mode!
1554 {\par}%
1555 \endgroup}

```

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

```

1556 \def\foreign@language#1{%
1557 % set name
1558 \edef\language#1}%
1559 \ifbbl@usedategroup
1560 \bbl@add\bbl@select@opts{,date,}%
1561 \bbl@usedategroupfalse
1562 \fi
1563 \bbl@fixname\language
1564 % TODO. name@map here?
1565 \bbl@provide@locale
1566 \bbl@iflanguage\language{%
1567 \expandafter\ifx\csname date\language\endcsname\relax
1568 \bbl@warning % TODO - why a warning, not an error?
1569 {Unknown language `#1'. Either you have\\%
1570 misspelled its name, it has not been installed,\\%
1571 or you requested it in a previous run. Fix its name,\\%
1572 install it or just rerun the file, respectively. In\\%
1573 some cases, you may need to remove the aux file.\\%
1574 I'll proceed, but expect wrong results.\\%
1575 Reported}%
1576 \fi
1577 % set type
1578 \let\bbl@select@type\@ne
1579 \expandafter\bbl@switch\expandafter{\language}}

```

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

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

```

1580 \let\bbl@hyphlist\@empty
1581 \let\bbl@hyphenation@relax
1582 \let\bbl@pttnlist\@empty

```

```

1583 \let\bbl@patterns@\relax
1584 \let\bbl@hymapsel=\@cc1v
1585 \def\bbl@patterns#1{%
1586   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1587     \csname l@#1\endcsname
1588     \edef\bbl@tempa{#1}%
1589   \else
1590     \csname l@#1:\f@encoding\endcsname
1591     \edef\bbl@tempa{#1:\f@encoding}%
1592   \fi
1593   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1594 % > luatex
1595 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1596   \begingroup
1597     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1598   \ifin@else
1599     \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
1600   \hyphenation{%
1601     \bbl@hyphenation@
1602     \@ifundefined{bbl@hyphenation@#1}%
1603     \@empty
1604     {\space\csname bbl@hyphenation@#1\endcsname}}%
1605   \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1606   \fi
1607   \endgroup}}

```

hyphenrules The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change `\language` and when the hyphenation rules specified were not loaded it has no effect. Note however, `\lccode`'s and font encodings are not set at all, so in most cases you should use `otherlanguage*`.

```

1608 \def\hyphenrules#1{%
1609   \edef\bbl@tempf{#1}%
1610   \bbl@fixname\bbl@tempf
1611   \bbl@iflanguage\bbl@tempf{%
1612     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1613     \ifx\languageshorthands\undefined\else
1614       \languageshorthands{none}%
1615     \fi
1616     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1617       \set@hyphenmins\tw@\thr@@\relax
1618     \else
1619       \expandafter\expandafter\expandafter\set@hyphenmins
1620       \csname\bbl@tempf hyphenmins\endcsname\relax
1621     \fi}}
1622 \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.

```

1623 \def\providehyphenmins#1#2{%
1624   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1625     \namedef{#1hyphenmins}{#2}%
1626   \fi}

```

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

```

1627 \def\set@hyphenmins#1#2{%
1628   \lefthyphenmin#1\relax
1629   \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.

```

1630 \ifx\ProvidesFile\@undefined
1631   \def\ProvidesLanguage#1[#2 #3 #4]{%
1632     \wlog{Language: #1 #4 #3 <#2>}%
1633   }
1634 \else
1635   \def\ProvidesLanguage#1{%
1636     \begingroup
1637       \catcode\ 10 %
1638       \@makeother\/%
1639       \@ifnextchar[%]
1640         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1641   \def\@provideslanguage#1[#2]{%
1642     \wlog{Language: #1 #2}%
1643     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1644     \endgroup}
1645 \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`.

```

1646 \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.

```

1647 \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’:

```

1648 \providecommand\setlocale{%
1649   \bbl@error
1650   {Not yet available}%
1651   {Find an armchair, sit down and wait}}
1652 \let\uselocale\setlocale
1653 \let\locale\setlocale
1654 \let\selectlocale\setlocale
1655 \let\localename\setlocale
1656 \let\textlocale\setlocale
1657 \let\textlanguage\setlocale
1658 \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_ϵ , 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.

```

1659 \edef\bbl@nulllanguage{\string\language=0}
1660 \ifx\PackageError\@undefined % TODO. Move to Plain
1661   \def\bbl@error#1#2{%

```

```

1662 \begingroup
1663 \newlinechar=`^^J
1664 \def\{^^J(babel) }%
1665 \errhelp{#2}\errmessage{\#1}%
1666 \endgroup}
1667 \def\bbl@warning#1{%
1668 \begingroup
1669 \newlinechar=`^^J
1670 \def\{^^J(babel) }%
1671 \message{\#1}%
1672 \endgroup}
1673 \let\bbl@infowarn\bbl@warning
1674 \def\bbl@info#1{%
1675 \begingroup
1676 \newlinechar=`^^J
1677 \def\{^^J}%
1678 \wlog{#1}%
1679 \endgroup}
1680 \fi
1681 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1682 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1683 \global\@namedef{#2}{\textbf{?#1?}}%
1684 \@nameuse{#2}%
1685 \edef\bbl@tempa{#1}%
1686 \bbl@sreplace\bbl@tempa{name}{}%
1687 \bbl@warning{% TODO.
1688 \@backslashchar#1 not set for '\language'. Please,\%
1689 define it after the language has been loaded\%
1690 (typically in the preamble) with:\%
1691 \string\setlocalecaption{\language}{\bbl@tempa}{..\%
1692 Reported}}
1693 \def\bbl@tentative{\protect\bbl@tentative@i}
1694 \def\bbl@tentative@i#1{%
1695 \bbl@warning{%
1696 Some functions for '#1' are tentative.\%
1697 They might not work as expected and their behavior\%
1698 could change in the future.\%
1699 Reported}}
1700 \def\@nolanerr#1{%
1701 \bbl@error
1702 {You haven't defined the language #1\space yet.\%
1703 Perhaps you misspelled it or your installation\%
1704 is not complete}%
1705 {Your command will be ignored, type <return> to proceed}}
1706 \def\@nopatterns#1{%
1707 \bbl@warning
1708 {No hyphenation patterns were preloaded for\%
1709 the language `#1' into the format.\%
1710 Please, configure your TeX system to add them and\%
1711 rebuild the format. Now I will use the patterns\%
1712 preloaded for \bbl@nulllanguage\space instead}}
1713 \let\bbl@usehooks\@gobbletwo
1714 \ifx\bbl@onlyswitch\@empty\endinput\fi
1715 % Here ended switch.def

Here ended switch.def.

1716 \ifx\directlua\@undefined\else
1717 \ifx\bbl@luapatterns\@undefined
1718 \input luababel.def

```

```

1719 \fi
1720 \fi
1721 <<Basic macros>>
1722 \bbl@trace{Compatibility with language.def}
1723 \ifx\bbl@languages\undefined
1724 \ifx\directlua\undefined
1725 \openin1 = language.def % TODO. Remove hardcoded number
1726 \ifeof1
1727 \closein1
1728 \message{I couldn't find the file language.def}
1729 \else
1730 \closein1
1731 \begingroup
1732 \def\addlanguage#1#2#3#4#5{%
1733 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1734 \global\expandafter\let\csname l@#1\expandafter\endcsname
1735 \csname lang@#1\endcsname
1736 \fi}%
1737 \def\uselanguage#1{%
1738 \input language.def
1739 \endgroup
1740 \fi
1741 \fi
1742 \chardef\l@english\z@
1743 \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.

```

1744 \def\addto#1#2{%
1745 \ifx#1\undefined
1746 \def#1{#2}%
1747 \else
1748 \ifx#1\relax
1749 \def#1{#2}%
1750 \else
1751 {\toks@\expandafter{#1#2}%
1752 \xdef#1{the\toks@}}%
1753 \fi
1754 \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?

```

1755 \def\bbl@withactive#1#2{%
1756 \begingroup
1757 \lccode`~=#2\relax
1758 \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 TeX 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.

```

1759 \def\bbl@redefine#1{%
1760 \edef\bbl@tempa{\bbl@stripslash#1}%
1761 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1762 \expandafter\def\csname\bbl@tempa\endcsname{
1763 \@onlypreamble\bbl@redefine

```


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

```
1764 \def\bbl@redefine@long#1{%
1765   \edef\bbl@tempa{\bbl@stripslash#1}%
1766   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1767   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1768 \@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_`.

```
1769 \def\bbl@redefineroobust#1{%
1770   \edef\bbl@tempa{\bbl@stripslash#1}%
1771   \bbl@ifunset{\bbl@tempa\space}%
1772     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1773      \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1774     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1775     \@namedef{\bbl@tempa\space}}
1776 \@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.

```
1777 \bbl@trace{Hooks}
1778 \newcommand\AddBabelHook[3][{}]{%
1779   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1780   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1781   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1782   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1783     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1784     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1785   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1786 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1787 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1788 \def\bbl@usehooks#1#2{%
1789   \def\bbl@elth##1{%
1790     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}{#2}}}%
1791   \bbl@cs{ev@#1@}%
1792   \ifx\language\@undefined\else % Test required for Plain (?)
1793     \def\bbl@elth##1{%
1794       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}{#2}}}%
1795     \bbl@cl{ev@#1}%
1796   \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).

```
1797 \def\bbl@evargs{% <- don't delete this comma
1798   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1799   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1800   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1801   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1802   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.

```

1803 \bbl@trace{Defining babelensure}
1804 \newcommand\babelensure[2][{}]{% TODO - revise test files
1805   \AddBabelHook{babel-ensure}{afterextras}{%
1806     \ifcase\bbl@select@type
1807       \bbl@cl{e}%
1808       \fi}%
1809   \begingroup
1810     \let\bbl@ens@include\@empty
1811     \let\bbl@ens@exclude\@empty
1812     \def\bbl@ens@fontenc{\relax}%
1813     \def\bbl@tempb##1{%
1814       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1815     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1816     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1817     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1818     \def\bbl@tempc{\bbl@ensure}%
1819     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1820       \expandafter{\bbl@ens@include}}%
1821     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1822       \expandafter{\bbl@ens@exclude}}%
1823     \toks@\expandafter{\bbl@tempc}%
1824     \bbl@exp{%
1825       \endgroup
1826       \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1827   \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1828     \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1829       \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1830         \edef##1{\noexpand\bbl@nocaption
1831           {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1832         \fi
1833         \ifx##1\@empty\else
1834           \in@{##1}{#2}%
1835           \ifin@ \else
1836             \bbl@ifunset{\bbl@ensure@\language\name}%
1837             {\bbl@exp{%
1838               \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1839                 \\\foreignlanguage{\language\name}%
1840                 {\ifx\relax#3\else
1841                   \\\fontencoding{#3}\selectfont
1842                   \fi
1843                 #####1}}}%
1844             }%
1845             \toks@\expandafter{##1}%
1846             \edef##1{%
1847               \bbl@csarg\noexpand{ensure@\language\name}%
1848               {\the\toks@}}%
1849             \fi
1850             \expandafter\bbl@tempb
1851             \fi}%
1852     \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1853     \def\bbl@tempa##1{% elt for include list

```

```

1854 \ifx##1\@empty\else
1855 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1856 \ifin\else
1857 \bbl@tempb##1\@empty
1858 \fi
1859 \expandafter\bbl@tempa
1860 \fi}%
1861 \bbl@tempa#1\@empty}
1862 \def\bbl@captionslist{%
1863 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1864 \contentsname\listfigurename\listtablename\indexname\figurename
1865 \tablename\partname\encname\ccname\headtoname\pagename\seename
1866 \alsoname\proofname\glossaryname}

```

9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1867 \bbl@trace{Macros for setting language files up}
1868 \def\bbl@ldfinit{%
1869 \let\bbl@screset\@empty
1870 \let\BabelStrings\bbl@opt@string
1871 \let\BabelOptions\@empty
1872 \let\BabelLanguages\relax
1873 \ifx\originalTeX\undefined
1874 \let\originalTeX\@empty
1875 \else
1876 \originalTeX
1877 \fi}
1878 \def\LdfInit#1#2{%
1879 \chardef\atcatcode=\catcode`\@
1880 \catcode`\@=11\relax
1881 \chardef\eqcatcode=\catcode`\=
1882 \catcode`\==12\relax
1883 \expandafter\if\expandafter\@backslashchar
1884 \expandafter\@car\string#2\@nil
1885 \ifx#2\undefined\else
1886 \ldf@quit{#1}%
1887 \fi
1888 \else
1889 \expandafter\ifx\csname#2\endcsname\relax\else
1890 \ldf@quit{#1}%

```

```

1891 \fi
1892 \fi
1893 \bbl@ldfinit}

```

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

```

1894 \def\ldf@quit#1{%
1895 \expandafter\main@language\expandafter{#1}%
1896 \catcode\@=\atcatcode \let\atcatcode\relax
1897 \catcode\==\eqcatcode \let\eqcatcode\relax
1898 \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.

```

1899 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1900 \bbl@afterlang
1901 \let\bbl@afterlang\relax
1902 \let\BabelModifiers\relax
1903 \let\bbl@screset\relax}%
1904 \def\ldf@finish#1{%
1905 \ifx\loadlocalcfg\undefined\else % For LaTeX 209
1906 \loadlocalcfg{#1}%
1907 \fi
1908 \bbl@afterldf{#1}%
1909 \expandafter\main@language\expandafter{#1}%
1910 \catcode\@=\atcatcode \let\atcatcode\relax
1911 \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 `ℒTEX`.

```

1912 \@onlypreamble\LdfInit
1913 \@onlypreamble\ldf@quit
1914 \@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.

```

1915 \def\main@language#1{%
1916 \def\bbl@main@language{#1}%
1917 \let\language\name\bbl@main@language % TODO. Set localname
1918 \bbl@id@assign
1919 \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`.

```

1920 \def\bbl@beforestart{%
1921 \bbl@usehooks{beforestart}}}%
1922 \global\let\bbl@beforestart\relax}
1923 \AtBeginDocument{%
1924 \@nameuse{bbl@beforestart}%
1925 \if@files
1926 \providecommand\babel@aux[2]{}%
1927 \immediate\write\@mainaux{%
1928 \string\providecommand\string\babel@aux[2]{}%
1929 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1930 \fi

```

```

1931 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1932 \ifbbl@single % must go after the line above.
1933 \renewcommand\selectlanguage[1]{}%
1934 \renewcommand\foreignlanguage[2]{#2}%
1935 \global\let\babel@aux\@gobbletwo % Also as flag
1936 \fi
1937 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1938 \def\select@language#1{%
1939 \ifcase\bbl@select@type
1940 \bbl@ifsamestring\language#1\fi\select@language{#1}%
1941 \else
1942 \select@language{#1}%
1943 \fi}

```

9.5 Shorthands

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

```

1944 \bbl@trace{Shorhands}
1945 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1946 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1947 \bbl@ifunset{@sanitize}\fi\bbl@add\@sanitize{\@makeother#1}%
1948 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1949 \begingroup
1950 \catcode`#1\active
1951 \nfss@catcodes
1952 \ifnum\catcode`#1=\active
1953 \endgroup
1954 \bbl@add\nfss@catcodes{\@makeother#1}%
1955 \else
1956 \endgroup
1957 \fi
1958 \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.

```

1959 \def\bbl@remove@special#1{%
1960 \begingroup
1961 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1962 \else\noexpand##1\noexpand##2\fi}%
1963 \def\do{\x\do}%
1964 \def\@makeother{\x\@makeother}%
1965 \edef\x{\endgroup
1966 \def\noexpand\dospecials{\dospecials}%
1967 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1968 \def\noexpand\@sanitize{\@sanitize}%
1969 \fi}%
1970 \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).

```

1971 \def\bbl@active@def#1#2#3#4{%
1972   \@namedef{#3#1}{%
1973     \expandafter\ifx\csname#2@sh@#1@endcsname\relax
1974       \bbl@afterelse\bbl@sh@select#2#1{#3#arg#1}{#4#1}%
1975     \else
1976       \bbl@afterfi\csname#2@sh@#1@endcsname
1977     \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.

```

1978   \long\@namedef{#3#arg#1}##1{%
1979     \expandafter\ifx\csname#2@sh@#1@string##1@endcsname\relax
1980       \bbl@afterelse\csname#4#1@endcsname##1%
1981     \else
1982       \bbl@afterfi\csname#2@sh@#1@string##1@endcsname
1983     \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.

```

1984 \def\initiate@active@char#1{%
1985   \bbl@ifunset{active@char\string#1}%
1986   {\bbl@withactive
1987     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1988   {}}

```

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`).

```

1989 \def\@initiate@active@char#1#2#3{%
1990   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1991   \ifx#1\@undefined
1992     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1993   \else
1994     \bbl@csarg\let{oridef@#2}#1%
1995     \bbl@csarg\edef{oridef@#2}{%
1996       \let\noexpand#1%
1997       \expandafter\noexpand\csname bbl@oridef@#2@endcsname}%
1998   \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`).

```

1999 \ifx#1#3\relax
2000 \expandafter\let\csname normal@char#2\endcsname#3%
2001 \else
2002 \bbl@info{Making #2 an active character}%
2003 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2004 \@namedef{normal@char#2}{%
2005 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
2006 \else
2007 \@namedef{normal@char#2}{#3}%
2008 \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).

```

2009 \bbl@restoreactive{#2}%
2010 \AtBeginDocument{%
2011 \catcode`#2\active
2012 \if@filesw
2013 \immediate\write\@mainaux{\catcode`\string#2\active}%
2014 \fi}%
2015 \expandafter\bbl@add@special\csname#2\endcsname
2016 \catcode`#2\active
2017 \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}).

```

2018 \let\bbl@tempa\@firstoftwo
2019 \if\string^#2%
2020 \def\bbl@tempa{\noexpand\textormath}%
2021 \else
2022 \ifx\bbl@mathnormal\@undefined\else
2023 \let\bbl@tempa\bbl@mathnormal
2024 \fi
2025 \fi
2026 \expandafter\edef\csname active@char#2\endcsname{%
2027 \bbl@tempa
2028 {\noexpand\if@safe@actives
2029 \noexpand\expandafter
2030 \expandafter\noexpand\csname normal@char#2\endcsname
2031 \noexpand\else
2032 \noexpand\expandafter
2033 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2034 \noexpand\fi}%
2035 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2036 \bbl@csarg\edef{doactive#2}{%
2037 \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!).

```

2038 \bbl@csarg\edef{active@#2}{%

```

```

2039 \noexpand\active@prefix\noexpand#1%
2040 \expandafter\noexpand\csname active@char#2\endcsname}%
2041 \bbl@csarg\edef{normal@#2}{%
2042 \noexpand\active@prefix\noexpand#1%
2043 \expandafter\noexpand\csname normal@char#2\endcsname}%
2044 \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.

```

2045 \bbl@active@def#2\user@group{user@active}{language@active}%
2046 \bbl@active@def#2\language@group{language@active}{system@active}%
2047 \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.

```

2048 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2049 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2050 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
2051 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

2052 \if\string'#2%
2053 \let\prim@s\bbl@prim@s
2054 \let\active@math@prime#1%
2055 \fi
2056 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}

```

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

```

2057 <<(*More package options)>> ≡
2058 \DeclareOption{math=active}{}
2059 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
2060 <</More package options>>

```

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package *and* the end of the ldf.

```

2061 \@ifpackagewith{babel}{KeepShorthandsActive}%
2062 {\let\bbl@restoreactive\@gobble}%
2063 {\def\bbl@restoreactive#1{%
2064 \bbl@exp{%
2065 \\\AfterBabelLanguage\\CurrentOption
2066 {\catcode`#1=\the\catcode`#1\relax}%
2067 \\\AtEndOfPackage
2068 {\catcode`#1=\the\catcode`#1\relax}}}%
2069 \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.


```

2070 \def\bbl@sh@select#1#2{%
2071   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2072   \bbl@afterelse\bbl@scndcs
2073   \else
2074   \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2075   \fi}

```

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

```

2076 \begingroup
2077 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2078 {\gdef\active@prefix#1{%
2079   \ifx\protect\@typeset@protect
2080   \else
2081   \ifx\protect\@unexpandable@protect
2082   \noexpand#1%
2083   \else
2084   \protect#1%
2085   \fi
2086   \expandafter\@gobble
2087   \fi}}
2088 {\gdef\active@prefix#1{%
2089   \ifincsname
2090   \string#1%
2091   \expandafter\@gobble
2092   \else
2093   \ifx\protect\@typeset@protect
2094   \else
2095   \ifx\protect\@unexpandable@protect
2096   \noexpand#1%
2097   \else
2098   \protect#1%
2099   \fi
2100   \expandafter\expandafter\expandafter\@gobble
2101   \fi
2102   \fi}}
2103 \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⟩`.

```

2104 \newif\if@safe@actives
2105 \@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.

```

2106 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

`\bbl@activate` Both macros take one argument, like `\initiate@active@char`. The macro is used to change the definition of an active character to expand to `\active@char⟨char⟩` in the case of `\bbl@activate`, or `\normal@char⟨char⟩` in the case of `\bbl@deactivate`.

```

2107 \def\bbl@activate#1{%
2108   \bbl@withactive{\expandafter\let\expandafter}#1%
2109   \csname bbl@active@\string#1\endcsname}

```

```

2110 \def\bbl@deactivate#1{%
2111   \bbl@withactive{\expandafter\let\expandafter}#1%
2112   \csname bbl@normal@\string#1\endcsname}

\bbl@firstcs  These macros are used only as a trick when declaring shorthands.
\bbl@scndcs  2113 \def\bbl@firstcs#1#2{\csname#1\endcsname}
               2114 \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.

2115 \def\babel@texpdf#1#2#3#4{%
2116   \ifx\texorpdfstring\undefined
2117     \textormath{#1}{#2}%
2118   \else
2119     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2120     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2121   \fi}
2122 %
2123 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2124 \def\@decl@short#1#2#3\@nil#4{%
2125   \def\bbl@tempa{#3}%
2126   \ifx\bbl@tempa\@empty
2127     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2128     \bbl@ifunset{#1@sh@\string#2@}{}%
2129     {\def\bbl@tempa{#4}%
2130      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2131      \else
2132        \bbl@info
2133          {Redefining #1 shorthand \string#2\\
2134           in language \CurrentOption}%
2135      \fi}%
2136     \@namedef{#1@sh@\string#2@}{#4}%
2137   \else
2138     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2139     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2140     {\def\bbl@tempa{#4}%
2141      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2142      \else
2143        \bbl@info
2144          {Redefining #1 shorthand \string#2\string#3\\
2145           in language \CurrentOption}%
2146      \fi}%
2147     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2148   \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.

2149 \def\textormath{%
2150   \ifmmode

```

```

2151 \expandafter\@secondoftwo
2152 \else
2153 \expandafter\@firstoftwo
2154 \fi}

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

2155 \def\user@group{user}
2156 \def\language@group{english} % TODO. I don't like defaults
2157 \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.

2158 \def\usesshorthands{%
2159 \ifstar\bbl@usessh@s{\bbl@usessh@x{}}
2160 \def\bbl@usessh@s#1{%
2161 \bbl@usessh@x
2162 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2163 {#1}}
2164 \def\bbl@usessh@x#1#2{%
2165 \bbl@ifshorthand{#2}%
2166 {\def\user@group{user}%
2167 \initiate@active@char{#2}%
2168 #1%
2169 \bbl@activate{#2}}%
2170 {\bbl@error
2171 {Cannot declare a shorthand turned off (\string#2)}
2172 {Sorry, but you cannot use shorthands which have been\\%
2173 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.

2174 \def\user@language@group{user@\language@group}
2175 \def\bbl@set@user@generic#1#2{%
2176 \bbl@ifunset{user@generic@active#1}%
2177 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2178 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2179 \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2180 \expandafter\noexpand\csname normal@char#1\endcsname}%
2181 \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2182 \expandafter\noexpand\csname user@active#1\endcsname}}%
2183 \@empty}
2184 \newcommand\defineshorthand[3][user]{%
2185 \edef\bbl@tempa{\zap@space#1 \@empty}%
2186 \bbl@for\bbl@tempb\bbl@tempa{%
2187 \if*\expandafter\@car\bbl@tempb\@nil
2188 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2189 \@expandtwoargs
2190 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2191 \fi
2192 \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].

```
2193 \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`.

```
2194 \def\aliasshorthand#1#2{%
2195   \bbl@ifshorthand{#2}%
2196   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2197     \ifx\document\@notprerr
2198       \@notshorthand{#2}%
2199     \else
2200       \initiate@active@char{#2}%
2201       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2202         \csname active@char\string#1\endcsname
2203       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2204         \csname normal@char\string#1\endcsname
2205       \bbl@activate{#2}%
2206     \fi
2207   \fi}%
2208   {\bbl@error
2209     {Cannot declare a shorthand turned off (\string#2)}
2210     {Sorry, but you cannot use shorthands which have been\\%
2211       turned off in the package options}}}
```

`\@notshorthand`

```
2212 \def\@notshorthand#1{%
2213   \bbl@error{%
2214     The character '\string #1' should be made a shorthand character;\\%
2215     add the command \string\usesshorthands\string{#1\string} to
2216     the preamble.\\%
2217     I will ignore your instruction}%
2218   {You may proceed, but expect unexpected results}}
```

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

```
2219 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2220 \DeclareRobustCommand*\shorthandoff{%
2221   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2222 \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.

```
2223 \def\bbl@switch@sh#1#2{%
2224   \ifx#2\@nnil\else
2225     \bbl@ifunset{\bbl@active@\string#2}%
2226     {\bbl@error
2227       {I cannot switch '\string#2' on or off--not a shorthand}%
2228       {This character is not a shorthand. Maybe you made\\%
2229         a typing mistake? I will ignore your instruction}}%
2230     {\ifcase#1%
```

```

2231 \catcode`#212\relax
2232 \or
2233 \catcode`#2\active
2234 \or
2235 \csname bbl@oricat@string#2\endcsname
2236 \csname bbl@oridef@string#2\endcsname
2237 \fi}%
2238 \bbl@afterfi\bbl@switch@sh#1%
2239 \fi}

```

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

```

2240 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2241 \def\bbl@putsh#1{%
2242 \bbl@ifunset{bbl@active@string#1}%
2243 {\bbl@putsh@i#1\empty\@nnil}%
2244 {\csname bbl@active@string#1\endcsname}}
2245 \def\bbl@putsh@i#1#2\@nnil{%
2246 \csname\language@group @sh@string#1@%
2247 \ifx\empty#2\else\string#2\fi\endcsname}
2248 \ifx\bbl@opt@shorthands\@nnil\else
2249 \let\bbl@s@initiate@active@char\initiate@active@char
2250 \def\initiate@active@char#1{%
2251 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2252 \let\bbl@s@switch@sh\bbl@switch@sh
2253 \def\bbl@switch@sh#1#2{%
2254 \ifx#2\@nnil\else
2255 \bbl@afterfi
2256 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2257 \fi}
2258 \let\bbl@s@activate\bbl@activate
2259 \def\bbl@activate#1{%
2260 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2261 \let\bbl@s@deactivate\bbl@deactivate
2262 \def\bbl@deactivate#1{%
2263 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2264 \fi

```

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

```

2265 \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.

```

2266 \def\bbl@prim@s{%
2267 \prime\futurelet\@let@token\bbl@pr@m@s}
2268 \def\bbl@if@primes#1#2{%
2269 \ifx#1\@let@token
2270 \expandafter\@firstoftwo
2271 \else\ifx#2\@let@token
2272 \bbl@afterelse\expandafter\@firstoftwo
2273 \else
2274 \bbl@afterfi\expandafter\@secondoftwo
2275 \fi\fi}
2276 \begingroup
2277 \catcode`\^=7 \catcode`\*=\active \lccode`\*='^
2278 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=' '
2279 \lowercase{%

```

```

2280 \gdef\bbl@pr@ms{%
2281 \bbl@if@primes"%
2282 \pr@@s
2283 {\bbl@if@primes*^{\pr@@t\egroup}}
2284 \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).

```

2285 \initiate@active@char{~}
2286 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2287 \bbl@activate{~}

```

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

```

2288 \expandafter\def\csname OT1dqpos\endcsname{127}
2289 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

2290 \ifx\f@encoding\@undefined
2291 \def\f@encoding{OT1}
2292 \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.

```

2293 \bbl@trace{Language attributes}
2294 \newcommand\languageattribute[2]{%
2295 \def\bbl@tempc{#1}%
2296 \bbl@fixname\bbl@tempc
2297 \bbl@iflanguage\bbl@tempc{%
2298 \bbl@vforeach{#2}{%

```

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

```

2299 \ifx\bbl@known@attribs\@undefined
2300 \in@false
2301 \else
2302 \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,}%
2303 \fi
2304 \ifin@
2305 \bbl@warning{%
2306 You have more than once selected the attribute '##1'\%
2307 for language #1. Reported}%
2308 \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 T_EX-code.

```

2309 \bbl@exp{%

```

```

2310      \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2311      \edef\bbl@tempa{\bbl@tempc-##1}%
2312      \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2313      {\csname\bbl@tempc @attr##1\endcsname}%
2314      {\@attrerr{\bbl@tempc}{##1}}%
2315      \fi}}
2316 \onlypreamble\languageattribute

```

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

```

2317 \newcommand*{\@attrerr}[2]{%
2318   \bbl@error
2319   {The attribute #2 is unknown for language #1.}%
2320   {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}`.

```

2321 \def\bbl@declare@ttribute#1#2#3{%
2322   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2323   \ifin@
2324     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2325   \fi
2326   \bbl@add@list\bbl@attributes{#1-#2}%
2327   \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret T_EX 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.

```

2328 \def\bbl@ifattributeset#1#2#3#4{%
2329   \ifx\bbl@known@attribs\@undefined
2330     \in@false
2331   \else
2332     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2333   \fi
2334   \ifin@
2335     \bbl@afterelse#3%
2336   \else
2337     \bbl@afterfi#4%
2338   \fi}

```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the T_EX-code to be executed when the attribute is known and the T_EX-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.

```

2339 \def\bbl@ifknown@ttrib#1#2{%
2340   \let\bbl@tempa\@secondoftwo
2341   \bbl@loopx\bbl@tempb{#2}{%
2342     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2343     \ifin@
2344       \let\bbl@tempa\@firstoftwo
2345     \else
2346       \fi}%
2347   \bbl@tempa}

```

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

```

2348 \def\bbl@clear@ttribs{%
2349   \ifx\bbl@attributes\undefined\else
2350     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2351       \expandafter\bbl@clear@ttrib\bbl@tempa.
2352     }%
2353   \let\bbl@attributes\undefined
2354   \fi}
2355 \def\bbl@clear@ttrib#1-#2.{%
2356   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2357 \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`

```

2358 \bbl@trace{Macros for saving definitions}
2359 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2360 \newcount\babel@savecnt
2361 \babel@beginsave

```

`\babel@save` The macro `\babel@save{<csname>}` saves the current meaning of the control sequence `<csname>` to `\originalTeX`³¹. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable{<variable>}` saves the value of the variable. `<variable>` can be anything allowed after the `\` the primitive.

```

2362 \def\babel@save#1{%
2363   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2364   \toks@\expandafter{\originalTeX\let#1=}
2365   \bbl@exp{%
2366     \def\\originalTeX{\the\toks@<\babel@\number\babel@savecnt>\relax}}
2367   \advance\babel@savecnt\@ne}
2368 \def\babel@savevariable#1{%
2369   \toks@\expandafter{\originalTeX #1=}
2370   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

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

```

2371 \def\bbl@frenchspacing{%
2372   \ifnum\the\sfcodes\.\.=\@m
2373     \let\bbl@nonfrenchspacing\relax
2374   \else
2375     \frenchspacing

```

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


```

2376 \let\bbl@nonfrenchspacing\nonfrenchspacing
2377 \fi}
2378 \let\bbl@nonfrenchspacing\nonfrenchspacing
2379 \let\bbl@elt\relax
2380 \edef\bbl@fs@chars{%
2381 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2382 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2383 \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.

```

2384 \bbl@trace{Short tags}
2385 \def\babeltags#1{%
2386 \edef\bbl@tempa{\zap@space#1 \@empty}%
2387 \def\bbl@tempb##1=##2\@{%
2388 \edef\bbl@tempc{%
2389 \noexpand\newcommand
2390 \expandafter\noexpand\csname ##1\endcsname{%
2391 \noexpand\protect
2392 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2393 \noexpand\newcommand
2394 \expandafter\noexpand\csname text##1\endcsname{%
2395 \noexpand\foreignlanguage{##2}}}
2396 \bbl@tempc}%
2397 \bbl@for\bbl@tempa\bbl@tempa{%
2398 \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.

```

2399 \bbl@trace{Hyphens}
2400 \@onlypreamble\babelhyphenation
2401 \AtEndOfPackage{%
2402 \newcommand\babelhyphenation[2][\@empty]{%
2403 \ifx\bbl@hyphenation@\relax
2404 \let\bbl@hyphenation@\@empty
2405 \fi
2406 \ifx\bbl@hyphlist\@empty\else
2407 \bbl@warning{%
2408 You must not intermingle \string\selectlanguage\space and\%
2409 \string\babelhyphenation\space or some exceptions will not\%
2410 be taken into account. Reported}%
2411 \fi
2412 \ifx\@empty#1%
2413 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2414 \else
2415 \bbl@vforeach{#1}{%
2416 \def\bbl@tempa{##1}%
2417 \bbl@fixname\bbl@tempa
2418 \bbl@iflanguage\bbl@tempa{%
2419 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2420 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2421 {}%

```

```

2422          {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2423          #2}}}%
2424    \fi}}

```

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

```

2425 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2426 \def\bbl@t@one{T1}
2427 \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`.

```

2428 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2429 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2430 \def\bbl@hyphen{%
2431   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2432 \def\bbl@hyphen@i#1#2{%
2433   \bbl@ifunset{bbl@hy#1#2\@empty}%
2434   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2435   {\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.

```

2436 \def\bbl@usehyphen#1{%
2437   \leavevmode
2438   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2439   \nobreak\hskip\z@skip}
2440 \def\bbl@usehyphen#1{%
2441   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2442 \def\bbl@hyphenchar{%
2443   \ifnum\hyphenchar\font=\m@ne
2444     \babellnullhyphen
2445   \else
2446     \char\hyphenchar\font
2447   \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.

```

2448 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2449 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2450 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2451 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2452 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2453 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2454 \def\bbl@hy@repeat{%
2455   \bbl@usehyphen%
2456   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2457 \def\bbl@hy@@repeat{%
2458   \bbl@usehyphen%

```

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

```

2459 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2460 \def\bbl@hy@empty{\hskip\z@skip}
2461 \def\bbl@hy@@empty{\discretionary{}{}{}}

```

`\bbl@disc` For some languages the macro `\bbl@disc` is used to ease the insertion of discretionary for letters that behave ‘abnormally’ at a breakpoint.

```

2462 \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.

```

2463 \bbl@trace{Multiencoding strings}
2464 \def\bbl@tglobal#1{\global\let#1#1}
2465 \def\bbl@recatcode#1{% TODO. Used only once}
2466 \@tempcnta="7F
2467 \def\bbl@tempa{%
2468   \ifnum\@tempcnta>"FF\else
2469     \catcode\@tempcnta=#1\relax
2470     \advance\@tempcnta\@ne
2471     \expandafter\bbl@tempa
2472   \fi}%
2473 \bbl@tempa}

```

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

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

and starts over (and similarly when lowercasing).

```

2474 \@ifpackagewith{babel}{nocase}%
2475 {\let\bbl@patchuclc\relax}%
2476 {\def\bbl@patchuclc{%
2477   \global\let\bbl@patchuclc\relax
2478   \g@addto@macro\@uclclist{\reserved@b\reserved@b\bbl@uclc}}%
2479   \gdef\bbl@uclc##1{%
2480     \let\bbl@encoded\bbl@encoded@uclc
2481     \bbl@ifunset{\language @bbl@uclc}% and resumes it
2482     {##1}%
2483     {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2484       \csname\language @bbl@uclc\endcsname}%
2485     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2486   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2487   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
2488 <<More package options>> ≡
2489 \DeclareOption{nocase}{}
2490 <</More package options>>

```

The following package options control the behavior of \SetString.

```

2491 <<{*More package options}>> ≡
2492 \let\bbl@opt@strings\@nnil % accept strings=value
2493 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2494 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2495 \def\BabelStringsDefault{generic}
2496 <</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.

```

2497 \@onlypreamble\StartBabelCommands
2498 \def\StartBabelCommands{%
2499   \begingroup
2500   \bbl@recatcode{11}%
2501   <<Macros local to BabelCommands>>
2502   \def\bbl@provstring##1##2{%
2503     \providecommand##1{##2}%
2504     \bbl@toglobal##1}%
2505   \global\let\bbl@scafter\@empty
2506   \let\StartBabelCommands\bbl@startcmds
2507   \ifx\BabelLanguages\relax
2508     \let\BabelLanguages\CurrentOption
2509   \fi
2510   \begingroup
2511   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2512   \StartBabelCommands}
2513 \def\bbl@startcmds{%
2514   \ifx\bbl@screset\@nnil\else
2515     \bbl@usehooks{stopcommands}{}%
2516   \fi
2517   \endgroup
2518   \begingroup
2519   \@ifstar
2520   {\ifx\bbl@opt@strings\@nnil
2521     \let\bbl@opt@strings\BabelStringsDefault
2522   \fi
2523   \bbl@startcmds@i}%
2524   \bbl@startcmds@i}
2525 \def\bbl@startcmds@i#1#2{%
2526   \edef\bbl@L{\zap@space#1 \@empty}%
2527   \edef\bbl@G{\zap@space#2 \@empty}%
2528   \bbl@startcmds@ii}
2529 \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.

```

2530 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2531   \let\SetString\@gobbletwo
2532   \let\bbl@stringdef\@gobbletwo
2533   \let\AfterBabelCommands\@gobble

```

```

2534 \ifx\@empty#1%
2535   \def\bbbl@sc@label{generic}%
2536   \def\bbbl@encstring##1##2{%
2537     \ProvideTextCommandDefault##1{##2}%
2538     \bbbl@toglobal##1%
2539     \expandafter\bbbl@toglobal\csname\string?\string##1\endcsname}%
2540   \let\bbbl@sctest\in@true
2541 \else
2542   \let\bbbl@sc@charset\space % <- zapped below
2543   \let\bbbl@sc@fontenc\space % <- " "
2544   \def\bbbl@tempa##1=##2\@nil{%
2545     \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2546   \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
2547   \def\bbbl@tempa##1 ##2{% space -> comma
2548     ##1%
2549     \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
2550   \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
2551   \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
2552   \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
2553   \def\bbbl@encstring##1##2{%
2554     \bbbl@foreach\bbbl@sc@fontenc{%
2555       \bbbl@ifunset{T#####1}%
2556       {}%
2557       {\ProvideTextCommand##1{#####1}{##2}%
2558         \bbbl@toglobal##1%
2559         \expandafter
2560         \bbbl@toglobal\csname#####1\string##1\endcsname}}}%
2561   \def\bbbl@sctest{%
2562     \bbbl@xin@{\bbbl@opt@strings,}{,\bbbl@sc@label,\bbbl@sc@fontenc,}}%
2563 \fi
2564 \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
2565 \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded
2566   \let\AfterBabelCommands\bbbl@aftercmds
2567   \let\SetString\bbbl@setstring
2568   \let\bbbl@stringdef\bbbl@encstring
2569 \else % ie, strings=value
2570 \bbbl@sctest
2571 \ifin@
2572   \let\AfterBabelCommands\bbbl@aftercmds
2573   \let\SetString\bbbl@setstring
2574   \let\bbbl@stringdef\bbbl@provstring
2575 \fi\fi\fi
2576 \bbbl@scswitch
2577 \ifx\bbbl@G\@empty
2578   \def\SetString##1##2{%
2579     \bbbl@error{Missing group for string \string##1}%
2580     {You must assign strings to some category, typically\\%
2581       captions or extras, but you set none}}%
2582 \fi
2583 \ifx\@empty#1%
2584   \bbbl@usehooks{defaultcommands}{}%
2585 \else
2586   \@expandtwoargs
2587   \bbbl@usehooks{encodedcommands}{\bbbl@sc@charset}\bbbl@sc@fontenc}%
2588 \fi}

```

There are two versions of `\bbbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group` (*language*) is reset, but only once (`\bbbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing.

The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date<language>` 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).

```

2589 \def\bbl@forlang#1#2{%
2590   \bbl@for#1\bbl@L{%
2591     \bbl@xin@{,#1,},{, \BabelLanguages,}%
2592     \ifin@#2\relax\fi}}
2593 \def\bbl@scswitch{%
2594   \bbl@forlang\bbl@tempa{%
2595     \ifx\bbl@G\@empty\else
2596       \ifx\SetString\@gobbletwo\else
2597         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2598         \bbl@xin@{,\bbl@GL,},{,\bbl@screset,}%
2599         \ifin@\else
2600           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2601           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2602         \fi
2603       \fi
2604     \fi}}
2605 \AtEndOfPackage{%
2606   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#{2}}}}%
2607   \let\bbl@scswitch\relax}
2608 \@onlypreamble\EndBabelCommands
2609 \def\EndBabelCommands{%
2610   \bbl@usehooks{stopcommands}{}%
2611   \endgroup
2612   \endgroup
2613   \bbl@scafter}
2614 \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.

```

2615 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2616   \bbl@forlang\bbl@tempa{%
2617     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2618     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2619     {\bbl@exp{%
2620       \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2621     }%
2622     \def\BabelString{#2}%
2623     \bbl@usehooks{stringprocess}{}%
2624     \expandafter\bbl@stringdef
2625     \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include `\bbl@encoded` for string to be expanded in case transformations. It is `\relax` by default, but in `\MakeUppercase` and `\MakeLowercase` its value is a modified expandable `\@changed@cmd`.

```

2626 \ifx\bbl@opt@strings\relax
2627   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2628   \bbl@patchuclc
2629   \let\bbl@encoded\relax

```

```

2630 \def\bbl@encoded@ucl#1{%
2631   \@inmathwarn#1%
2632   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2633     \expandafter\ifx\csname ?\string#1\endcsname\relax
2634       \TextSymbolUnavailable#1%
2635     \else
2636       \csname ?\string#1\endcsname
2637     \fi
2638   \else
2639     \csname\cf@encoding\string#1\endcsname
2640   \fi}
2641 \else
2642   \def\bbl@scset#1#2{\def#1{#2}}
2643 \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.

```

2644 <<(*Macros local to BabelCommands)>> ≡
2645 \def\SetStringLoop##1##2{%
2646   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2647   \count@\z@
2648   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2649     \advance\count@\@ne
2650     \toks@\expandafter{\bbl@tempa}%
2651     \bbl@exp{%
2652       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2653       \count@=\the\count@\relax}}}%
2654 <</Macros local to BabelCommands>>

```

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

```

2655 \def\bbl@aftercmds#1{%
2656   \toks@\expandafter{\bbl@scafter#1}%
2657   \xdef\bbl@scafter{\the\toks@}}

```

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

```

2658 <<(*Macros local to BabelCommands)>> ≡
2659 \newcommand\SetCase[3][]{%
2660   \bbl@patchucl
2661   \bbl@forlang\bbl@tempa{%
2662     \expandafter\bbl@encstring
2663     \csname\bbl@tempa @bbl@ucl\endcsname{\bbl@tempa##1}%
2664     \expandafter\bbl@encstring
2665     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2666     \expandafter\bbl@encstring
2667     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2668 <</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.

```

2669 <<(*Macros local to BabelCommands)>> ≡
2670 \newcommand\SetHyphenMap[1]{%
2671   \bbl@forlang\bbl@tempa{%
2672     \expandafter\bbl@stringdef
2673     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%

```

```
2674 <</Macros local to BabelCommands>>
```

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

```
2675 \newcommand\BabelLower[2]{% one to one.
2676   \ifnum\lccode#1=#2\else
2677     \babel@savevariable{\lccode#1}%
2678     \lccode#1=#2\relax
2679   \fi}
2680 \newcommand\BabelLowerMM[4]{% many-to-many
2681   \@tempcnta=#1\relax
2682   \@tempcntb=#4\relax
2683   \def\bb1@tempa{%
2684     \ifnum\@tempcnta>#2\else
2685       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2686       \advance\@tempcnta#3\relax
2687       \advance\@tempcntb#3\relax
2688       \expandafter\bb1@tempa
2689     \fi}%
2690   \bb1@tempa}
2691 \newcommand\BabelLowerMO[4]{% many-to-one
2692   \@tempcnta=#1\relax
2693   \def\bb1@tempa{%
2694     \ifnum\@tempcnta>#2\else
2695       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2696       \advance\@tempcnta#3
2697       \expandafter\bb1@tempa
2698     \fi}%
2699   \bb1@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2700 <<{*More package options}>> ≡
2701 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
2702 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
2703 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
2704 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@}
2705 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2706 <</More package options>>
```

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

```
2707 \AtEndOfPackage{%
2708   \ifx\bb1@opt@hyphenmap\undefined
2709     \bb1@xin@{,}{\bb1@language@opts}%
2710     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
2711   \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.

```
2712 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2713   \@ifstar\bb1@setcaption@s\bb1@setcaption@x}
2714 \def\bb1@setcaption@x#1#2#3{% language caption-name string
2715   \bb1@trim@def\bb1@tempa{#2}%
2716   \bb1@xin@{.template}{\bb1@tempa}%
2717   \ifin@
2718     \bb1@ini@captions@template{#3}{#1}%
2719   \else
2720     \edef\bb1@tempd{%
2721       \expandafter\expandafter\expandafter
2722       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%

```



```

2723 \bbl@xin@
2724 {\expandafter\string\csname #2name\endcsname}%
2725 {\bbl@tempd}%
2726 \ifin@ % Renew caption
2727 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2728 \ifin@
2729 \bbl@exp{%
2730 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2731 {\\\bbl@scset\<#2name>\<#1#2name>}%
2732 {}}%
2733 \else % Old way converts to new way
2734 \bbl@ifunset{#1#2name}%
2735 {\bbl@exp{%
2736 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2737 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2738 {\def\<#2name>{\<#1#2name>}}%
2739 {}}}%
2740 {}}%
2741 \fi
2742 \else
2743 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2744 \ifin@ % New way
2745 \bbl@exp{%
2746 \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2747 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2748 {\\\bbl@scset\<#2name>\<#1#2name>}%
2749 {}}%
2750 \else % Old way, but defined in the new way
2751 \bbl@exp{%
2752 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2753 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2754 {\def\<#2name>{\<#1#2name>}}%
2755 {}}%
2756 \fi%
2757 \fi
2758 \@namedef{#1#2name}{#3}%
2759 \toks@{\expandafter{\bbl@captionslist}%
2760 \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}}%
2761 \ifin@ \else
2762 \bbl@exp{\\\bbl@add\\bbl@captionslist{\<#2name>}}%
2763 \bbl@tglobal\bbl@captionslist
2764 \fi
2765 \fi}
2766 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

9.11 Macros common to a number of languages

`\set@low@box` The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```

2767 \bbl@trace{Macros related to glyphs}
2768 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2769 \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2770 \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.

```

2771 \def\save@sf@q#1{\leavevmode
2772 \begingroup
2773 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2774 \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.

```
2775 \ProvideTextCommand{\quotedblbase}{OT1}{%
2776   \save@sf@q{\set@low@box{\textquotedblright\}%
2777     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
2778 \ProvideTextCommandDefault{\quotedblbase}{%
2779   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2780 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2781   \save@sf@q{\set@low@box{\textquoteright\}%
2782     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
2783 \ProvideTextCommandDefault{\quotesinglbase}{%
2784   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2785 \ProvideTextCommand{\guillemetleft}{OT1}{%
2786   \ifmmode
2787     \ll
2788   \else
2789     \save@sf@q{\nobreak
2790       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2791   \fi}
2792 \ProvideTextCommand{\guillemetright}{OT1}{%
2793   \ifmmode
2794     \gg
2795   \else
2796     \save@sf@q{\nobreak
2797       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2798   \fi}
2799 \ProvideTextCommand{\guillemotleft}{OT1}{%
2800   \ifmmode
2801     \ll
2802   \else
2803     \save@sf@q{\nobreak
2804       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2805   \fi}
2806 \ProvideTextCommand{\guillemotright}{OT1}{%
2807   \ifmmode
2808     \gg
2809   \else
2810     \save@sf@q{\nobreak
2811       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2812   \fi}
```

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

```
2813 \ProvideTextCommandDefault{\guillemetleft}{%
2814   \UseTextSymbol{OT1}{\guillemetleft}}
2815 \ProvideTextCommandDefault{\guillemetright}{%
2816   \UseTextSymbol{OT1}{\guillemetright}}
2817 \ProvideTextCommandDefault{\guillemotleft}{%
2818   \UseTextSymbol{OT1}{\guillemotleft}}
2819 \ProvideTextCommandDefault{\guillemotright}{%
2820   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2821 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2822   \ifmmode
2823     <%
2824   \else
2825     \save@sf@q{\nobreak
2826       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2827   \fi}
2828 \ProvideTextCommand{\guilsinglright}{OT1}{%
2829   \ifmmode
2830     >%
2831   \else
2832     \save@sf@q{\nobreak
2833       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2834   \fi}
```

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

```
2835 \ProvideTextCommandDefault{\guilsinglleft}{%
2836   \UseTextSymbol{OT1}{\guilsinglleft}}
2837 \ProvideTextCommandDefault{\guilsinglright}{%
2838   \UseTextSymbol{OT1}{\guilsinglright}}
```

9.12.2 Letters

`\ij` The dutch language uses the letter ‘ij’. It is available in T1 encoded fonts, but not in the OT1 encoded
`\IJ` fonts. Therefore we fake it for the OT1 encoding.

```
2839 \DeclareTextCommand{\ij}{OT1}{%
2840   i\kern-0.02em\bbl@allowhyphens j}
2841 \DeclareTextCommand{\IJ}{OT1}{%
2842   I\kern-0.02em\bbl@allowhyphens J}
2843 \DeclareTextCommand{\ij}{T1}{\char188}
2844 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2845 \ProvideTextCommandDefault{\ij}{%
2846   \UseTextSymbol{OT1}{\ij}}
2847 \ProvideTextCommandDefault{\IJ}{%
2848   \UseTextSymbol{OT1}{\IJ}}
```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, but not in
`\DJ` the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```
2849 \def\crrtic@{\hrule height0.1ex width0.3em}
2850 \def\crttic@{\hrule height0.1ex width0.33em}
2851 \def\ddj@{%
2852   \setbox0\hbox{d}\dimen@=\ht0
2853   \advance\dimen@1ex
```

```

2854 \dimen@.45\dimen@
2855 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2856 \advance\dimen@ii.5ex
2857 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2858 \def\DDJ@{%
2859 \setbox0\hbox{D}\dimen@=.55\ht0
2860 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2861 \advance\dimen@ii.15ex % correction for the dash position
2862 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2863 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2864 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2865 %
2866 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2867 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2868 \ProvideTextCommandDefault{\dj}{%
2869 \UseTextSymbol{OT1}{\dj}}
2870 \ProvideTextCommandDefault{\DJ}{%
2871 \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.

```

2872 \DeclareTextCommand{\SS}{OT1}{\SS}
2873 \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

```

2874 \ProvideTextCommandDefault{\glq}{%
2875 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2876 \ProvideTextCommand{\grq}{T1}{%
2877 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2878 \ProvideTextCommand{\grq}{TU}{%
2879 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2880 \ProvideTextCommand{\grq}{OT1}{%
2881 \save@sf@q{\kern-.0125em
2882 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2883 \kern.07em\relax}}
2884 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

\glqq The ‘german’ double quotes.

\grqq

```

2885 \ProvideTextCommandDefault{\glqq}{%
2886 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2887 \ProvideTextCommand{\grqq}{T1}{%
2888 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2889 \ProvideTextCommand{\grqq}{TU}{%
2890 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2891 \ProvideTextCommand{\grqq}{OT1}{%
2892 \save@sf@q{\kern-.07em
2893 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%

```


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

```

2927 \AtBeginDocument{%
2928   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2929   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2930   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2931   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
2932   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2933   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2934   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2935   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2936   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2937   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2938   \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}%

```

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

```

2939 \ifx\l@english\@undefined
2940   \chardef\l@english\z@
2941 \fi
2942 % The following is used to cancel rules in ini files (see Amharic).
2943 \ifx\l@babelnohyphens\@undefined
2944   \newlanguage\l@babelnohyphens
2945 \fi

```

9.13 Layout

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

```

2946 \bbl@trace{Bidi layout}
2947 \providecommand\IfBabelLayout[3]{#3}%
2948 \newcommand\BabelPatchSection[1]{%
2949   \@ifundefined{#1}{%
2950     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2951     \@namedef{#1}{%
2952       \@ifstar{\bbl@presec@s{#1}}%
2953       {\@dblarg{\bbl@presec@x{#1}}}}%
2954   \def\bbl@presec@x#1[#2]#3{%
2955     \bbl@exp{%
2956       \\select@language@x{\bbl@main@language}%
2957       \\bbl@cs{sspre@#1}%
2958       \\bbl@cs{ss@#1}%
2959       [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2960       {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2961       \\select@language@x{\language}}%
2962   \def\bbl@presec@s#1#2{%
2963     \bbl@exp{%
2964       \\select@language@x{\bbl@main@language}%
2965       \\bbl@cs{sspre@#1}%
2966       \\bbl@cs{ss@#1}*%
2967       {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2968       \\select@language@x{\language}}%
2969   \IfBabelLayout{sectioning}%
2970   {\BabelPatchSection{part}}%
2971   \BabelPatchSection{chapter}%

```

```

2972 \BabelPatchSection{section}%
2973 \BabelPatchSection{subsection}%
2974 \BabelPatchSection{subsubsection}%
2975 \BabelPatchSection{paragraph}%
2976 \BabelPatchSection{subparagraph}%
2977 \def\babel@toc#1{%
2978     \select@language@x{\bbl@main@language}}{}
2979 \IfBabelLayout{captions}%
2980 {\BabelPatchSection{caption}}{}

```

9.14 Load engine specific macros

```

2981 \bbl@trace{Input engine specific macros}
2982 \ifcase\bbl@engine
2983 \input txtbabel.def
2984 \or
2985 \input luababel.def
2986 \or
2987 \input xebabel.def
2988 \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.

```

2989 \bbl@trace{Creating languages and reading ini files}
2990 \newcommand\babelprovide[2][]{%
2991     \let\bbl@savelangname\languagename
2992     \edef\bbl@savelocaleid{\the\localeid}%
2993     % Set name and locale id
2994     \edef\languagename{#2}%
2995     % \global\@namedef{\bbl@lcname@#2}{#2}%
2996     \bbl@id@assign
2997     \let\bbl@KVP@captions\@nil
2998     \let\bbl@KVP@date\@nil
2999     \let\bbl@KVP@import\@nil
3000     \let\bbl@KVP@main\@nil
3001     \let\bbl@KVP@script\@nil
3002     \let\bbl@KVP@language\@nil
3003     \let\bbl@KVP@hyphenrules\@nil
3004     \let\bbl@KVP@mapfont\@nil
3005     \let\bbl@KVP@maparabic\@nil
3006     \let\bbl@KVP@mapdigits\@nil
3007     \let\bbl@KVP@intraspace\@nil
3008     \let\bbl@KVP@intrapenalty\@nil
3009     \let\bbl@KVP@onchar\@nil
3010     \let\bbl@KVP@alph\@nil
3011     \let\bbl@KVP@Alph\@nil
3012     \let\bbl@KVP@labels\@nil
3013     \bbl@csarg\let{KVP@labels*}\@nil
3014     \global\let\bbl@inidata\@empty
3015     \bbl@forkv{#1}{% TODO - error handling
3016         \in@{/}{##1}%
3017         \ifin@
3018             \bbl@renewinikey##1\@{##2}%
3019         \else
3020             \bbl@csarg\def{KVP@##1}{##2}%
3021         \fi}%

```

```

3022 % == init ==
3023 \ifx\bb1@screset\@undefined
3024 \bb1@ldfinit
3025 \fi
3026 % ==
3027 \let\bb1@lbkflag\relax % \@empty = do setup linebreak
3028 \bb1@ifunset{date#2}%
3029   {\let\bb1@lbkflag\@empty}% new
3030   {\ifx\bb1@KVP@hyphenrules\@nil\else
3031     \let\bb1@lbkflag\@empty
3032   \fi
3033   \ifx\bb1@KVP@import\@nil\else
3034     \let\bb1@lbkflag\@empty
3035   \fi}%
3036 % == import, captions ==
3037 \ifx\bb1@KVP@import\@nil\else
3038   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
3039   {\ifx\bb1@initoload\relax
3040     \begingroup
3041       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3042       \bb1@input@texini{#2}%
3043     \endgroup
3044   \else
3045     \xdef\bb1@KVP@import{\bb1@initoload}%
3046   \fi}%
3047 {}%
3048 \fi
3049 \ifx\bb1@KVP@captions\@nil
3050   \let\bb1@KVP@captions\bb1@KVP@import
3051 \fi
3052 % Load ini
3053 \bb1@ifunset{date#2}%
3054   {\bb1@provide@new{#2}}%
3055   {\bb1@ifblank{#1}%
3056     {}% With \bb1@load@basic below
3057     {\bb1@provide@renew{#2}}}%
3058 % Post tasks
3059 % -----
3060 % == ensure captions ==
3061 \ifx\bb1@KVP@captions\@nil\else
3062   \bb1@ifunset{\bb1@extracaps@#2}%
3063     {\bb1@exp{\bb1@babelensure[exclude=\\today]{#2}}}%
3064     {\toks@ \expandafter \expandafter \expandafter
3065      {\csname \bb1@extracaps@#2\endcsname}%
3066      \bb1@exp{\bb1@babelensure[exclude=\\today,include=\the\toks@]{#2}}%
3067   \bb1@ifunset{\bb1@ensure@\language}%
3068     {\bb1@exp{%
3069       \\DeclareRobustCommand\<\bb1@ensure@\language>[1]{%
3070         \\foreignlanguage{\language}%
3071         {###1}}}%
3072     {}%
3073   \bb1@exp{%
3074     \\bb1@tglobal\<\bb1@ensure@\language>%
3075     \\bb1@tglobal\<\bb1@ensure@\language\space>}%
3076 \fi
3077 % ==
3078 % At this point all parameters are defined if 'import'. Now we
3079 % execute some code depending on them. But what about if nothing was
3080 % imported? We just set the basic parameters, but still loading the

```



```

3081 % whole ini file.
3082 \bbl@load@basic{#2}%
3083 % == script, language ==
3084 % Override the values from ini or defines them
3085 \ifx\bbl@KVP@script\@nil\else
3086   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
3087 \fi
3088 \ifx\bbl@KVP@language\@nil\else
3089   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
3090 \fi
3091 % == onchar ==
3092 \ifx\bbl@KVP@onchar\@nil\else
3093   \bbl@luahyphenate
3094   \directlua{
3095     if Babel.locale_mapped == nil then
3096       Babel.locale_mapped = true
3097       Babel.linebreaking.add_before(Babel.locale_map)
3098       Babel.loc_to_scr = {}
3099       Babel.chr_to_loc = Babel.chr_to_loc or {}
3100     end}%
3101   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3102   \ifin@
3103     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
3104       \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
3105     \fi
3106     \bbl@exp{\bbl@add\bbl@starthyphens
3107       {\bbl@patterns@lua{languagename}}}%
3108     % TODO - error/warning if no script
3109     \directlua{
3110       if Babel.script_blocks['\bbl@cl{sbc}'] then
3111         Babel.loc_to_scr[\the\localeid] =
3112           Babel.script_blocks['\bbl@cl{sbc}']
3113         Babel.locale_props[\the\localeid].lc = \the\localeid\space
3114         Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@languagename}\space
3115       end
3116     }%
3117   \fi
3118   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3119   \ifin@
3120     \bbl@ifunset{bbl@lsys@languagename}{\bbl@provide@lsys{languagename}}{}%
3121     \bbl@ifunset{bbl@wdir@languagename}{\bbl@provide@dirs{languagename}}{}%
3122     \directlua{
3123       if Babel.script_blocks['\bbl@cl{sbc}'] then
3124         Babel.loc_to_scr[\the\localeid] =
3125           Babel.script_blocks['\bbl@cl{sbc}']
3126       end}%
3127     \ifx\bbl@mapselect\@undefined
3128       \AtBeginDocument{%
3129         \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3130         {\selectfont}}%
3131       \def\bbl@mapselect{%
3132         \let\bbl@mapselect\relax
3133         \edef\bbl@prefontid{\fontid\font}}%
3134       \def\bbl@mapdir##1{%
3135         {\def\languagename{##1}%
3136         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3137         \bbl@switchfont
3138         \directlua{
3139           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%

```

```

3140         ['\bbl@prefontid'] = \fontid\font\space}}}%
3141     \fi
3142     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3143     \fi
3144     % TODO - catch non-valid values
3145 \fi
3146 % == mapfont ==
3147 % For bidi texts, to switch the font based on direction
3148 \ifx\bbl@KVP@mapfont\@nil\else
3149     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
3150     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
3151         mapfont. Use 'direction'.%
3152         {See the manual for details.}}}%
3153     \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys@\language}}}%
3154     \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}}%
3155     \ifx\bbl@mapselect\undefined
3156         \AtBeginDocument{%
3157             \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3158             {\selectfont}}%
3159         \def\bbl@mapselect{%
3160             \let\bbl@mapselect\relax
3161             \edef\bbl@prefontid{\fontid\font}}%
3162         \def\bbl@mapdir##1{%
3163             {\def\language{##1}%
3164             \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3165             \bbl@switchfont
3166             \directlua{Babel.fontmap
3167             [\the\csname bbl@wdir@##1\endcsname]%
3168             [\bbl@prefontid]=\fontid\font}}}%
3169     \fi
3170     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3171 \fi
3172 % == Line breaking: intraspace, intrapenalty ==
3173 % For CJK, East Asian, Southeast Asian, if interspace in ini
3174 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3175     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3176 \fi
3177 \bbl@provide@intraspace
3178 % == Line breaking: hyphenate.other.locale/.script==
3179 \ifx\bbl@lbkflag\@empty
3180     \bbl@ifunset{\bbl@hyotl@\language}}}%
3181     {\bbl@csarg\bbl@replace{hyotl@\language}{ }{,}}%
3182     \bbl@startcommands*\language}}}%
3183     \bbl@csarg\bbl@foreach{hyotl@\language}{%
3184         \ifcase\bbl@engine
3185             \ifnum##1<257
3186                 \SetHyphenMap{\BabelLower{##1}{##1}}%
3187             \fi
3188         \else
3189             \SetHyphenMap{\BabelLower{##1}{##1}}%
3190         \fi}%
3191     \bbl@endcommands}%
3192 \bbl@ifunset{\bbl@hyots@\language}}}%
3193 {\bbl@csarg\bbl@replace{hyots@\language}{ }{,}}%
3194 \bbl@csarg\bbl@foreach{hyots@\language}{%
3195     \ifcase\bbl@engine
3196         \ifnum##1<257
3197             \global\lccode##1=##1\relax
3198         \fi

```

```

3199         \else
3200             \global\lccode##1=##1\relax
3201         \fi}}%
3202 \fi
3203 % == Counters: maparabic ==
3204 % Native digits, if provided in ini (TeX level, xe and lua)
3205 \ifcase\bbbl@engine\else
3206     \bbbl@ifunset{\bbbl@dgnat@\languagename}{}%
3207     {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
3208         \expandafter\expandafter\expandafter
3209         \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
3210         \ifx\bbbl@KVP@maparabic\@nil\else
3211             \ifx\bbbl@latinarabic\@undefined
3212                 \expandafter\let\expandafter\@arabic
3213                 \csname bbl@counter@\languagename\endcsname
3214             \else % ie, if layout=counters, which redefines \@arabic
3215                 \expandafter\let\expandafter\bbbl@latinarabic
3216                 \csname bbl@counter@\languagename\endcsname
3217             \fi
3218         \fi
3219     \fi}%
3220 \fi
3221 % == Counters: mapdigits ==
3222 % Native digits (lua level).
3223 \ifodd\bbbl@engine
3224     \ifx\bbbl@KVP@mapdigits\@nil\else
3225         \bbbl@ifunset{\bbbl@dgnat@\languagename}{}%
3226         {\RequirePackage{luatexbase}%
3227         \bbl@activate@preotf
3228         \directlua{
3229             Babel = Babel or {} %% -> presets in luababel
3230             Babel.digits_mapped = true
3231             Babel.digits = Babel.digits or {}
3232             Babel.digits[\the\localeid] =
3233             table.pack(string.utfvalue('\bbbl@cl{dgnat}'))
3234             if not Babel.numbers then
3235                 function Babel.numbers(head)
3236                     local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3237                     local GLYPH = node.id'glyph'
3238                     local inmath = false
3239                     for item in node.traverse(head) do
3240                         if not inmath and item.id == GLYPH then
3241                             local temp = node.get_attribute(item, LOCALE)
3242                             if Babel.digits[temp] then
3243                                 local chr = item.char
3244                                 if chr > 47 and chr < 58 then
3245                                     item.char = Babel.digits[temp][chr-47]
3246                                 end
3247                             end
3248                         elseif item.id == node.id'math' then
3249                             inmath = (item.subtype == 0)
3250                         end
3251                     end
3252                     return head
3253                 end
3254             end
3255         }}%
3256     \fi
3257 \fi

```

```

3258 % == Counters: alph, Alph ==
3259 % What if extras<lang> contains a \babel@save\@alph? It won't be
3260 % restored correctly when exiting the language, so we ignore
3261 % this change with the \bbl@alph@saved trick.
3262 \ifx\bbl@KVP@alph\@nil\else
3263   \toks@\expandafter\expandafter\expandafter{%
3264     \csname extras\language\endcsname}%
3265   \bbl@exp{%
3266     \def\<extras\language>{%
3267       \let\\bbl@alph@saved\\@alph
3268       \the\toks@
3269       \let\\@alph\\bbl@alph@saved
3270       \\babel@save\\@alph
3271       \let\\@alph<bbl@cntr@bbl@KVP@alph @\language>}}%
3272 \fi
3273 \ifx\bbl@KVP@Alph\@nil\else
3274   \toks@\expandafter\expandafter\expandafter{%
3275     \csname extras\language\endcsname}%
3276   \bbl@exp{%
3277     \def\<extras\language>{%
3278       \let\\bbl@Alph@saved\\@Alph
3279       \the\toks@
3280       \let\\@Alph\\bbl@Alph@saved
3281       \\babel@save\\@Alph
3282       \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language>}}%
3283 \fi
3284 % == require.babel in ini ==
3285 % To load or reload the babel-*.tex, if require.babel in ini
3286 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3287   \bbl@ifunset{bbl@rqtex@\language}{}%
3288   {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3289     \let\BabelBeforeIni@gobbletwo
3290     \chardef\atcatcode=\catcode`\@
3291     \catcode`\@=11\relax
3292     \bbl@input@texini{\bbl@cs{rqtex@\language}}%
3293     \catcode`\@=\atcatcode
3294     \let\atcatcode\relax
3295   \fi}%
3296 \fi
3297 % == main ==
3298 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3299   \let\language\bbl@savelangname
3300   \chardef\localeid\bbl@savelocaleid\relax
3301 \fi}

```

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

```

3302 \def\bbl@provide@new#1{%
3303   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3304   \@namedef{extras#1}{}%
3305   \@namedef{noextras#1}{}%
3306   \bbl@startcommands*{#1}{captions}%
3307   \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3308     \def\bbl@tempb##1% elt for \bbl@captionslist
3309     \ifx##1\@empty\else
3310       \bbl@exp{%
3311         \\SetString\\##1%
3312         \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}%
3313       \expandafter\bbl@tempb
3314     \fi}%

```

```

3315 \expandafter\bb1@tempb\bb1@captionslist\@empty
3316 \else
3317 \ifx\bb1@initoload\relax
3318 \bb1@read@ini{\bb1@KVP@captions}2% % Here letters cat = 11
3319 \else
3320 \bb1@read@ini{\bb1@initoload}2% % Same
3321 \fi
3322 \fi
3323 \StartBabelCommands*{#1}{date}%
3324 \ifx\bb1@KVP@import\@nil
3325 \bb1@exp{%
3326 \SetString\today{\bb1@nocaption{today}{#1today}}}%
3327 \else
3328 \bb1@savetoday
3329 \bb1@savetoday
3330 \fi
3331 \bb1@endcommands
3332 \bb1@load@basic{#1}%
3333 % == hyphenmins == (only if new)
3334 \bb1@exp{%
3335 \gdef\<#1hyphenmins>{%
3336 {\bb1@ifunset{\bb1@lftm@#1}{2}{\bb1@cs{lftm@#1}}}%
3337 {\bb1@ifunset{\bb1@rgtm@#1}{3}{\bb1@cs{rgtm@#1}}}%
3338 % == hyphenrules ==
3339 \bb1@provide@hyphens{#1}%
3340 % == frenchspacing == (only if new)
3341 \bb1@ifunset{\bb1@frspc@#1}{}%
3342 {\edef\bb1@tempa{\bb1@cl{frspc}}%
3343 \edef\bb1@tempa{\expandafter\@car\bb1@tempa\@nil}%
3344 \if u\bb1@tempa % do nothing
3345 \else\if n\bb1@tempa % non french
3346 \expandafter\bb1@add\csname extras#1\endcsname{%
3347 \let\bb1@elt\bb1@fs@elt@i
3348 \bb1@fs@chars}%
3349 \else\if y\bb1@tempa % french
3350 \expandafter\bb1@add\csname extras#1\endcsname{%
3351 \let\bb1@elt\bb1@fs@elt@ii
3352 \bb1@fs@chars}%
3353 \fi\fi\fi}%
3354 %
3355 \ifx\bb1@KVP@main\@nil\else
3356 \expandafter\main@language\expandafter{#1}%
3357 \fi}
3358 % A couple of macros used above, to avoid hashes #####...
3359 \def\bb1@fs@elt@i#1#2#3{%
3360 \ifnum\sfcode`#1=#2\relax
3361 \babel@savevariable{\sfcode`#1}%
3362 \sfcode`#1=#3\relax
3363 \fi}%
3364 \def\bb1@fs@elt@ii#1#2#3{%
3365 \ifnum\sfcode`#1=#3\relax
3366 \babel@savevariable{\sfcode`#1}%
3367 \sfcode`#1=#2\relax
3368 \fi}%
3369 %
3370 \def\bb1@provide@renew#1{%
3371 \ifx\bb1@KVP@captions\@nil\else
3372 \StartBabelCommands*{#1}{captions}%
3373 \bb1@read@ini{\bb1@KVP@captions}2% % Here all letters cat = 11

```

```

3374 \EndBabelCommands
3375 \fi
3376 \ifx\bbbl@KVP@import\@nil\else
3377 \StartBabelCommands*{#1}{date}%
3378 \bbbl@savetoday
3379 \bbbl@savestate
3380 \EndBabelCommands
3381 \fi
3382 % == hyphenrules ==
3383 \ifx\bbbl@lbfkflag\@empty
3384 \bbbl@provide@hyphens{#1}%
3385 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values.

```

3386 \def\bbbl@load@basic#1{%
3387 \bbbl@ifunset{bbbl@inidata@\language\language}{}%
3388 {\getlocaleproperty\bbbl@tempa{\language\language}{identification/load.level}%
3389 \ifcase\bbbl@tempa
3390 \bbbl@csarg\let{lname@\language\language}\relax
3391 \fi}%
3392 \bbbl@ifunset{bbbl@lname@#1}%
3393 {\def\BabelBeforeIni##1##2{%
3394 \begingroup
3395 \let\bbbl@ini@captions@aux\@gobbletwo
3396 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3397 \bbbl@read@ini{##1}1%
3398 \ifx\bbbl@initoload\relax\endinput\fi
3399 \endgroup}%
3400 \begingroup % boxed, to avoid extra spaces:
3401 \ifx\bbbl@initoload\relax
3402 \bbbl@input@texini{##1}%
3403 \else
3404 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
3405 \fi
3406 \endgroup}%
3407 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3408 \def\bbbl@provide@hyphens#1{%
3409 \let\bbbl@tempa\relax
3410 \ifx\bbbl@KVP@hyphenrules\@nil\else
3411 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3412 \bbbl@foreach\bbbl@KVP@hyphenrules{%
3413 \ifx\bbbl@tempa\relax % if not yet found
3414 \bbbl@ifsamestring{##1}{+}%
3415 {\bbbl@exp{\addlanguage\<l@##1>}}}%
3416 {}%
3417 \bbbl@ifunset{l@##1}%
3418 {}%
3419 {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}}%
3420 \fi}%
3421 \fi
3422 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
3423 \ifx\bbbl@KVP@import\@nil
3424 \ifx\bbbl@initoload\relax\else
3425 \bbbl@exp{ % and hyphenrules is not empty
3426 \bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%

```

```

3427         {}%
3428         {\let\bbbl@tempa\<l@bbbl@c1{hyphr}>}}%
3429     \fi
3430     \else % if importing
3431         \bbbl@exp{%                and hyphenrules is not empty
3432             \bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3433             {}%
3434             {\let\bbbl@tempa\<l@bbbl@c1{hyphr}>}}%
3435     \fi
3436 \fi
3437 \bbbl@ifunset{bbbl@tempa}%        ie, relax or undefined
3438 {\bbbl@ifunset{l@#1}%            no hyphenrules found - fallback
3439     {\bbbl@exp{\adddialect\<l@#1>\language}}%
3440     {}}%                          so, l@<lang> is ok - nothing to do
3441 {\bbbl@exp{\adddialect\<l@#1>\bbbl@tempa}}% found in opt list or ini

```

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

```

3442 \def\bbbl@input@texini#1{%
3443     \bbbl@bsphack
3444     \bbbl@exp{%
3445         \catcode\=14 \catcode\=0
3446         \catcode\={1 \catcode\}=2
3447         \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
3448         \catcode\=the\catcode\}\relax
3449         \catcode\=the\catcode\}\relax
3450         \catcode\={the\catcode\}\relax
3451         \catcode\}=the\catcode\}\relax}%
3452     \bbbl@esphack}

```

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

```

3453 \def\bbbl@inline#1\bbbl@inline{%
3454     \@ifnextchar[\bbbl@inisect{\@ifnextchar\bbbl@iniskip\bbbl@inistore}#1\@@% ]
3455 \def\bbbl@inisect[#1]#2\@@{\def\bbbl@section{#1}}%
3456 \def\bbbl@iniskip#1\@@{%          if starts with ;
3457 \def\bbbl@inistore#1=#2\@@{%      full (default)
3458     \bbbl@trim@def\bbbl@tempa{#1}%
3459     \bbbl@trim\toks@{#2}%
3460     \bbbl@ifunset{bbbl@KVP@\bbbl@section/\bbbl@tempa}%
3461     {\bbbl@exp{%
3462         \g@addto@macro\bbbl@inidata{%
3463             \bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3464     }}%
3465 \def\bbbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbbl@read@ini)
3466     \bbbl@trim@def\bbbl@tempa{#1}%
3467     \bbbl@trim\toks@{#2}%
3468     \bbbl@xin@{.identification.}{.\bbbl@section.}%
3469     \ifin@
3470         \bbbl@exp{\g@addto@macro\bbbl@inidata{%
3471             \bbbl@elt{identification}{\bbbl@tempa}{\the\toks@}}}%
3472     \fi}%

```

Now, the 'main loop', which **must be executed inside a group**. At this point, \bbbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

```

3473 \ifx\bbl@readstream\@undefined
3474 \csname newread\endcsname\bbl@readstream
3475 \fi
3476 \def\bbl@read@ini#1#2{%
3477 \openin\bbl@readstream=babel-#1.ini
3478 \ifeof\bbl@readstream
3479 \bbl@error
3480 {There is no ini file for the requested language\%
3481 (#1). Perhaps you misspelled it or your installation\%
3482 is not complete.}%
3483 {Fix the name or reinstall babel.}%
3484 \else
3485 % Store ini data in \bbl@inidata
3486 \catcode\ [=12 \catcode\ ]=12 \catcode\ ==12 \catcode\ &=12
3487 \catcode\ ;=12 \catcode\ |=12 \catcode\ %=14 \catcode\ -=12
3488 \bbl@info{Importing
3489 \ifcase#2font and identification \or basic \fi
3490 data for \language\%
3491 from babel-#1.ini. Reported}%
3492 \ifnum#2=\z@
3493 \global\let\bbl@inidata\@empty
3494 \let\bbl@inistore\bbl@inistore@min % Remember it's local
3495 \fi
3496 \def\bbl@section{identification}%
3497 \bbl@exp{\ \bbl@inistore tag.ini=#1\ \ \ \}%
3498 \bbl@inistore load.level=#2\ \ \
3499 \loop
3500 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3501 \endlinechar\m@ne
3502 \read\bbl@readstream to \bbl@line
3503 \endlinechar\^^M
3504 \ifx\bbl@line\@empty\else
3505 \expandafter\bbl@inline\bbl@line\bbl@inline
3506 \fi
3507 \repeat
3508 % Process stored data
3509 \bbl@csarg\xdef\lini@language{#1}%
3510 \let\bbl@savestrings\@empty
3511 \let\bbl@savetoday\@empty
3512 \let\bbl@savestate\@empty
3513 \def\bbl@elt##1##2##3{%
3514 \def\bbl@section{##1}%
3515 \in@{=date.}{=##1}% Find a better place
3516 \ifin@
3517 \bbl@ini@calendar{##1}%
3518 \fi
3519 \global\bbl@csarg\let\bbl@KVP@##1/##2\relax
3520 \bbl@ifunset\bbl@inikv@##1-{}%
3521 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
3522 \bbl@inidata
3523 % 'Export' data
3524 \bbl@ini@exports{#2}%
3525 \global\bbl@csarg\let\bbl@inidata@language\bbl@inidata
3526 \global\let\bbl@inidata\@empty
3527 \bbl@exp{\ \bbl@add@list\ \bbl@ini@loaded{language}}%
3528 \bbl@tglobal\bbl@ini@loaded
3529 \fi}

```

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


```

3530 \def\bbl@ini@calendar#1{%
3531   \lowercase{\def\bbl@tempa{=#1=}}%
3532   \bbl@replace\bbl@tempa{=date.gregorian}{}%
3533   \bbl@replace\bbl@tempa{=date.}{}%
3534   \in@{.licr=}{#1=}%
3535   \ifin@
3536     \ifcase\bbl@engine
3537       \bbl@replace\bbl@tempa{.licr=}{}%
3538     \else
3539       \let\bbl@tempa\relax
3540     \fi
3541 \fi
3542 \ifx\bbl@tempa\relax\else
3543   \bbl@replace\bbl@tempa{=}{}%
3544   \bbl@exp{%
3545     \def\<bbl@inikv@#1>####1####2{%
3546       \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}%
3547 \fi}

```

A key with a slash in `\babelprovide` replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in `\bbl@inistore` above).

```

3548 \def\bbl@renewinikey#1/#2\@#3{%
3549   \edef\bbl@tempa{\zap@space #1 \@empty}% section
3550   \edef\bbl@tempb{\zap@space #2 \@empty}% key
3551   \bbl@trim\toks@{#3}% value
3552   \bbl@exp{%
3553     \global\let\<bbl@KVP@\bbl@tempa/\bbl@tempb>\\@empty % just a flag
3554     \\\g@addto@macro\\bbl@inidata{%
3555       \\\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

3556 \def\bbl@exportkey#1#2#3{%
3557   \bbl@ifunset{bbl@kv@#2}%
3558   {\bbl@csarg\gdef{#1@\language}\@empty}%
3559   {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3560     \bbl@csarg\gdef{#1@\language}\@empty}%
3561   \else
3562     \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3563   \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

```

3564 \def\bbl@iniwarning#1{%
3565   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3566   {\bbl@warning{%
3567     From babel-\bbl@cs{lini@\language}.ini:\%
3568     \bbl@cs{kv@identification.warning#1}\%
3569     Reported }}%
3570 %
3571 \def\bbl@ini@exports#1{%
3572   % Identification always exported
3573   \bbl@iniwarning}%
3574   \ifcase\bbl@engine
3575     \bbl@iniwarning{.pdflatex}%
3576   \or

```

```

3577 \bbl@iniwarning{.lualatex}%
3578 \or
3579 \bbl@iniwarning{.xelatex}%
3580 \fi%
3581 \bbl@exportkey{elname}{identification.name.english}{}%
3582 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3583 {\csname bbl@elname@language\endcsname}}%
3584 \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
3585 \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
3586 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3587 \bbl@exportkey{esname}{identification.script.name}{}%
3588 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3589 {\csname bbl@esname@language\endcsname}}%
3590 \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3591 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3592 % Also maps bcp47 -> language
3593 \ifbbl@bcptoname
3594 \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{language}%
3595 \fi
3596 % Conditional
3597 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
3598 \bbl@exportkey{lncr}{typography.linebreaking}{h}%
3599 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3600 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3601 \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3602 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3603 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3604 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3605 \bbl@exportkey{intsp}{typography.intraspace}{}%
3606 \bbl@exportkey{chrng}{characters.ranges}{}%
3607 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3608 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3609 \ifnum#1=\tw@ % only (re)new
3610 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3611 \bbl@tglobal\bbl@savetoday
3612 \bbl@tglobal\bbl@savestate
3613 \bbl@savestrings
3614 \fi
3615 \fi}

```

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

```

3616 \def\bbl@inikv#1#2{% key=value
3617 \toks@{#2}% This hides #'s from ini values
3618 \bbl@csarg\xdef{kv@bbl@section.#1}{\the\toks@}}

```

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

```

3619 \let\bbl@inikv@identification\bbl@inikv
3620 \let\bbl@inikv@typography\bbl@inikv
3621 \let\bbl@inikv@characters\bbl@inikv
3622 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3623 \def\bbl@inikv@counters#1#2{%
3624 \bbl@ifsamestring{#1}{digits}%
3625 {\bbl@error{The counter name 'digits' is reserved for mapping\\
3626 decimal digits}%
3627 {Use another name.}}%

```



```

3681 \bbl@exp{%
3682   \\bbl@add\\bbl@savestrings{%
3683     \\SetString\<\bbl@tempa name>{\the\toks@}}}%
3684 \toks@\expandafter{\bbl@captionslist}%
3685 \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}}%
3686 \ifin@else
3687   \bbl@exp{%
3688     \\bbl@add\<bbl@extracaps@\language name>{\<\bbl@tempa name>}%
3689     \\bbl@toglobal\<bbl@extracaps@\language name>}%
3690   \fi
3691 \fi}

```

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

```

3692 \def\bbl@list@the{%
3693   part,chapter,section,subsection,subsubsection,paragraph,%
3694   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3695   table,page,footnote,mpfootnote,mpfn}
3696 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3697   \bbl@ifunset{bbl@map@#1@\language name}%
3698     {\@nameuse{#1}}}%
3699     {\@nameuse{bbl@map@#1@\language name}}}%
3700 \def\bbl@inikv@labels#1#2{%
3701   \in@{.map}{#1}%
3702   \ifin@
3703     \ifx\bbl@KVP@labels\@nil\else
3704       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3705     \ifin@
3706       \def\bbl@tempc{#1}%
3707       \bbl@replace\bbl@tempc{.map}{}%
3708       \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3709       \bbl@exp{%
3710         \gdef\<bbl@map@\bbl@tempc @\language name>%
3711         {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3712       \bbl@foreach\bbl@list@the{%
3713         \bbl@ifunset{the##1}{}%
3714         {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3715           \bbl@exp{%
3716             \\bbl@sreplace\<the##1>%
3717             {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3718             \\bbl@sreplace\<the##1>%
3719             {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3720           \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3721             \toks@\expandafter\expandafter\expandafter{%
3722               \csname the##1\endcsname}%
3723             \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}}%
3724           \fi}}}%
3725   \fi
3726 \fi
3727 %
3728 \else
3729   %
3730   % The following code is still under study. You can test it and make
3731   % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3732   % language dependent.
3733   \in@{enumerate.}{#1}%
3734   \ifin@
3735     \def\bbl@tempa{#1}%
3736     \bbl@replace\bbl@tempa{enumerate.}{}%
3737   \def\bbl@toreplace{#2}%

```

```

3738 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3739 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3740 \bbl@replace\bbl@toreplace{}}{\endcsname{}}}%
3741 \toks@ \expandafter{\bbl@toreplace}%
3742 \bbl@exp{%
3743   \\\bbl@add\<extras\language>{%
3744     \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3745     \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3746     \\\bbl@tglobal\<extras\language>}%
3747   \fi
3748 \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.

```

3749 \def\bbl@chapttype{chapter}
3750 \ifx\@makechapterhead\@undefined
3751   \let\bbl@patchchapter\relax
3752 \else\ifx\thechapter\@undefined
3753   \let\bbl@patchchapter\relax
3754 \else\ifx\ps@headings\@undefined
3755   \let\bbl@patchchapter\relax
3756 \else
3757   \def\bbl@patchchapter{%
3758     \global\let\bbl@patchchapter\relax
3759     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3760     \bbl@tglobal\appendix
3761     \bbl@sreplace\ps@headings
3762       {\@chapapp\ \thechapter}%
3763       {\bbl@chapterformat}%
3764     \bbl@tglobal\ps@headings
3765     \bbl@sreplace\chaptermark
3766       {\@chapapp\ \thechapter}%
3767       {\bbl@chapterformat}%
3768     \bbl@tglobal\chaptermark
3769     \bbl@sreplace\@makechapterhead
3770       {\@chapapp\space\thechapter}%
3771       {\bbl@chapterformat}%
3772     \bbl@tglobal\@makechapterhead
3773     \gdef\bbl@chapterformat{%
3774       \bbl@ifunset{bbl@\bbl@chapttype fmt@\language}%
3775       {\@chapapp\space\thechapter}
3776       {\@nameuse{bbl@\bbl@chapttype fmt@\language}}}}
3777   \let\bbl@patchappendix\bbl@patchchapter
3778 \fi\fi\fi
3779 \ifx\@part\@undefined
3780   \let\bbl@patchpart\relax
3781 \else
3782   \def\bbl@patchpart{%
3783     \global\let\bbl@patchpart\relax
3784     \bbl@sreplace\@part
3785       {\partname\nobreakspace\thepart}%
3786       {\bbl@partformat}%
3787     \bbl@tglobal\@part
3788     \gdef\bbl@partformat{%
3789       \bbl@ifunset{bbl@partfmt@\language}%
3790       {\partname\nobreakspace\thepart}
3791       {\@nameuse{bbl@partfmt@\language}}}}

```

3792 \fi

Date. TODO. Document

```
3793% Arguments are _not_ protected.
3794 \let\bbl@calendar\@empty
3795 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3796 \def\bbl@localedate#1#2#3#4{%
3797   \begingroup
3798     \ifx\@empty#1\@empty\else
3799       \let\bbl@ld@calendar\@empty
3800       \let\bbl@ld@variant\@empty
3801       \edef\bbl@tempa{\zap@space#1 \@empty}%
3802       \def\bbl@tempb##1=##2\@@{\@namedef{\bbl@ld@##1}{##2}}%
3803       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3804       \edef\bbl@calendar{%
3805         \bbl@ld@calendar
3806         \ifx\bbl@ld@variant\@empty\else
3807           .\bbl@ld@variant
3808         \fi}%
3809       \bbl@replace\bbl@calendar{gregorian}{}%
3810     \fi
3811     \bbl@cased
3812     {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3813   \endgroup}
3814% eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3815 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3816   \bbl@trim@def\bbl@tempa{#1.#2}%
3817   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3818   {\bbl@trim@def\bbl@tempa{#3}%
3819     \bbl@trim\toks@{#5}%
3820     \@temptokena\expandafter{\bbl@savestate}%
3821     \bbl@exp{% Reverse order - in ini last wins
3822       \def\\bbl@savestate{%
3823         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3824         \the\@temptokena}}%
3825     {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3826       {\lowercase{\def\bbl@tempb{#6}}%
3827         \bbl@trim@def\bbl@toreplace{#5}%
3828         \bbl@TG@@date
3829         \bbl@ifunset{\bbl@date@\language @}%
3830         {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3831           % TODO. Move to a better place.
3832           \bbl@exp{%
3833             \gdef\<\language date>{\protect\<\language date >}}%
3834             \gdef\<\language date >####1####2####3{%
3835               \\bbl@usedategroupttrue
3836               \<bbl@ensure@\language >{%
3837                 \\localedate{####1}{####2}{####3}}}%
3838             \\bbl@add\\bbl@savestate%
3839             \\SetString\\today{%
3840               \<\language date>%
3841               {\the\year}{\the\month}{\the\day}}}%
3842             {}%
3843           \ifx\bbl@tempb\@empty\else
3844             \global\bbl@csarg\let{date@\language @}\bbl@tempb\bbl@toreplace
3845           \fi}%
3846         {}}}
```

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.

```

3847 \let\bbl@calendar\@empty
3848 \newcommand\BabelDateSpace{\nobreakspace}
3849 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3850 \newcommand\BabelDated[1]{\number#1}
3851 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3852 \newcommand\BabelDateM[1]{\number#1}
3853 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3854 \newcommand\BabelDateMMMM[1]{\%
3855 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3856 \newcommand\BabelDatey[1]{\number#1}%
3857 \newcommand\BabelDateyy[1]{\%
3858 \ifnum#1<10 0\number#1 %
3859 \else\ifnum#1<100 \number#1 %
3860 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3861 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3862 \else
3863 \bbl@error
3864 {Currently two-digit years are restricted to the\
3865 range 0-9999.}%
3866 {There is little you can do. Sorry.}%
3867 \fi\fi\fi}
3868 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3869 \def\bbl@replace@finish@iii#1{%
3870 \bbl@exp{\def\#1###1####2###3{\the\toks@}}
3871 \def\bbl@TG@date{%
3872 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3873 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot}}%
3874 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3875 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3876 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3877 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3878 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3879 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3880 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3881 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3882 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3883 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3884 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3885 % Note after \bbl@replace \toks@ contains the resulting string.
3886 % TODO - Using this implicit behavior doesn't seem a good idea.
3887 \bbl@replace@finish@iii\bbl@toreplace}
3888 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3889 \def\bbl@xdatecctr[#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.

```

3890 \def\bbl@provide@lsys#1{%
3891 \bbl@ifunset{bbl@lname@#1}%
3892 {\bbl@load@info{#1}}%
3893 {}%
3894 \bbl@csarg\let{lsys@#1}\@empty
3895 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3896 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3897 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3898 \bbl@ifunset{bbl@lname@#1}{%
3899 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3900 \ifcase\bbl@engine\or\or
3901 \bbl@ifunset{bbl@prehc@#1}{}%

```

```

3902      {\bbl@exp{\bb@ifblank{\bbl@cs{prehc@#1}}}%
3903      }%
3904      {\ifx\bbl@xenohyph\undefined
3905       \let\bbl@xenohyph\bbl@xenohyph@d
3906       \ifx\AtBeginDocument\@notprerr
3907        \expandafter\@secondoftwo % to execute right now
3908       \fi
3909       \AtBeginDocument{%
3910        \expandafter\bbl@add
3911        \csname selectfont \endcsname{\bbl@xenohyph}%
3912        \expandafter\selectlanguage\expandafter{\language}%
3913        \expandafter\bbl@toglobal\csname selectfont \endcsname}%
3914      \fi}%
3915 \fi
3916 \bbl@csarg\bbl@toglobal{\sys@#1}}
3917 \def\bbl@xenohyph@d{%
3918   \bbl@ifset{\bbl@prehc\language}%
3919   {\ifnum\hyphenchar\font=\defaultshyphenchar
3920    \iffontchar\font\bbl@cl{prehc}\relax
3921    \hyphenchar\font\bbl@cl{prehc}\relax
3922   \else\iffontchar\font"200B
3923    \hyphenchar\font"200B
3924   \else
3925    \bbl@warning
3926    {Neither 0 nor ZERO WIDTH SPACE are available\\%
3927     in the current font, and therefore the hyphen\\%
3928     will be printed. Try changing the fontspec's\\%
3929     'HyphenChar' to another value, but be aware\\%
3930     this setting is not safe (see the manual))}%
3931    \hyphenchar\font\defaultshyphenchar
3932   \fi\fi
3933   \fi}%
3934   {\hyphenchar\font\defaultshyphenchar}}
3935 % \fi}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3936 \def\bbl@load@info#1{%
3937   \def\BabelBeforeIni##1##2{%
3938     \begingroup
3939     \bbl@read@ini{##1}0%
3940     \endinput          % babel- .tex may contain onlypreamble's
3941     \endgroup}%        boxed, to avoid extra spaces:
3942   {\bbl@input@texini{#1}}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in \TeX . Non-digits characters are kept. The first macro is the generic “localized” command.

```

3943 \def\bbl@setdigits#1#2#3#4#5{%
3944   \bbl@exp{%
3945     \def\<\language digits>####1{%      ie, \langdigits
3946       \<\bbl@digits@\language>####1\@nil}%
3947       \let\<\bbl@cntr@digits@\language>\<\language digits>%
3948       \def\<\language counter>####1{%    ie, \langcounter
3949         \expandafter\<\bbl@counter@\language>%
3950         \csname c@####1\endcsname}%
3951       \def\<\bbl@counter@\language>####1{% ie, \bbl@counter@lang

```


[illegible]

```

3974 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3975   \ifx\\#1%                % \\ before, in case #1 is multiletter
3976     \bbl@exp{%
3977       \def\\bbl@tempa####1{%
3978         \<ifcase>####1\space\the\toks@\<else>\\<ctrerr>\<fi>}}%
3979   \else
3980     \toks@\expandafter{\the\toks@\or #1}%
3981     \expandafter\bbl@buildifcase
3982   \fi}

```

```

3983 \newcommand\locallenumerical[2]{\bbl@cs{cnt@#1\language}\#2}%
3984 \def\bbl@localecnt#1#2{\locallenumerical{#2}{#1}}
3985 \newcommand\localecounter[2]{%
3986   \expandafter\bbl@localecnt#1
3987   \expandafter{\number\csname c@#2\endcsname}\#1}%
3988 \def\bbl@alphnumerical#1#2{%
3989   \expandafter\bbl@alphnumerical@i\number#2 76543210@@{#1}}
3990 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
3991   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
3992     \bbl@alphnumerical@ii{#9}000000#1\or
3993     \bbl@alphnumerical@ii{#9}00000#1#2\or
3994     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3995     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3996     \bbl@alphnum@invalid{>9999}%
3997   \fi}
3998 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
3999   \bbl@ifunset{bbl@cnt@#1.F.\number#5#6#7#8@\language}%
4000   {\bbl@cs{cnt@#1.4@\language}\#5%
4001     \bbl@cs{cnt@#1.3@\language}\#6%
4002     \bbl@cs{cnt@#1.2@\language}\#7%

```

```

4003 \bbl@cs{cntr@#1.1@\language}\language#8%
4004 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4005 \bbl@ifunset{bbl@cntr@#1.S.321@\language}\language}%
4006 {\bbl@cs{cntr@#1.S.321@\language}\language}%
4007 \fi}%
4008 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}\language}}
4009 \def\bbl@alphnum@invalid#1{%
4010 \bbl@error{Alphabetic numeral too large (#1)}%
4011 {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.

```

4012 \newcommand\localeinfo[1]{%
4013 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
4014 {\bbl@error{I've found no info for the current locale.\%
4015 The corresponding ini file has not been loaded\%
4016 Perhaps it doesn't exist}%
4017 {See the manual for details.}}%
4018 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}%
4019 \namedef{bbl@info@name.locale}{lname}
4020 \namedef{bbl@info@tag.ini}{lini}
4021 \namedef{bbl@info@name.english}{elname}
4022 \namedef{bbl@info@name.opentype}{lname}
4023 \namedef{bbl@info@tag.bcp47}{tbc}
4024 \namedef{bbl@info@language.tag.bcp47}{lbc}
4025 \namedef{bbl@info@tag.opentype}{lotf}
4026 \namedef{bbl@info@script.name}{esname}
4027 \namedef{bbl@info@script.name.opentype}{sname}
4028 \namedef{bbl@info@script.tag.bcp47}{sbc}
4029 \namedef{bbl@info@script.tag.opentype}{sotf}
4030 \let\bbl@ensureinfo\@gobble
4031 \newcommand\BabelEnsureInfo{%
4032 \ifx\InputIfFileExists\undefined\else
4033 \def\bbl@ensureinfo##1{%
4034 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
4035 \fi
4036 \bbl@foreach\bbl@loaded{%
4037 \def\language{##1}%
4038 \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`.

```

4039 \newcommand\getlocaleproperty{%
4040 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4041 \def\bbl@getproperty@s#1#2#3{%
4042 \let#1\relax
4043 \def\bbl@elt##1##2##3{%
4044 \bbl@ifsamestring{##1/##2}{##3}%
4045 {\providecommand#1{##3}%
4046 \def\bbl@elt####1####2####3{}}}%
4047 {}}%
4048 \bbl@cs{inidata@#2}}%
4049 \def\bbl@getproperty@x#1#2#3{%
4050 \bbl@getproperty@s{#1}{#2}{#3}%
4051 \ifx#1\relax
4052 \bbl@error
4053 {Unknown key for locale '#2':\%
4054 #3}%

```

```

4055     \string#1 will be set to \relax}%
4056     {Perhaps you misspelled it.}%
4057   \fi}
4058 \let\bbl@ini@loaded\@empty
4059 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel bahavior

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

```

4060 \newcommand\babeladjust[1]{% TODO. Error handling.
4061   \bbl@forkv{#1}{%
4062     \bbl@ifunset{\bbl@ADJ@##1@##2}%
4063     {\bbl@cs{ADJ@##1}{##2}}}%
4064     {\bbl@cs{ADJ@##1@##2}}}%
4065 %
4066 \def\bbl@adjust@lua#1#2{%
4067   \ifvmode
4068     \ifnum\currentgrouplevel=\z@
4069       \directlua{ Babel.#2 }%
4070       \expandafter\expandafter\expandafter\@gobble
4071     \fi
4072   \fi
4073   {\bbl@error % The error is gobbled if everything went ok.
4074     {Currently, #1 related features can be adjusted only\\%
4075       in the main vertical list.}%
4076     {Maybe things change in the future, but this is what it is.}}}
4077 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
4078   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4079 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
4080   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4081 \@namedef{\bbl@ADJ@bidi.text@on}{%
4082   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4083 \@namedef{\bbl@ADJ@bidi.text@off}{%
4084   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4085 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
4086   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4087 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
4088   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4089 %
4090 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
4091   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4092 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
4093   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4094 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
4095   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4096 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
4097   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4098 %
4099 \def\bbl@adjust@layout#1{%
4100   \ifvmode
4101     #1%
4102     \expandafter\@gobble
4103   \fi
4104   {\bbl@error % The error is gobbled if everything went ok.
4105     {Currently, layout related features can be adjusted only\\%
4106       in vertical mode.}%
4107     {Maybe things change in the future, but this is what it is.}}}
4108 \@namedef{\bbl@ADJ@layout.tabular@on}{%

```

```

4109 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4110 \@namedef{bbl@ADJ@layout.tabular@off}{%
4111 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4112 \@namedef{bbl@ADJ@layout.lists@on}{%
4113 \bbl@adjust@layout{\let\list\bbl@NL@list}}
4114 \@namedef{bbl@ADJ@layout.lists@off}{%
4115 \bbl@adjust@layout{\let\list\bbl@OL@list}}
4116 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4117 \bbl@activateposthyphen}
4118 %
4119 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4120 \bbl@bcpallowedtrue}
4121 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4122 \bbl@bcpallowedfalse}
4123 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
4124 \def\bbl@bcp@prefix{#1}}
4125 \def\bbl@bcp@prefix{bcp47-}
4126 \@namedef{bbl@ADJ@autoload.options#1}{%
4127 \def\bbl@autoload@options{#1}}
4128 \let\bbl@autoload@bcptoptions\@empty
4129 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
4130 \def\bbl@autoload@bcptoptions{#1}}
4131 \newif\ifbbl@bcptoname
4132 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4133 \bbl@bcptonametrue}
4134 \BabelEnsureInfo}
4135 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4136 \bbl@bcptonamefalse}
4137 % TODO: use babel name, override
4138 %
4139 % As the final task, load the code for lua.
4140 %
4141 \ifx\directlua\@undefined\else
4142 \ifx\bbl@luapatterns\@undefined
4143 \input luababel.def
4144 \fi
4145 \fi
4146 </core>

A proxy file for switch.def
4147 <*kernel>
4148 \let\bbl@onlyswitch\@empty
4149 \input babel.def
4150 \let\bbl@onlyswitch\@undefined
4151 </kernel>
4152 <*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.

```

4153 <<Make sure ProvidesFile is defined>>
4154 \ProvidesFile{hyphen.cfg}[\<<date>>] \<<version>> Babel hyphens]
4155 \xdef\bbl@format{\jobname}
4156 \def\bbl@version{\<<version>>}
4157 \def\bbl@date{\<<date>>}
4158 \ifx\AtBeginDocument\@undefined
4159   \def\@empty{}
4160   \let\orig@dump\dump
4161   \def\dump{%
4162     \ifx\@ztryfc\@undefined
4163       \else
4164         \toks0=\expandafter{\@preamblecmds}%
4165         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4166         \def\@begindocumenthook{}%
4167       \fi
4168       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4169 \fi
4170 <<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.

```

4171 \def\process@line#1#2 #3 #4 {%
4172   \ifx=#1%
4173     \process@synonym{#2}%
4174   \else
4175     \process@language{#1#2}{#3}{#4}%
4176   \fi
4177   \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.

```

4178 \toks@{}
4179 \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.

```

4180 \def\process@synonym#1{%
4181   \ifnum\last@language=\m@ne
4182     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4183   \else
4184     \expandafter\chardef\csname l@#1\endcsname\last@language
4185     \wlog{\string\l@#1=\string\language\the\last@language}%
4186     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4187       \csname\languagename hyphenmins\endcsname
4188     \let\bbl@elt\relax
4189     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}%
4190   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```

4191 \def\process@language#1#2#3{%
4192   \expandafter\addlanguage\csname l@#1\endcsname
4193   \expandafter\language\csname l@#1\endcsname
4194   \edef\language#1#2#3{%
4195     \bbl@hook@everylanguage{#1}%
4196     % > luatex
4197     \bbl@get@enc#1::@@@
4198     \begin{group}
4199       \lefthyphenmin\m@ne
4200       \bbl@hook@loadpatterns{#2}%
4201       % > luatex
4202       \ifnum\lefthyphenmin=\m@ne
4203         \else
4204           \expandafter\xdef\csname #1hyphenmins\endcsname{%
4205             \the\lefthyphenmin\the\righthyphenmin}%
4206           \fi
4207       \end{group}
4208       \def\bbl@tempa{#3}%
4209       \ifx\bbl@tempa\@empty\else
4210         \bbl@hook@loadexceptions{#3}%
4211         % > luatex
4212       \fi
4213       \let\bbl@elt\relax
4214       \edef\bbl@languages{%
4215         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4216       \ifnum\the\language=\z@
4217         \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4218           \set@hyphenmins\tw@\thr@@\relax
4219         \else
4220           \expandafter\expandafter\expandafter\set@hyphenmins
4221             \csname #1hyphenmins\endcsname
4222         \fi
4223         \the\toks@
4224         \toks@{}%
4225       \fi}

```

`\bbl@get@enc` The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in
`\bbl@hyph@enc`

\bbl@hyph@enc. It uses delimited arguments to achieve this.

```
4226 \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.

```
4227 \def\bbl@hook@everylanguage#1{}
4228 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4229 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4230 \def\bbl@hook@loadkernel#1{%
4231   \def\addlanguage{\csname newlanguage\endcsname}%
4232   \def\adddialect##1##2{%
4233     \global\chardef##1##2\relax
4234     \wlog{\string##1 = a dialect from \string\language##2}}%
4235   \def\iflanguage##1{%
4236     \expandafter\ifx\csname l@##1\endcsname\relax
4237       \@nolanerr{##1}%
4238     \else
4239       \ifnum\csname l@##1\endcsname=\language
4240         \expandafter\expandafter\expandafter\@firstoftwo
4241       \else
4242         \expandafter\expandafter\expandafter\@secondoftwo
4243       \fi
4244     \fi}%
4245   \def\providehyphenmins##1##2{%
4246     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4247       \@namedef{##1hyphenmins}{##2}%
4248     \fi}%
4249   \def\set@hyphenmins##1##2{%
4250     \lefthyphenmin##1\relax
4251     \righthyphenmin##2\relax}%
4252   \def\selectlanguage{%
4253     \errhelp{Selecting a language requires a package supporting it}%
4254     \errmessage{Not loaded}}%
4255   \let\foreignlanguage\selectlanguage
4256   \let\otherlanguage\selectlanguage
4257   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4258   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4259     \def\setlocale{%
4260       \errhelp{Find an armchair, sit down and wait}%
4261       \errmessage{Not yet available}}%
4262     \let\uselocale\setlocale
4263     \let\locale\setlocale
4264     \let\selectlocale\setlocale
4265     \let\localename\setlocale
4266     \let\textlocale\setlocale
4267     \let\textlanguage\setlocale
4268     \let\languagetext\setlocale}
4269   \begingroup
4270   \def\AddBabelHook#1#2{%
4271     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4272       \def\next{\toks1}%
4273     \else
4274       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4275     \fi
4276     \next}
4277   \ifx\directlua\@undefined
4278     \ifx\XeTeXinputencoding\@undefined\else
```

```

4279     \input xebabel.def
4280     \fi
4281 \else
4282     \input luababel.def
4283 \fi
4284 \openin1 = babel-\bbl@format.cfg
4285 \ifeof1
4286 \else
4287     \input babel-\bbl@format.cfg\relax
4288 \fi
4289 \closein1
4290 \endgroup
4291 \bbl@hook@loadkernel{switch.def}

```

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

```

4292 \openin1 = language.dat

```

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

```

4293 \def\language{english}%
4294 \ifeof1
4295     \message{I couldn't find the file language.dat,\space
4296             I will try the file hyphen.tex}
4297     \input hyphen.tex\relax
4298     \chardef\l@english\z@
4299 \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 .

```

4300 \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.

```

4301 \loop
4302     \endlinechar\m@ne
4303     \read1 to \bbl@line
4304     \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.

```

4305     \if T\ifeof1F\fi T\relax
4306     \ifx\bbl@line\@empty\else
4307         \edef\bbl@line{\bbl@line\space\space\space}%
4308         \expandafter\process@line\bbl@line\relax
4309     \fi
4310 \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.

```

4311 \begingroup
4312 \def\bbl@elt#1#2#3#4{%
4313     \global\language=#2\relax
4314     \gdef\language{#1}%
4315     \def\bbl@elt##1##2##3##4{}}%
4316 \bbl@languages

```



```

4317 \endgroup
4318 \fi
4319 \closein1

```

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

```

4320 \if\the\toks@\else
4321 \errhelp{language.dat loads no language, only synonyms}
4322 \errmessage{Orphan language synonym}
4323 \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.

```

4324 \let\bbl@line\@undefined
4325 \let\process@line\@undefined
4326 \let\process@synonym\@undefined
4327 \let\process@language\@undefined
4328 \let\bbl@get@enc\@undefined
4329 \let\bbl@hyph@enc\@undefined
4330 \let\bbl@tempa\@undefined
4331 \let\bbl@hook@loadkernel\@undefined
4332 \let\bbl@hook@everylanguage\@undefined
4333 \let\bbl@hook@loadpatterns\@undefined
4334 \let\bbl@hook@loadexceptions\@undefined
4335 \patterns)

```

Here the code for `iniTeX` ends.

12 Font handling with fontspec

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

```

4336 <<(*More package options)>> ≡
4337 \chardef\bbl@bidimode\z@
4338 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4339 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4340 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4341 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4342 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4343 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4344 <</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.

```

4345 <<(*Font selection)>> ≡
4346 \bbl@trace{Font handling with fontspec}
4347 \ifx\ExplSyntaxOn\@undefined\else
4348 \ExplSyntaxOn
4349 \catcode`\ =10
4350 \def\bbl@loadfontspec{%
4351 \usepackage{fontspec}%
4352 \expandafter
4353 \def\csname msg-text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4354 Font '\l_fontspec_fontname_tl' is using the\%
4355 default features for language '##1'.\\%

```

```

4356     That's usually fine, because many languages\\%
4357     require no specific features, but if the output is\\%
4358     not as expected, consider selecting another font.}
4359 \expandafter
4360 \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4361     Font '\l_fontspec_fontname_tl' is using the\\%
4362     default features for script '##2'.\\%
4363     That's not always wrong, but if the output is\\%
4364     not as expected, consider selecting another font.}}
4365 \ExplSyntaxOff
4366 \fi
4367 \@onlypreamble\babelfont
4368 \newcommand\babelfont[2][{% 1=langs/scripts 2=fam
4369 \bbl@foreach{#1}{%
4370 \expandafter\ifx\csname date##1\endcsname\relax
4371 \IfFileExists{babel-##1.tex}%
4372 {\babelprovide{##1}}%
4373 }{%
4374 \fi}%
4375 \edef\bbl@tempa{#1}%
4376 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4377 \ifx\fontspec\@undefined
4378 \bbl@loadfontspec
4379 \fi
4380 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4381 \bbl@bblfont}
4382 \newcommand\bbl@bblfont[2][{% 1=features 2=fontname, @font=rm|sf|tt
4383 \bbl@ifunset{\bbl@tempb family}%
4384 {\bbl@providedefam{\bbl@tempb}}%
4385 {\bbl@exp{%
4386 \\\bbl@sreplace\<\bbl@tempb family >%
4387 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4388 % For the default font, just in case:
4389 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4390 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4391 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4392 \bbl@exp{%
4393 \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4394 \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4395 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4396 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4397 \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:

```

4398 \def\bbl@providedefam#1{%
4399 \bbl@exp{%
4400 \\\newcommand\<#1default>{% Just define it
4401 \\\bbl@add@list\\\bbl@font@fams{#1}%
4402 \\\DeclareRobustCommand\<#1family>{%
4403 \\\not@math@alphabet\<#1family>\relax
4404 \\\fontfamily\<#1default>\selectfont}%
4405 \\\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.

```

4406 \def\bbl@nostdfont#1{%
4407 \bbl@ifunset{\bbl@WFF@\f@family}%
4408 {\bbl@csarg\gdef{WFF@\f@family}}% Flag, to avoid dupl warns
4409 \bbl@infowarn{The current font is not a babel standard family:\\%

```

```

4410      #1%
4411      \fontname\font\\%
4412      There is nothing intrinsically wrong with this warning, and\\%
4413      you can ignore it altogether if you do not need these\\%
4414      families. But if they are used in the document, you should be\\%
4415      aware 'babel' will no set Script and Language for them, so\\%
4416      you may consider defining a new family with \string\babelfont.\\%
4417      See the manual for further details about \string\babelfont.\\%
4418      Reported}}
4419  {}}%
4420 \gdef\bbbl@switchfont{%
4421  \bbbl@ifunset{bbbl@sys@\language}{\bbbl@provide@sys{\language}}}%
4422  \bbbl@exp{% eg Arabic -> arabic
4423  \lowercase{\edef\\bbbl@tempa{\bbbl@cl{sname}}}}%
4424  \bbbl@foreach\bbbl@font@fams{%
4425  \bbbl@ifunset{bbbl@##1dflt@\language}% (1) language?
4426  {\bbbl@ifunset{bbbl@##1dflt@*\bbbl@tempa}% (2) from script?
4427  {\bbbl@ifunset{bbbl@##1dflt@}% 2=F - (3) from generic?
4428  {}% 123=F - nothing!
4429  {\bbbl@exp{% 3=T - from generic
4430  \global\let<bbbl@##1dflt@\language>%
4431  \<bbbl@##1dflt@>}}}%
4432  {\bbbl@exp{% 2=T - from script
4433  \global\let<bbbl@##1dflt@\language>%
4434  \<bbbl@##1dflt@*\bbbl@tempa>}}}%
4435  {}}% 1=T - language, already defined
4436  \def\bbbl@tempa{\bbbl@nostdfont{}}%
4437  \bbbl@foreach\bbbl@font@fams{% don't gather with prev for
4438  \bbbl@ifunset{bbbl@##1dflt@\language}%
4439  {\bbbl@cs{famrst@##1}%
4440  \global\bbbl@csarg\let{famrst@##1}\relax}%
4441  {\bbbl@exp{% order is relevant. TODO: but sometimes wrong!
4442  \\bbbl@add\\originalTeX{%
4443  \\bbbl@font@rst{\bbbl@cl{##1dflt}}}%
4444  \<##1default>\<##1family>{##1}}}%
4445  \\bbbl@font@set<bbbl@##1dflt@\language>% the main part!
4446  \<##1default>\<##1family>}}}%
4447  \bbbl@ifrestoring{ }\bbbl@tempa}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

4448 \ifx\f@family\undefined\else % if latex
4449 \ifcase\bbbl@engine % if pdftex
4450 \let\bbbl@ckeckstdfonts\relax
4451 \else
4452 \def\bbbl@ckeckstdfonts{%
4453 \begingroup
4454 \global\let\bbbl@ckeckstdfonts\relax
4455 \let\bbbl@tempa\empty
4456 \bbbl@foreach\bbbl@font@fams{%
4457 \bbbl@ifunset{bbbl@##1dflt@}%
4458 {\@nameuse{##1family}%
4459 \bbbl@csarg\gdef{WFF@f@family}}}% Flag
4460 \bbbl@exp{\\bbbl@add\\bbbl@tempa{* \<##1family>= \f@family\\}%
4461 \space\space\fontname\font\\}%
4462 \bbbl@csarg\xdef{##1dflt@}{\f@family}%
4463 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4464 {}}%
4465 \ifx\bbbl@tempa\empty\else

```

```

4466         \bbl@infowarn{The following font families will use the default\\%
4467         settings for all or some languages:\\%
4468         \bbl@tempa
4469         There is nothing intrinsically wrong with it, but\\%
4470         'babel' will no set Script and Language, which could\\%
4471         be relevant in some languages. If your document uses\\%
4472         these families, consider redefining them with \string\babelfont.\\%
4473         Reported}%
4474     \fi
4475 \endgroup}
4476 \fi
4477 \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.

```

4478 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4479 \bbl@xin@{<>}{#1}%
4480 \ifin@
4481 \bbl@exp{\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4482 \fi
4483 \bbl@exp{%          'Unprotected' macros return prev values
4484 \def\\#2{#1}%      eg, \rmdefault{\bbl@rmdflt@lang}
4485 \\bbl@ifsamestring{#2}{\f@family}%
4486 {\\#3%
4487 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}}%
4488 \let\\bbl@tempa\relax}%
4489 {}}
4490 %      TODO - next should be global?, but even local does its job. I'm
4491 %      still not sure -- must investigate:
4492 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4493 \let\bbl@tempa\bbl@mapselect
4494 \let\bbl@mapselect\relax
4495 \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4496 \let#4\@empty %      Make sure \renewfontfamily is valid
4497 \bbl@exp{%
4498 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4499 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4500 {\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4501 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4502 {\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4503 \\renewfontfamily\\#4%
4504 [\bbl@cs{lsys@\languagename},#2]{#3}% ie \bbl@exp{.}{#3}
4505 \begingroup
4506 #4%
4507 \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4508 \endgroup
4509 \let#4\bbl@temp@fam
4510 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4511 \let\bbl@mapselect\bbl@tempa}%

```

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

```

4512 \def\bbl@font@rst#1#2#3#4{%
4513 \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.

```

4514 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for `\babelFSfeatures`. The reason is explained in the user guide, but essentially – that was not the way to go :-).

```

4515 \newcommand\babelFSstore[2][{}%
4516   \bbl@ifblank{#1}%
4517   {\bbl@csarg\def\sname@#2}{Latin}}%
4518   {\bbl@csarg\def\sname@#2}{#1}}%
4519   \bbl@provide@dirs{#2}%
4520   \bbl@csarg\ifnum{wdir@#2}>\z@
4521     \let\bbl@beforeforeign\leavevmode
4522     \EnableBabelHook{babel-bidi}%
4523   \fi
4524   \bbl@foreach{#2}{%
4525     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4526     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4527     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4528 \def\bbl@FSstore#1#2#3#4{%
4529   \bbl@csarg\edef{#2default#1}{#3}%
4530   \expandafter\addto\csname extras#1\endcsname{%
4531     \let#4#3%
4532     \ifx#3\f@family
4533       \edef#3{\csname bbl@#2default#1\endcsname}%
4534       \fontfamily{#3}\selectfont
4535     \else
4536       \edef#3{\csname bbl@#2default#1\endcsname}%
4537       \fi}%
4538   \expandafter\addto\csname noextras#1\endcsname{%
4539     \ifx#3\f@family
4540       \fontfamily{#4}\selectfont
4541       \fi
4542     \let#3#4}}
4543 \let\bbl@langfeatures\@empty
4544 \def\babelFSfeatures{% make sure \fontspec is redefined once
4545   \let\bbl@ori@fontspec\fontspec
4546   \renewcommand\fontspec[1][{}%
4547     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4548   \let\babelFSfeatures\bbl@FSfeatures
4549   \babelFSfeatures}
4550 \def\bbl@FSfeatures#1#2{%
4551   \expandafter\addto\csname extras#1\endcsname{%
4552     \babel@save\bbl@langfeatures
4553     \edef\bbl@langfeatures{#2,}}
4554 \</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.

```

4555 \<{*Footnote changes}> \equiv
4556 \bbl@trace{Bidi footnotes}
4557 \ifnum\bbl@bidimode>\z@
4558   \def\bbl@footnote#1#2#3{%
4559     \@ifnextchar[%
4560       {\bbl@footnote@o{#1}{#2}{#3}}%
4561       {\bbl@footnote@x{#1}{#2}{#3}}}

```

```

4562 \long\def\bbl@footnote@x#1#2#3#4{%
4563   \bgroup
4564     \select@language@x{\bbl@main@language}%
4565     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4566   \egroup}
4567 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4568   \bgroup
4569     \select@language@x{\bbl@main@language}%
4570     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4571   \egroup}
4572 \def\bbl@footnotetext#1#2#3{%
4573   \@ifnextchar[%
4574     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4575     {\bbl@footnotetext@x{#1}{#2}{#3}}%
4576 \long\def\bbl@footnotetext@x#1#2#3#4{%
4577   \bgroup
4578     \select@language@x{\bbl@main@language}%
4579     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4580   \egroup}
4581 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4582   \bgroup
4583     \select@language@x{\bbl@main@language}%
4584     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4585   \egroup}
4586 \def\BabelFootnote#1#2#3#4{%
4587   \ifx\bbl@fn@footnote\@undefined
4588     \let\bbl@fn@footnote\footnote
4589   \fi
4590   \ifx\bbl@fn@footnotetext\@undefined
4591     \let\bbl@fn@footnotetext\footnotetext
4592   \fi
4593   \bbl@ifblank{#2}%
4594     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4595     \@namedef{\bbl@stripslash#1text}%
4596     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4597     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4598     \@namedef{\bbl@stripslash#1text}%
4599     {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4600 \fi
4601 <</Footnote changes>>

```

Now, the code.

```

4602 (*xetex)
4603 \def\BabelStringsDefault{unicode}
4604 \let\xebbl@stop\relax
4605 \AddBabelHook{xetex}{encodedcommands}{%
4606   \def\bbl@tempa{#1}%
4607   \ifx\bbl@tempa\@empty
4608     \XeTeXinputencoding"bytes"%
4609   \else
4610     \XeTeXinputencoding"#1"%
4611   \fi
4612   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4613 \AddBabelHook{xetex}{stopcommands}{%
4614   \xebbl@stop
4615   \let\xebbl@stop\relax}
4616 \def\bbl@intraspace#1 #2 #3\@@{%
4617   \bbl@csarg\gdef{\xeisp@language}%
4618   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}

```

```

4619 \def\bbl@intrapenalty#1\@@{%
4620   \bbl@csarg\gdef{xeipn@\language}%
4621   {\XeTeXlinebreakpenalty #1\relax}}
4622 \def\bbl@provide@intraspace{%
4623   \bbl@xin@\bbl@cl{\lnbrk}}{s}%
4624   \ifin@else\bbl@xin@\bbl@cl{\lnbrk}}{c}\fi
4625   \ifin@
4626     \bbl@ifunset{\bbl@intsp@\language}{}%
4627     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4628       \ifx\bbl@KVP@intraspace\@nil
4629         \bbl@exp{%
4630           \\bbl@intraspace\bbl@cl{intsp}}\\@}%
4631       \fi
4632       \ifx\bbl@KVP@intrapenalty\@nil
4633         \bbl@intrapenalty0@@
4634       \fi
4635     \fi
4636     \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4637       \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4638     \fi
4639     \ifx\bbl@KVP@intrapenalty\@nil\else
4640       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4641     \fi
4642     \bbl@exp{%
4643       \\bbl@add\<extras\language>%
4644       \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4645       \<bbl@xeisp@\language>%
4646       \<bbl@xeipn@\language>%
4647       \\bbl@tglobal\<extras\language>%
4648       \\bbl@add\<noextras\language>%
4649       \XeTeXlinebreaklocale "en"%
4650       \\bbl@tglobal\<noextras\language>}%
4651     \ifx\bbl@ispace\@undefined
4652       \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4653     \ifx\AtBeginDocument\@notprerr
4654       \expandafter\@secondoftwo % to execute right now
4655     \fi
4656     \AtBeginDocument{%
4657       \expandafter\bbl@add
4658       \csname selectfont \endcsname{\bbl@ispace}%
4659       \expandafter\bbl@tglobal\csname selectfont \endcsname}%
4660     \fi}%
4661   \fi}
4662 \ifx\DisableBabelHook\@undefined\endinput\fi
4663 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4664 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4665 \DisableBabelHook{babel-fontspec}
4666 <<Font selection>>
4667 \input txtbabel.def
4668 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titlesp, and geometry.

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

Consider `txtbabel` as a shorthand for *tex-xet babel*, which is the bidi model in both `pdfTeX` and `xetex`.

```

4669 (*texxet)
4670 \providecommand\bbl@provide@intraspace{}
4671 \bbl@trace{Redefinitions for bidi layout}
4672 \def\bbl@sspre@caption{%
4673   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}}
4674 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4675 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4676 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4677 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4678   \def\@hangfrom#1{%
4679     \setbox\@tempboxa\hbox{#1}%
4680     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4681     \noindent\box\@tempboxa}
4682 \def\raggedright{%
4683   \let\@centercr
4684   \bbl@startskip\z@skip
4685   \@rightskip\@flushglue
4686   \bbl@endskip\@rightskip
4687   \parindent\z@
4688   \parfillskip\bbl@startskip}
4689 \def\raggedleft{%
4690   \let\@centercr
4691   \bbl@startskip\@flushglue
4692   \bbl@endskip\z@skip
4693   \parindent\z@
4694   \parfillskip\bbl@endskip}
4695 \fi
4696 \IfBabelLayout{lists}
4697   {\bbl@sreplace\list
4698     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4699     \def\bbl@listleftmargin{%
4700       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4701     \ifcase\bbl@engine
4702       \def\labelenumii{}\theenumii{}% pdfTeX doesn't reverse ()
4703       \def\p@enumiii{\p@enumii}\theenumii{}%
4704     \fi
4705     \bbl@sreplace\@verbatim
4706       {\leftskip\@totalleftmargin}%
4707       {\bbl@startskip\textwidth
4708         \advance\bbl@startskip-\linewidth}%
4709     \bbl@sreplace\@verbatim
4710       {\rightskip\z@skip}%
4711       {\bbl@endskip\z@skip}}%
4712   {}
4713 \IfBabelLayout{contents}
4714   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4715     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4716   {}
4717 \IfBabelLayout{columns}
4718   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4719     \def\bbl@outputbox#1{%
4720       \hb@xt@\textwidth{%
4721         \hskip\columnwidth
4722         \hfil
4723         {\normalcolor\vrule \@width\columnseprule}%
4724         \hfil
4725         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%

```



```

4726      \hskip-\textwidth
4727      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4728      \hskip\columnsep
4729      \hskip\columnwidth}}}%
4730  {}
4731  <<Footnote changes>>
4732  \IfBabelLayout{footnotes}%
4733  {\BabelFootnote\footnote\language\language\language}%
4734  \BabelFootnote\localfootnote\language\language}%
4735  \BabelFootnote\mainfootnote\language\language}%
4736  {}

```

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.

```

4737 \IfBabelLayout{counters}%
4738  {\let\bbl@latinarabic=\@arabic
4739   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4740   \let\bbl@asciroman=\@roman
4741   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4742   \let\bbl@asciiRoman=\@Roman
4743   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
4744 </texet>

```

13.3 LuaTeX

The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4745 <*\luatex>

```

```

4746 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4747 \bbl@trace{Read language.dat}
4748 \ifx\bbl@readstream\undefined
4749 \csname newread\endcsname\bbl@readstream
4750 \fi
4751 \beginingroup
4752 \toks@{}
4753 \count@\z@ % 0=start, 1=0th, 2=normal
4754 \def\bbl@process@line#1#2 #3 #4 {%
4755   \ifx=#1%
4756     \bbl@process@synonym{#2}%
4757   \else
4758     \bbl@process@language{#1#2}{#3}{#4}%
4759   \fi
4760   \ignorespaces}
4761 \def\bbl@manylang{%
4762   \ifnum\bbl@last>\@ne
4763     \bbl@info{Non-standard hyphenation setup}%
4764   \fi
4765   \let\bbl@manylang\relax}
4766 \def\bbl@process@language#1#2#3{%
4767   \ifcase\count@
4768     \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4769   \or
4770     \count@\tw@
4771   \fi
4772   \ifnum\count@=\tw@
4773     \expandafter\addlanguage\csname l@#1\endcsname
4774     \language\allocationnumber
4775     \chardef\bbl@last\allocationnumber
4776     \bbl@manylang
4777     \let\bbl@elt\relax
4778     \xdef\bbl@languages{%
4779       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4780   \fi
4781   \the\toks@
4782   \toks@{}}
4783 \def\bbl@process@synonym@aux#1#2{%
4784   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4785   \let\bbl@elt\relax
4786   \xdef\bbl@languages{%
4787     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4788 \def\bbl@process@synonym#1{%
4789   \ifcase\count@
4790     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4791   \or
4792     \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4793   \else
4794     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4795   \fi}
4796 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4797   \chardef\l@english\z@
4798   \chardef\l@USenglish\z@
4799   \chardef\bbl@last\z@
4800   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4801   \gdef\bbl@languages{%
4802     \bbl@elt{english}{0}{hyphen.tex}}%
4803   \bbl@elt{USenglish}{0}{}
4804 \else

```

```

4805 \global\let\bbl@languages@format\bbl@languages
4806 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4807 \ifnum#2>\z@\else
4808 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4809 \fi}%
4810 \xdef\bbl@languages{\bbl@languages}%
4811 \fi
4812 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4813 \bbl@languages
4814 \openin\bbl@readstream=language.dat
4815 \ifeof\bbl@readstream
4816 \bbl@warning{I couldn't find language.dat. No additional\\%
4817 patterns loaded. Reported}%
4818 \else
4819 \loop
4820 \endlinechar\m@ne
4821 \read\bbl@readstream to \bbl@line
4822 \endlinechar\^^M
4823 \if T\ifeof\bbl@readstream F\fi T\relax
4824 \ifx\bbl@line\empty\else
4825 \edef\bbl@line{\bbl@line\space\space\space}%
4826 \expandafter\bbl@process@line\bbl@line\relax
4827 \fi
4828 \repeat
4829 \fi
4830 \endgroup
4831 \bbl@trace{Macros for reading patterns files}
4832 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4833 \ifx\babelcatcodetablenum\undefined
4834 \ifx\newcatcodetable\undefined
4835 \def\babelcatcodetablenum{5211}
4836 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4837 \else
4838 \newcatcodetable\babelcatcodetablenum
4839 \newcatcodetable\bbl@pattcodes
4840 \fi
4841 \else
4842 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4843 \fi
4844 \def\bbl@luapatterns#1#2{%
4845 \bbl@get@enc#1::\@@
4846 \setbox\z@\hbox\bgroup
4847 \begingroup
4848 \savecatcodetable\babelcatcodetablenum\relax
4849 \initcatcodetable\bbl@pattcodes\relax
4850 \catcodetable\bbl@pattcodes\relax
4851 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4852 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4853 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4854 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4855 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4856 \catcode`\'=12 \catcode`\`=12 \catcode`\`=12
4857 \input #1\relax
4858 \catcodetable\babelcatcodetablenum\relax
4859 \endgroup
4860 \def\bbl@tempa{#2}%
4861 \ifx\bbl@tempa\empty\else
4862 \input #2\relax
4863 \fi

```

```

4864 \egroup}%
4865 \def\bbl@patterns@lua#1{%
4866 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4867 \csname l@#1\endcsname
4868 \edef\bbl@tempa{#1}%
4869 \else
4870 \csname l@#1:\f@encoding\endcsname
4871 \edef\bbl@tempa{#1:\f@encoding}%
4872 \fi\relax
4873 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4874 \@ifundefined{bbl@hyphendata@the\language}%
4875 {\def\bbl@elt##1##2##3##4{%
4876 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4877 \def\bbl@tempb{##3}%
4878 \ifx\bbl@tempb@empty\else % if not a synonymous
4879 \def\bbl@tempc{##3}{##4}%
4880 \fi
4881 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4882 \fi}%
4883 \bbl@languages
4884 \@ifundefined{bbl@hyphendata@the\language}%
4885 {\bbl@info{No hyphenation patterns were set for\%
4886 language '\bbl@tempa'. Reported}}%
4887 {\expandafter\expandafter\expandafter\bbl@luapatterns
4888 \csname bbl@hyphendata@the\language\endcsname}}}%
4889 \endinput\fi
4890 % Here ends \ifx\AddBabelHook\@undefined
4891 % A few lines are only read by hyphen.cfg
4892 \ifx\DisableBabelHook\@undefined
4893 \AddBabelHook{luatex}{everylanguage}{%
4894 \def\process@language##1##2##3{%
4895 \def\process@line####1####2 ####3 ####4 {}}%
4896 \AddBabelHook{luatex}{loadpatterns}{%
4897 \input #1\relax
4898 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4899 {{#1}}}%
4900 \AddBabelHook{luatex}{loadexceptions}{%
4901 \input #1\relax
4902 \def\bbl@tempb##1##2{{#1}{#1}}%
4903 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4904 {\expandafter\expandafter\expandafter\bbl@tempb
4905 \csname bbl@hyphendata@the\language\endcsname}}
4906 \endinput\fi
4907 % Here stops reading code for hyphen.cfg
4908 % The following is read the 2nd time it's loaded
4909 \begingroup % TODO - to a lua file
4910 \catcode`\%=12
4911 \catcode`\'=12
4912 \catcode`\%=12
4913 \catcode`\:=12
4914 \directlua{
4915 Babel = Babel or {}
4916 function Babel.bytes(line)
4917 return line:gsub(".",
4918 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4919 end
4920 function Babel.begin_process_input()
4921 if luatexbase and luatexbase.add_to_callback then
4922 luatexbase.add_to_callback('process_input_buffer',

```

```

4923                                     Babel.bytes,'Babel.bytes')
4924     else
4925         Babel.callback = callback.find('process_input_buffer')
4926         callback.register('process_input_buffer',Babel.bytes)
4927     end
4928 end
4929 function Babel.end_process_input ()
4930     if luatexbase and luatexbase.remove_from_callback then
4931         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4932     else
4933         callback.register('process_input_buffer',Babel.callback)
4934     end
4935 end
4936 function Babel.addpatterns(pp, lg)
4937     local lg = lang.new(lg)
4938     local pats = lang.patterns(lg) or ''
4939     lang.clear_patterns(lg)
4940     for p in pp:gmatch('[^%s]+') do
4941         ss = ''
4942         for i in string.utfcharacters(p:gsub('%d', '')) do
4943             ss = ss .. '%d?' .. i
4944         end
4945         ss = ss:gsub('^%%d%?%.','%%.') .. '%d?'
4946         ss = ss:gsub('%.%%d%?$','%%.')
4947         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4948         if n == 0 then
4949             tex.sprint(
4950                 [[\string\csname\space bbl@info\endcsname{New pattern: }
4951                 .. p .. [{}]])
4952             pats = pats .. ' ' .. p
4953         else
4954             tex.sprint(
4955                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }
4956                 .. p .. [{}]])
4957         end
4958     end
4959     lang.patterns(lg, pats)
4960 end
4961 }
4962 \endgroup
4963 \ifx\newattribute\@undefined\else
4964     \newattribute\bbl@attr@locale
4965     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4966     \AddBabelHook{luatex}{beforeextras}{%
4967         \setattribute\bbl@attr@locale\localeid}
4968 \fi
4969 \def\BabelStringsDefault{unicode}
4970 \let\luabbl@stop\relax
4971 \AddBabelHook{luatex}{encodedcommands}{%
4972     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4973     \ifx\bbl@tempa\bbl@tempb\else
4974         \directlua{Babel.begin_process_input()}%
4975         \def\luabbl@stop{%
4976             \directlua{Babel.end_process_input()}}%
4977     \fi}%
4978 \AddBabelHook{luatex}{stopcommands}{%
4979     \luabbl@stop
4980     \let\luabbl@stop\relax}
4981 \AddBabelHook{luatex}{patterns}{%

```

```

4982 \@ifundefined{bbl@hyphendata@the\language}%
4983 {\def\bbl@elt##1##2##3##4{%
4984     \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4985     \def\bbl@tempb{##3}%
4986     \ifx\bbl@tempb\@empty\else % if not a synonymous
4987     \def\bbl@tempc{##3}{##4}}%
4988     \fi
4989     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4990     \fi}%
4991 \bbl@languages
4992 \@ifundefined{bbl@hyphendata@the\language}%
4993 {\bbl@info{No hyphenation patterns were set for\%
4994     language '#2'. Reported}}%
4995 {\expandafter\expandafter\expandafter\bbl@luapatterns
4996     \csname bbl@hyphendata@the\language\endcsname}}}%
4997 \@ifundefined{bbl@patterns@}{}%
4998 \begingroup
4999 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5000 \ifin\else
5001 \ifx\bbl@patterns@\@empty\else
5002 \directlua{ Babel.addpatterns(
5003     [[\bbl@patterns@]], \number\language) }%
5004 \fi
5005 \@ifundefined{bbl@patterns@#1}%
5006 \@empty
5007 {\directlua{ Babel.addpatterns(
5008     [[\space\csname bbl@patterns@#1\endcsname]],
5009     \number\language) }}%
5010 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5011 \fi
5012 \endgroup}%
5013 \bbl@exp{%
5014 \bbl@ifunset{bbl@prehc\language\name}{}%
5015 {\bbl@ifblank{\bbl@cs{prehc\language\name}}}%
5016 {\prehyphenchar=\bbl@c1{prehc}\relax}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

5017 \@onlypreamble\babelpatterns
5018 \AtEndOfPackage{%
5019 \newcommand\babelpatterns[2][\@empty]{%
5020 \ifx\bbl@patterns@\relax
5021 \let\bbl@patterns@\@empty
5022 \fi
5023 \ifx\bbl@pttnlist\@empty\else
5024 \bbl@warning{%
5025     You must not intermingle \string\selectlanguage\space and\%
5026     \string\babelpatterns\space or some patterns will not\%
5027     be taken into account. Reported}%
5028 \fi
5029 \ifx\@empty#1%
5030 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5031 \else
5032 \edef\bbl@tempb{\zap@space#1 \@empty}%
5033 \bbl@for\bbl@tempa\bbl@tempb{%
5034 \bbl@fixname\bbl@tempa
5035 \bbl@iflanguage\bbl@tempa{%
5036 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%

```

```

5037         \@ifundefined{bbl@patterns@bbl@tempa}%
5038         \@empty
5039         {\csname bbl@patterns@bbl@tempa\endcsname\space}%
5040         #2}}}%
5041     \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```

5042 % TODO - to a lua file
5043 \directlua{
5044   Babel = Babel or {}
5045   Babel.linebreaking = Babel.linebreaking or {}
5046   Babel.linebreaking.before = {}
5047   Babel.linebreaking.after = {}
5048   Babel.locale = {} % Free to use, indexed with \localeid
5049   function Babel.linebreaking.add_before(func)
5050     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5051     table.insert(Babel.linebreaking.before, func)
5052   end
5053   function Babel.linebreaking.add_after(func)
5054     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5055     table.insert(Babel.linebreaking.after, func)
5056   end
5057 }
5058 \def\bbl@intraspace#1 #2 #3\@@{%
5059   \directlua{
5060     Babel = Babel or {}
5061     Babel.intraspaces = Babel.intraspaces or {}
5062     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5063       {b = #1, p = #2, m = #3}
5064     Babel.locale_props[\the\localeid].intraspace = %
5065       {b = #1, p = #2, m = #3}
5066   }}
5067 \def\bbl@intrapenalty#1\@@{%
5068   \directlua{
5069     Babel = Babel or {}
5070     Babel.intrapenalties = Babel.intrapenalties or {}
5071     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5072     Babel.locale_props[\the\localeid].intrapenalty = #1
5073   }}
5074 \begingroup
5075 \catcode`\%=12
5076 \catcode`\^=14
5077 \catcode`\'=12
5078 \catcode`\~=12
5079 \gdef\bbl@seaintraspace{^
5080   \let\bbl@seaintraspace\relax
5081   \directlua{
5082     Babel = Babel or {}
5083     Babel.sea_enabled = true
5084     Babel.sea_ranges = Babel.sea_ranges or {}
5085     function Babel.set_chranges (script, chrng)
5086       local c = 0
5087       for s, e in string.gmatch(chrng..' ', '(-)%.%.(.)%s') do

```

```

5088     Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5089     c = c + 1
5090 end
5091 end
5092 function Babel.sea_disc_to_space (head)
5093     local sea_ranges = Babel.sea_ranges
5094     local last_char = nil
5095     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5096     for item in node.traverse(head) do
5097         local i = item.id
5098         if i == node.id'glyph' then
5099             last_char = item
5100         elseif i == 7 and item.subtype == 3 and last_char
5101             and last_char.char > 0x0C99 then
5102             quad = font.getfont(last_char.font).size
5103             for lg, rg in pairs(sea_ranges) do
5104                 if last_char.char > rg[1] and last_char.char < rg[2] then
5105                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5106                     local intraspace = Babel.intraspaces[lg]
5107                     local intrapenalty = Babel.intrapenalties[lg]
5108                     local n
5109                     if intrapenalty ~= 0 then
5110                         n = node.new(14, 0)      ^% penalty
5111                         n.penalty = intrapenalty
5112                         node.insert_before(head, item, n)
5113                     end
5114                     n = node.new(12, 13)      ^% (glue, spaceskip)
5115                     node.setglue(n, intraspace.b * quad,
5116                                 intraspace.p * quad,
5117                                 intraspace.m * quad)
5118                     node.insert_before(head, item, n)
5119                     node.remove(head, item)
5120                 end
5121             end
5122         end
5123     end
5124 end
5125 }^^
5126 \bbl@luahyphenate}
5127 \catcode`\%=14
5128 \gdef\bbl@cjkintraspaces{%
5129 \let\bbl@cjkintraspaces\relax
5130 \directlua{
5131     Babel = Babel or {}
5132     require('babel-data-cjk.lua')
5133     Babel.cjk_enabled = true
5134     function Babel.cjk_linebreak(head)
5135         local GLYPH = node.id'glyph'
5136         local last_char = nil
5137         local quad = 655360      % 10 pt = 655360 = 10 * 65536
5138         local last_class = nil
5139         local last_lang = nil
5140
5141         for item in node.traverse(head) do
5142             if item.id == GLYPH then
5143
5144                 local lang = item.lang
5145
5146                 local LOCALE = node.get_attribute(item,

```



```

5147         luatexbase.registernumber'bbl@attr@locale')
5148     local props = Babel.locale_props[LOCALE]
5149
5150     local class = Babel.cjk_class[item.char].c
5151
5152     if class == 'cp' then class = 'cl' end % ]) as CL
5153     if class == 'id' then class = 'I' end
5154
5155     local br = 0
5156     if class and last_class and Babel.cjk_breaks[last_class][class] then
5157         br = Babel.cjk_breaks[last_class][class]
5158     end
5159
5160     if br == 1 and props.linebreak == 'c' and
5161         lang ~= \the\l@nohyphenation\space and
5162         last_lang ~= \the\l@nohyphenation then
5163         local intrapenalty = props.intrapenalty
5164         if intrapenalty ~= 0 then
5165             local n = node.new(14, 0)    % penalty
5166             n.penalty = intrapenalty
5167             node.insert_before(head, item, n)
5168         end
5169         local intraspace = props.intraspace
5170         local n = node.new(12, 13)      % (glue, spaceskip)
5171         node.setglue(n, intraspace.b * quad,
5172             intraspace.p * quad,
5173             intraspace.m * quad)
5174         node.insert_before(head, item, n)
5175     end
5176
5177     if font.getfont(item.font) then
5178         quad = font.getfont(item.font).size
5179     end
5180     last_class = class
5181     last_lang = lang
5182     else % if penalty, glue or anything else
5183         last_class = nil
5184     end
5185 end
5186 lang.hyphenate(head)
5187 end
5188 }%
5189 \bbl@luahyphenate}
5190 \gdef\bbl@luahyphenate{%
5191 \let\bbl@luahyphenate\relax
5192 \directlua{
5193     luatexbase.add_to_callback('hyphenate',
5194     function (head, tail)
5195         if Babel.linebreaking.before then
5196             for k, func in ipairs(Babel.linebreaking.before) do
5197                 func(head)
5198             end
5199         end
5200         if Babel.cjk_enabled then
5201             Babel.cjk_linebreak(head)
5202         end
5203         lang.hyphenate(head)
5204         if Babel.linebreaking.after then
5205             for k, func in ipairs(Babel.linebreaking.after) do

```

```

5206         func(head)
5207     end
5208 end
5209 if Babel.sea_enabled then
5210     Babel.sea_disc_to_space(head)
5211 end
5212 end,
5213 'Babel.hyphenate')
5214 }
5215 }
5216 \endgroup
5217 \def\bbl@provide@intraspace{%
5218 \bbl@ifunset{\bbl@intsp@{language}}{%
5219 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5220 \bbl@xin@{\bbl@cl{lbrk}}{c}%
5221 \ifin@ % cjk
5222 \bbl@cjk@intraspace
5223 \directlua{
5224     Babel = Babel or {}
5225     Babel.locale_props = Babel.locale_props or {}
5226     Babel.locale_props[\the\localeid].linebreak = 'c'
5227 }%
5228 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5229 \ifx\bbl@KVP@intrapenalty\@nil
5230 \bbl@intrapenalty0\@@
5231 \fi
5232 \else % sea
5233 \bbl@sea@intraspace
5234 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5235 \directlua{
5236     Babel = Babel or {}
5237     Babel.sea_ranges = Babel.sea_ranges or {}
5238     Babel.set_chranges('\bbl@cl{sbc}',
5239                       '\bbl@cl{chrng}')
5240 }%
5241 \ifx\bbl@KVP@intrapenalty\@nil
5242 \bbl@intrapenalty0\@@
5243 \fi
5244 \fi
5245 \fi
5246 \ifx\bbl@KVP@intrapenalty\@nil\else
5247 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5248 \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.

```

5249 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5250 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5251 \DisableBabelHook{babel-fontspec}
5252 <<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 form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5253% TODO - to a lua file
5254 \directlua{
5255 Babel.script_blocks = {
5256   ['dflt'] = {},
5257   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5258             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5259   ['Armn'] = {{0x0530, 0x058F}},
5260   ['Beng'] = {{0x0980, 0x09FF}},
5261   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5262   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5263   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5264             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5265   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5266   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5267             {0xAB00, 0xAB2F}},
5268   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5269   % Don't follow strictly Unicode, which places some Coptic letters in
5270   % the 'Greek and Coptic' block
5271   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5272   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5273             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5274             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5275             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5276             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5277             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5278   ['Hebr'] = {{0x0590, 0x05FF}},
5279   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5280             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5281   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5282   ['Knda'] = {{0x0C80, 0x0CFF}},
5283   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5284             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5285             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5286   ['Laoo'] = {{0x0E80, 0x0EFF}},
5287   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5288             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5289             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5290   ['Mahj'] = {{0x11150, 0x1117F}},
5291   ['Mlym'] = {{0x0D00, 0x0D7F}},
5292   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5293   ['Orya'] = {{0x0B00, 0x0B7F}},
5294   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5295   ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5296   ['Taml'] = {{0x0B80, 0x0BFF}},
5297   ['Telu'] = {{0x0C00, 0x0C7F}},
5298   ['Tfng'] = {{0x2D30, 0x2D7F}},
5299   ['Thai'] = {{0x0E00, 0x0E7F}},
5300   ['Tibt'] = {{0x0F00, 0x0FFF}},
5301   ['Vaii'] = {{0xA500, 0xA63F}},
5302   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
```

```

5303 }
5304
5305 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5306 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5307 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5308
5309 function Babel.locale_map(head)
5310   if not Babel.locale_mapped then return head end
5311
5312   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5313   local GLYPH = node.id('glyph')
5314   local inmath = false
5315   local toloc_save
5316   for item in node.traverse(head) do
5317     local toloc
5318     if not inmath and item.id == GLYPH then
5319       % Optimization: build a table with the chars found
5320       if Babel.chr_to_loc[item.char] then
5321         toloc = Babel.chr_to_loc[item.char]
5322       else
5323         for lc, maps in pairs(Babel.loc_to_scr) do
5324           for _, rg in pairs(maps) do
5325             if item.char >= rg[1] and item.char <= rg[2] then
5326               Babel.chr_to_loc[item.char] = lc
5327               toloc = lc
5328               break
5329             end
5330           end
5331         end
5332       end
5333       % Now, take action, but treat composite chars in a different
5334       % fashion, because they 'inherit' the previous locale. Not yet
5335       % optimized.
5336       if not toloc and
5337         (item.char >= 0x0300 and item.char <= 0x036F) or
5338         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5339         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5340         toloc = toloc_save
5341       end
5342       if toloc and toloc > -1 then
5343         if Babel.locale_props[toloc].lg then
5344           item.lang = Babel.locale_props[toloc].lg
5345           node.set_attribute(item, LOCALE, toloc)
5346         end
5347         if Babel.locale_props[toloc]['/'..item.font] then
5348           item.font = Babel.locale_props[toloc]['/'..item.font]
5349         end
5350         toloc_save = toloc
5351       end
5352     elseif not inmath and item.id == 7 then
5353       item.replace = item.replace and Babel.locale_map(item.replace)
5354       item.pre = item.pre and Babel.locale_map(item.pre)
5355       item.post = item.post and Babel.locale_map(item.post)
5356     elseif item.id == node.id'math' then
5357       inmath = (item.subtype == 0)
5358     end
5359   end
5360   return head
5361 end

```

5362 }

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

```

5363 \newcommand\babelcharproperty[1]{%
5364   \count@=#1\relax
5365   \ifvmode
5366     \expandafter\bb1@chprop
5367   \else
5368     \bb1@error{\string\babelcharproperty\space can be used only in\%
5369               vertical mode (preamble or between paragraphs)}%
5370     {See the manual for futher info}%
5371   \fi}
5372 \newcommand\bb1@chprop[3][\the\count@]{%
5373   \@tempcnta=#1\relax
5374   \bb1@ifunset{\bb1@chprop@#2}%
5375   {\bb1@error{No property named '#2'. Allowed values are\%
5376             direction (bc), mirror (bmg), and linebreak (lb)}%
5377   {See the manual for futher info}}%
5378   {}%
5379   \loop
5380     \bb1@cs{chprop@#2}{#3}%
5381     \ifnum\count@<\@tempcnta
5382       \advance\count@\@ne
5383     \repeat}
5384 \def\bb1@chprop@direction#1{%
5385   \directlua{
5386     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5387     Babel.characters[\the\count@]['d'] = '#1'
5388   }}
5389 \let\bb1@chprop@bc\bb1@chprop@direction
5390 \def\bb1@chprop@mirror#1{%
5391   \directlua{
5392     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5393     Babel.characters[\the\count@]['m'] = '\number#1'
5394   }}
5395 \let\bb1@chprop@bmg\bb1@chprop@mirror
5396 \def\bb1@chprop@linebreak#1{%
5397   \directlua{
5398     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5399     Babel.cjk_characters[\the\count@]['c'] = '#1'
5400   }}
5401 \let\bb1@chprop@lb\bb1@chprop@linebreak
5402 \def\bb1@chprop@locale#1{%
5403   \directlua{
5404     Babel.chr_to_loc = Babel.chr_to_loc or {}
5405     Babel.chr_to_loc[\the\count@] =
5406       \bb1@ifblank{#1}{-1000}{\the\bb1@cs{id@#1}}\space
5407   }}

```

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 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.

```

5408 \begingroup % TODO - to a lua file
5409 \catcode`\~ = 12
5410 \catcode`\# = 12
5411 \catcode`\% = 12
5412 \catcode`\& = 14
5413 \directlua{
5414   Babel.linebreaking.replacements = {}
5415   Babel.linebreaking.replacements[0] = {} &% pre
5416   Babel.linebreaking.replacements[1] = {} &% post
5417
5418   &% Discretionaries contain strings as nodes
5419   function Babel.str_to_nodes(fn, matches, base)
5420     local n, head, last
5421     if fn == nil then return nil end
5422     for s in string.utfvalues(fn(matches)) do
5423       if base.id == 7 then
5424         base = base.replace
5425       end
5426       n = node.copy(base)
5427       n.char = s
5428       if not head then
5429         head = n
5430       else
5431         last.next = n
5432       end
5433       last = n
5434     end
5435     return head
5436   end
5437
5438   Babel.fetch_subtext = {}
5439
5440   &% Merging both functions doesn't seem feasible, because there are too
5441   &% many differences.
5442   Babel.fetch_subtext[0] = function(head)
5443     local word_string = ''
5444     local word_nodes = {}
5445     local lang
5446     local item = head
5447     local inmath = false
5448
5449     while item do
5450
5451       if item.id == 11 then
5452         inmath = (item.subtype == 0)
5453       end
5454
5455       if inmath then
5456         &% pass
5457       elseif item.id == 29 then
5458         local locale = node.get_attribute(item, Babel.attr_locale)
5459
5460         if lang == locale or lang == nil then
5461           if (item.char ~= 124) then &% ie, not | = space

```

```

5463         lang = lang or locale
5464         word_string = word_string .. unicode.utf8.char(item.char)
5465         word_nodes[#word_nodes+1] = item
5466     end
5467     else
5468         break
5469     end
5470
5471     elseif item.id == 12 and item.subtype == 13 then
5472         word_string = word_string .. '|'
5473         word_nodes[#word_nodes+1] = item
5474
5475         %% Ignore leading unrecognized nodes, too.
5476         elseif word_string ~= '' then
5477             word_string = word_string .. Babel.us_char
5478             word_nodes[#word_nodes+1] = item %% Will be ignored
5479         end
5480
5481         item = item.next
5482     end
5483
5484     %% Here and above we remove some trailing chars but not the
5485     %% corresponding nodes. But they aren't accessed.
5486     if word_string:sub(-1) == '|' then
5487         word_string = word_string:sub(1,-2)
5488     end
5489     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5490     return word_string, word_nodes, item, lang
5491 end
5492
5493 Babel.fetch_subtext[1] = function(head)
5494     local word_string = ''
5495     local word_nodes = {}
5496     local lang
5497     local item = head
5498     local inmath = false
5499
5500     while item do
5501
5502         if item.id == 11 then
5503             inmath = (item.subtype == 0)
5504         end
5505
5506         if inmath then
5507             %% pass
5508         end
5509
5510         elseif item.id == 29 then
5511             if item.lang == lang or lang == nil then
5512                 if (item.char ~= 124) and (item.char ~= 61) then %% not =, not |
5513                     lang = lang or item.lang
5514                     word_string = word_string .. unicode.utf8.char(item.char)
5515                     word_nodes[#word_nodes+1] = item
5516                 end
5517             else
5518                 break
5519             end
5520
5521             elseif item.id == 7 and item.subtype == 2 then
5522                 word_string = word_string .. '='

```

```

5522         word_nodes[#word_nodes+1] = item
5523
5524     elseif item.id == 7 and item.subtype == 3 then
5525         word_string = word_string .. '|'
5526         word_nodes[#word_nodes+1] = item
5527
5528         &% (1) Go to next word if nothing was found, and (2) implicitly
5529         &% remove leading USs.
5530     elseif word_string == '' then
5531         &% pass
5532
5533         &% This is the responsible for splitting by words.
5534     elseif (item.id == 12 and item.subtype == 13) then
5535         break
5536
5537     else
5538         word_string = word_string .. Babel.us_char
5539         word_nodes[#word_nodes+1] = item &% Will be ignored
5540     end
5541
5542     item = item.next
5543 end
5544
5545 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5546 return word_string, word_nodes, item, lang
5547 end
5548
5549 function Babel.pre_hyphenate_replace(head)
5550     Babel.hyphenate_replace(head, 0)
5551 end
5552
5553 function Babel.post_hyphenate_replace(head)
5554     Babel.hyphenate_replace(head, 1)
5555 end
5556
5557 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5558     local ss = ''
5559     for pp = 1, 40 do
5560         if wn[pp] then
5561             if wn[pp].id == 29 then
5562                 ss = ss .. unicode.utf8.char(wn[pp].char)
5563             else
5564                 ss = ss .. '{' .. wn[pp].id .. '}'
5565             end
5566         end
5567     end
5568     print('nod', ss)
5569     print('lst_m',
5570         string.rep(' ', unicode.utf8.len(
5571             string.sub(w, 1, last_match))-1) .. '>')
5572     print('str', w)
5573     print('sc', string.rep(' ', sc-1) .. '^')
5574     if first == last then
5575         print('f=l', string.rep(' ', first-1) .. '!!')
5576     else
5577         print('f/l', string.rep(' ', first-1) .. '[' ..
5578             string.rep(' ', last-first-1) .. ']')
5579     end
5580 end

```



```

5581
5582 Babel.us_char = string.char(31)
5583
5584 function Babel.hyphenate_replace(head, mode)
5585     local u = unicode.utf8
5586     local lbkr = Babel.linebreaking.replacements[mode]
5587
5588     local word_head = head
5589
5590     while true do    %% for each subtext block
5591
5592         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5593
5594         if Babel.debug then
5595             print()
5596             print((mode == 0) and '@@@<' or '@@@>', w)
5597         end
5598
5599         if nw == nil and w == '' then break end
5600
5601         if not lang then goto next end
5602         if not lbkr[lang] then goto next end
5603
5604         %% For each saved (pre|post)hyphenation. TODO. Reconsider how
5605         %% loops are nested.
5606         for k=1, #lbkr[lang] do
5607             local p = lbkr[lang][k].pattern
5608             local r = lbkr[lang][k].replace
5609
5610             if Babel.debug then
5611                 print('*****', p, mode)
5612             end
5613
5614             %% This variable is set in some cases below to the first *byte*
5615             %% after the match, either as found by u.match (faster) or the
5616             %% computed position based on sc if w has changed.
5617             local last_match = 0
5618
5619             %% For every match.
5620             while true do
5621                 if Babel.debug then
5622                     print('====')
5623                 end
5624                 local new    %% used when inserting and removing nodes
5625                 local refetch = false
5626
5627                 local matches = { u.match(w, p, last_match) }
5628                 if #matches < 2 then break end
5629
5630                 %% Get and remove empty captures (with ()'s, which return a
5631                 %% number with the position), and keep actual captures
5632                 %% (from (...)), if any, in matches.
5633                 local first = table.remove(matches, 1)
5634                 local last  = table.remove(matches, #matches)
5635                 %% Non re-fetched substrings may contain \31, which separates
5636                 %% subsubstrings.
5637                 if string.find(w:sub(first, last-1), Babel.us_char) then break end
5638
5639                 local save_last = last    %% with A()BC()D, points to D

```

```

5640
5641      %% Fix offsets, from bytes to unicode. Explained above.
5642      first = u.len(w:sub(1, first-1)) + 1
5643      last  = u.len(w:sub(1, last-1)) %% now last points to C
5644
5645      %% This loop stores in n small table the nodes
5646      %% corresponding to the pattern. Used by 'data' to provide a
5647      %% predictable behavior with 'insert' (now w_nodes is modified on
5648      %% the fly), and also access to 'remove'd nodes.
5649      local sc = first-1          %% Used below, too
5650      local data_nodes = {}
5651
5652      for q = 1, last-first+1 do
5653          data_nodes[q] = w_nodes[sc+q]
5654      end
5655
5656      %% This loop traverses the matched substring and takes the
5657      %% corresponding action stored in the replacement list.
5658      %% sc = the position in substr nodes / string
5659      %% rc = the replacement table index
5660      local rc = 0
5661
5662      while rc < last-first+1 do %% for each replacement
5663          if Babel.debug then
5664              print('.....', rc + 1)
5665          end
5666          sc = sc + 1
5667          rc = rc + 1
5668
5669          if Babel.debug then
5670              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5671          end
5672
5673          local crep = r[rc]
5674          local item = w_nodes[sc]
5675          local item_base = item
5676          local placeholder = Babel.us_char
5677          local d
5678
5679          if crep and crep.data then
5680              item_base = data_nodes[crep.data]
5681          end
5682
5683          if crep and next(crep) == nil then %% = {}
5684              last_match = save_last      %% Optimization
5685              goto next
5686
5687          elseif crep == nil then %% = remove
5688              node.remove(head, item)
5689              table.remove(w_nodes, sc)
5690              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5691              sc = sc - 1 %% Nothing has been inserted.
5692              last_match = utf8.offset(w, sc+1)
5693              goto next
5694
5695          elseif crep and crep.string then
5696              local str = crep.string(matches)
5697              if str == '' then %% Gather with nil
5698                  node.remove(head, item)

```

```

5699         table.remove(w_nodes, sc)
5700         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5701         sc = sc - 1  && Nothing has been inserted.
5702     else
5703         local loop_first = true
5704         for s in string.utfvalues(str) do
5705             d = node.copy(item_base)
5706             d.char = s
5707             if loop_first then
5708                 loop_first = false
5709                 head, new = node.insert_before(head, item, d)
5710                 if sc == 1 then
5711                     word_head = head
5712                 end
5713                 w_nodes[sc] = d
5714                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
5715             else
5716                 sc = sc + 1
5717                 head, new = node.insert_before(head, item, d)
5718                 table.insert(w_nodes, sc, new)
5719                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
5720             end
5721             if Babel.debug then
5722                 print('.....', 'str')
5723                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5724             end
5725             end  && for
5726             node.remove(head, item)
5727         end  && if ''
5728         last_match = utf8.offset(w, sc+1)
5729         goto next
5730
5731     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
5732         d = node.new(7, 0)  && (disc, discretionary)
5733         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
5734         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
5735         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
5736         d.attr = item_base.attr
5737         if crep.pre == nil then  && TeXbook p96
5738             d.penalty = crep.penalty or tex.hyphenpenalty
5739         else
5740             d.penalty = crep.penalty or tex.exhyphenpenalty
5741         end
5742         placeholder = '|'
5743         head, new = node.insert_before(head, item, d)
5744
5745     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
5746         && ERROR
5747
5748     elseif crep and crep.penalty then
5749         d = node.new(14, 0)  && (penalty, userpenalty)
5750         d.attr = item_base.attr
5751         d.penalty = crep.penalty
5752         head, new = node.insert_before(head, item, d)
5753
5754     elseif crep and crep.space then
5755         && 655360 = 10 pt = 10 * 65536 sp
5756         d = node.new(12, 13)  && (glue, spaceskip)
5757         local quad = font.getfont(item_base.font).size or 655360

```

```

5758         node.setglue(d, crep.space[1] * quad,
5759                       crep.space[2] * quad,
5760                       crep.space[3] * quad)
5761         if mode == 0 then
5762             placeholder = '|'
5763         end
5764         head, new = node.insert_before(head, item, d)
5765
5766     elseif mode == 0 and crep and crep.space then
5767         %% ERROR
5768
5769     end    %% ie replacement cases
5770
5771     %% Shared by disc, space and penalty.
5772     if sc == 1 then
5773         word_head = head
5774     end
5775     if crep.insert then
5776         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
5777         table.insert(w_nodes, sc, new)
5778         last = last + 1
5779     else
5780         w_nodes[sc] = d
5781         node.remove(head, item)
5782         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
5783     end
5784
5785     last_match = utf8.offset(w, sc+1)
5786
5787     ::next::
5788
5789     end    %% for each replacement
5790
5791     if Babel.debug then
5792         print('.....', '/')
5793         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5794     end
5795
5796     end    %% for match
5797
5798     end    %% for patterns
5799
5800     ::next::
5801     word_head = nw
5802     end    %% for substring
5803     return head
5804 end
5805
5806 %% This table stores capture maps, numbered consecutively
5807 Babel.capture_maps = {}
5808
5809 %% The following functions belong to the next macro
5810 function Babel.capture_func(key, cap)
5811     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..["] .. "]"
5812     ret = ret:gsub('{{[0-9]}|([^\]|+)|(.-)}', Babel.capture_func_map)
5813     ret = ret:gsub("%[%[%]]%.", '')
5814     ret = ret:gsub("%.%[%[%]]%", '')
5815     return key .. [[=function(m) return ]] .. ret .. [[ end]]
5816 end

```

```

5817
5818 function Babel.capt_map(from, mapno)
5819     return Babel.capture_maps[mapno][from] or from
5820 end
5821
5822 &% Handle the {n|abc|ABC} syntax in captures
5823 function Babel.capture_func_map(capno, from, to)
5824     local froms = {}
5825     for s in string.utfcharacters(from) do
5826         table.insert(froms, s)
5827     end
5828     local cnt = 1
5829     table.insert(Babel.capture_maps, {})
5830     local mlen = table.getn(Babel.capture_maps)
5831     for s in string.utfcharacters(to) do
5832         Babel.capture_maps[mlen][froms[cnt]] = s
5833         cnt = cnt + 1
5834     end
5835     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5836         (mlen) .. ").. " .. "["
5837 end
5838 }

```

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 $\text{function}(m) \text{ return } m[1]..m[1]..'-' \text{ end}$, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m) \text{ return } \text{Babel.capt_map}(m[1],1) \text{ end}$, 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 $\backslash\text{directlua}$ does not take into account the current catcode of $@$, we just avoid this character in macro names (which explains the internal group, too).

```

5839 \catcode`\#=6
5840 \gdef\babelposthyphenation#1#2#3{&%
5841     \bbl@activateposthyphen
5842     \begingroup
5843         \def\babeltempa{\bbl@add@list\babeltempb}&%
5844         \let\babeltempb\@empty
5845         \bbl@foreach{#3}{&%
5846             \bbl@ifsamestring{##1}{remove}&%
5847             {\bbl@add@list\babeltempb{nil}}&%
5848             {\directlua{
5849                 local rep = [[#1]]
5850                 rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5851                 rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5852                 rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5853                 rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5854                 rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5855                 tex.print([[\\string\babeltempa{}}] .. rep .. [[}}]])
5856             }}&%
5857         \directlua{
5858             local lbkr = Babel.linebreaking.replacements[1]
5859             local u = unicode.utf8
5860             &% Convert pattern:
5861             local patt = string.gsub([=[#2]=], '%s', '')
5862             if not u.find(patt, '()', nil, true) then
5863                 patt = '()' .. patt .. '()'
5864             end
5865             patt = string.gsub(patt, '%(%)%^', '^()')

```

```

5866     patt = string.gsub(patt, '%$(%)', '()$')
5867     patt = u.gsub(patt, '{(.)}',
5868         function (n)
5869             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5870         end)
5871     lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5872     table.insert(lbkr[\the\csname l@#1\endcsname],
5873         { pattern = patt, replace = { \babeltempb } })
5874 }&%
5875 \endgroup}
5876 % TODO. Copypaste pattern.
5877 \gdef\babelprehyphenation#1#2#3{&%
5878     \bbl@activateprehyphen
5879     \begingroup
5880     \def\babeltempa{\bbl@add@list\babeltempb}&%
5881     \let\babeltempb\@empty
5882     \bbl@foreach{#3}{&%
5883         \bbl@ifsamestring{##1}{remove}&%
5884         {\bbl@add@list\babeltempb{nil}}&%
5885         {\directlua{
5886             local rep = [[#1]]
5887             rep = rep.gsub('^s*(insert)%s*', 'insert = true, ')
5888             rep = rep.gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5889             rep = rep.gsub(' (space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5890                 'space = {' .. '%2, %3, %4' .. '}')
5891             tex.print([[string\babeltempa{}}] .. rep .. [[}}]])
5892         }}&%
5893     \directlua{
5894         local lbkr = Babel.linebreaking.replacements[0]
5895         local u = unicode.utf8
5896         &% Convert pattern:
5897         local patt = string.gsub([=[#2]=], '%s', '')
5898         if not u.find(patt, '()', nil, true) then
5899             patt = '()' .. patt .. '()'
5900         end
5901         &% patt = string.gsub(patt, '%(%)%', '^()')
5902         &% patt = string.gsub(patt, '([^\%])%$(%)', '%1()$')
5903         patt = u.gsub(patt, '{(.)}',
5904             function (n)
5905                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5906             end)
5907         lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5908         table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5909             { pattern = patt, replace = { \babeltempb } })
5910     }&%
5911 \endgroup}
5912 \endgroup
5913 \def\bbl@activateposthyphen{%
5914     \let\bbl@activateposthyphen\relax
5915     \directlua{
5916         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5917     }}
5918 \def\bbl@activateprehyphen{%
5919     \let\bbl@activateprehyphen\relax
5920     \directlua{
5921         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5922     }}

```

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.

```
5923 \bbl@trace{Redefinitions for bidi layout}
5924 \ifx\@eqnnum\undefined\else
5925   \ifx\bbl@attr@dir\undefined\else
5926     \edef\@eqnnum{%
5927       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5928       \unexpanded\expandafter{\@eqnnum}}
5929   \fi
5930 \fi
5931 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5932 \ifnum\bbl@bidimode>\z@
5933   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5934     \bbl@exp{%
5935       \mathdir\the\bodydir
5936       #1% Once entered in math, set boxes to restore values
5937       \<ifmmode>%
5938         \everyvbox{%
5939           \the\everyvbox
5940           \bodydir\the\bodydir
5941           \mathdir\the\mathdir
5942           \everyhbox{\the\everyhbox}%
5943           \everyvbox{\the\everyvbox}}%
5944         \everyhbox{%
5945           \the\everyhbox
5946           \bodydir\the\bodydir
5947           \mathdir\the\mathdir
5948           \everyhbox{\the\everyhbox}%
5949           \everyvbox{\the\everyvbox}}%
5950       \<fi>}}%
5951   \def\@hangfrom#1{%
5952     \setbox\@tempboxa\hbox{{#1}}%
5953     \hangindent\wd\@tempboxa
5954     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5955       \shapemode\@ne
5956     \fi
5957     \noindent\box\@tempboxa}
5958 \fi
5959 \IfBabelLayout{tabular}
5960   {\let\bbl@OL@@tabular\@tabular
5961     \bbl@replace\@tabular{$$}\bbl@nextfake$}%
5962   {\let\bbl@NL@@tabular\@tabular
5963     \AtBeginDocument{%
5964       \ifx\bbl@NL@@tabular\@tabular\else
5965         \bbl@replace\@tabular{$$}\bbl@nextfake$}%
5966       \let\bbl@NL@@tabular\@tabular
5967     \fi}}
```

```

5968 {}
5969 \IfBabelLayout{lists}
5970 {\let\bbl@OL@list\list
5971 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5972 \let\bbl@NL@list\list
5973 \def\bbl@listparshape#1#2#3{%
5974 \parshape #1 #2 #3 %
5975 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5976 \shapemode\tw@
5977 \fi}}
5978 {}
5979 \IfBabelLayout{graphics}
5980 {\let\bbl@pictresetdir\relax
5981 \def\bbl@pictsetdir#1{%
5982 \ifcase\bbl@thetextdir
5983 \let\bbl@pictresetdir\relax
5984 \else
5985 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5986 \or\textdir TLT
5987 \else\bodydir TLT \textdir TLT
5988 \fi
5989 % \(\text|par)dir required in pgf:
5990 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5991 \fi}%
5992 \ifx\AddToHook\undefined\else
5993 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
5994 \directlua{
5995 Babel.get_picture_dir = true
5996 Babel.picture_has_bidi = 0
5997 function Babel.picture_dir (head)
5998 if not Babel.get_picture_dir then return head end
5999 for item in node.traverse(head) do
6000 if item.id == node.id'glyph' then
6001 local itemchar = item.char
6002 % TODO. Copypaste pattern from Babel.bidi (-r)
6003 local chardata = Babel.characters[itemchar]
6004 local dir = chardata and chardata.d or nil
6005 if not dir then
6006 for nn, et in ipairs(Babel.ranges) do
6007 if itemchar < et[1] then
6008 break
6009 elseif itemchar <= et[2] then
6010 dir = et[3]
6011 break
6012 end
6013 end
6014 end
6015 if dir and (dir == 'al' or dir == 'r') then
6016 Babel.picture_has_bidi = 1
6017 end
6018 end
6019 end
6020 return head
6021 end
6022 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6023 "Babel.picture_dir")
6024 }%
6025 \AtBeginDocument{%
6026 \long\def\put(#1,#2)#3{%

```



```

6027 \killglue
6028 % Try:
6029 \ifx\bbbl@pictresetdir\relax
6030 \def\bbbl@tempc{0}%
6031 \else
6032 \directlua{
6033     Babel.get_picture_dir = true
6034     Babel.picture_has_bidi = 0
6035 }%
6036 \setbox\z@\hb@xt@\z@{%
6037     \@defaultunitsset\@tempdimc{#1}\unitlength
6038     \kern\@tempdimc
6039     #3\hss}%
6040 \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6041 \fi
6042 % Do:
6043 \@defaultunitsset\@tempdimc{#2}\unitlength
6044 \raise\@tempdimc\hb@xt@\z@{%
6045     \@defaultunitsset\@tempdimc{#1}\unitlength
6046     \kern\@tempdimc
6047     {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6048 \ignorespaces}%
6049 \MakeRobust\put}%
6050 \fi
6051 \AtBeginDocument
6052 {\ifx\tikz@atbegin@node\undefined\else
6053     \ifx\AddToHook\undefined\else % TODO. Still tentative.
6054         \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6055         \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6056     \fi
6057     \let\bbbl@OL@pgfpicture\pgfpicture
6058     \bbbl@sreplace\pgfpicture{\pgfpicturetrue}%
6059     {\bbbl@pictsetdir\z@\pgfpicturetrue}%
6060     \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6061     \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6062     \bbbl@sreplace\tikz{\beginpgroup}%
6063     {\beginpgroup\bbbl@pictsetdir\tw}%
6064 \fi
6065 \ifx\AddToHook\undefined\else
6066     \AddToHook{env/tcolorbox/begin}{\bbbl@pictsetdir\@ne}%
6067 \fi
6068 }}
6069 {}

```

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.

```

6070 \IfBabelLayout{counters}%
6071 {\let\bbbl@OL@@textsuperscript\textsuperscript
6072 \bbbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6073 \let\bbbl@latinarabic=\@arabic
6074 \let\bbbl@OL@@arabic\@arabic
6075 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
6076 \@ifpackagewith{babel}{bidi=default}%
6077 {\let\bbbl@asciroman=\@roman
6078 \let\bbbl@OL@@roman\@roman
6079 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
6080 \let\bbbl@asciiRoman=\@Roman
6081 \let\bbbl@OL@@roman\@Roman

```

```

6082 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6083 \let\bbl@OL@labelenumii\labelenumii
6084 \def\labelenumii{}\theenumii}%
6085 \let\bbl@OL@p@enumiii\p@enumiii
6086 \def\p@enumiii{\p@enumii)\theenumii{}}{}{}
6087 <<Footnote changes>>
6088 \IfBabelLayout{footnotes}%
6089 {\let\bbl@OL@footnote\footnote
6090 \BabelFootnote\footnote\language{}{}}%
6091 \BabelFootnote\localfootnote\language{}{}}%
6092 \BabelFootnote\mainfootnote{}}{}{}
6093 {}

```

```

6094 \IfBabelLayout{extras}%
6095   {\let\bbl@OL@underline\underline
6096    \bbl@sreplace\underline{$\@@underline}\{ \bbl@nextfake$\@@underline}%
6097    \let\bbl@OL@LaTeX2e\LaTeX2e
6098    \DeclareRobustCommand{\LaTeXe}\{\mbox{\m@th
6099      \if b\expandafter\@car\@f@series\@nil\boldmath\fi
6100      \babelsublr}%
6101      \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6102   {}
6103 \end{luatex}

```

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:

For the meaning of these codes, see the Unicode standard.

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!

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.

```

6104 (*basic-r)
6105 Babel = Babel or {}
6106
6107 Babel.bidi_enabled = true
6108
6109 require('babel-data-bidi.lua')
6110
6111 local characters = Babel.characters
6112 local ranges = Babel.ranges
6113
6114 local DIR = node.id("dir")
6115
6116 local function dir_mark(head, from, to, outer)
6117   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6118   local d = node.new(DIR)
6119   d.dir = '+' .. dir
6120   node.insert_before(head, from, d)
6121   d = node.new(DIR)
6122   d.dir = '-' .. dir
6123   node.insert_after(head, to, d)
6124 end
6125
6126 function Babel.bidi(head, ispar)
6127   local first_n, last_n          -- first and last char with nums
6128   local last_es                  -- an auxiliary 'last' used with nums
6129   local first_d, last_d          -- first and last char in L/R block
6130   local dir, dir_real
6131   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6132   local strong_lr = (strong == 'l') and 'l' or 'r'
6133   local outer = strong
6134
6135   local new_dir = false
6136   local first_dir = false
6137   local inmath = false
6138
6139   local last_lr
6140
6141   local type_n = ''
6142
6143   for item in node.traverse(head) do
6144
6145     -- three cases: glyph, dir, otherwise
6146     if item.id == node.id'glyph'
6147       or (item.id == 7 and item.subtype == 2) then
6148
6149       local itemchar
6150       if item.id == 7 and item.subtype == 2 then
6151         itemchar = item.replace.char
6152       else
6153         itemchar = item.char
6154       end

```

```

6155     local chardata = characters[itemchar]
6156     dir = chardata and chardata.d or nil
6157     if not dir then
6158         for nn, et in ipairs(ranges) do
6159             if itemchar < et[1] then
6160                 break
6161             elseif itemchar <= et[2] then
6162                 dir = et[3]
6163                 break
6164             end
6165         end
6166     end
6167     dir = dir or 'l'
6168     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).

```

6169     if new_dir then
6170         attr_dir = 0
6171         for at in node.traverse(item.attr) do
6172             if at.number == luatexbase.registernumber'bbl@attr@dir' then
6173                 attr_dir = at.value % 3
6174             end
6175         end
6176         if attr_dir == 1 then
6177             strong = 'r'
6178         elseif attr_dir == 2 then
6179             strong = 'al'
6180         else
6181             strong = 'l'
6182         end
6183         strong_lr = (strong == 'l') and 'l' or 'r'
6184         outer = strong_lr
6185         new_dir = false
6186     end
6187
6188     if dir == 'nsm' then dir = strong end -- W1

```

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

```

6189     dir_real = dir -- We need dir_real to set strong below
6190     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:

```

6191     if strong == 'al' then
6192         if dir == 'en' then dir = 'an' end -- W2
6193         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6194         strong_lr = 'r' -- W3
6195     end

```

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

```

6196     elseif item.id == node.id'dir' and not inmath then
6197         new_dir = true
6198         dir = nil
6199     elseif item.id == node.id'math' then
6200         inmath = (item.subtype == 0)

```

```

6201     else
6202         dir = nil          -- Not a char
6203     end

```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```

6204     if dir == 'en' or dir == 'an' or dir == 'et' then
6205         if dir ~= 'et' then
6206             type_n = dir
6207         end
6208         first_n = first_n or item
6209         last_n = last_es or item
6210         last_es = nil
6211     elseif dir == 'es' and last_n then -- W3+W6
6212         last_es = item
6213     elseif dir == 'cs' then          -- it's right - do nothing
6214     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6215         if strong_lr == 'r' and type_n ~= '' then
6216             dir_mark(head, first_n, last_n, 'r')
6217         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6218             dir_mark(head, first_n, last_n, 'r')
6219             dir_mark(head, first_d, last_d, outer)
6220             first_d, last_d = nil, nil
6221         elseif strong_lr == 'l' and type_n ~= '' then
6222             last_d = last_n
6223         end
6224         type_n = ''
6225         first_n, last_n = nil, nil
6226     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:

```

6227     if dir == 'l' or dir == 'r' then
6228         if dir ~= outer then
6229             first_d = first_d or item
6230             last_d = item
6231         elseif first_d and dir ~= strong_lr then
6232             dir_mark(head, first_d, last_d, outer)
6233             first_d, last_d = nil, nil
6234         end
6235     end

```

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

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

```

6236     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6237         item.char = characters[item.char] and
6238             characters[item.char].m or item.char
6239     elseif (dir or new_dir) and last_lr ~= item then
6240         local mir = outer .. strong_lr .. (dir or outer)
6241         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6242             for ch in node.traverse(node.next(last_lr)) do

```

```

6243         if ch == item then break end
6244         if ch.id == node.id'glyph' and characters[ch.char] then
6245             ch.char = characters[ch.char].m or ch.char
6246         end
6247     end
6248 end
6249 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).

```

6250     if dir == 'l' or dir == 'r' then
6251         last_lr = item
6252         strong = dir_real          -- Don't search back - best save now
6253         strong_lr = (strong == 'l') and 'l' or 'r'
6254     elseif new_dir then
6255         last_lr = nil
6256     end
6257 end

```

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

```

6258     if last_lr and outer == 'r' then
6259         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6260             if characters[ch.char] then
6261                 ch.char = characters[ch.char].m or ch.char
6262             end
6263         end
6264     end
6265     if first_n then
6266         dir_mark(head, first_n, last_n, outer)
6267     end
6268     if first_d then
6269         dir_mark(head, first_d, last_d, outer)
6270     end

```

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

```

6271     return node.prev(head) or head
6272 end
6273 </basic-r>

```

And here the Lua code for bidi=basic:

```

6274 <(*basic)
6275 Babel = Babel or {}
6276
6277 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6278
6279 Babel.fontmap = Babel.fontmap or {}
6280 Babel.fontmap[0] = {}          -- l
6281 Babel.fontmap[1] = {}          -- r
6282 Babel.fontmap[2] = {}          -- al/an
6283
6284 Babel.bidi_enabled = true
6285 Babel.mirroring_enabled = true
6286
6287 require('babel-data-bidi.lua')
6288
6289 local characters = Babel.characters
6290 local ranges = Babel.ranges
6291

```

```

6292 local DIR = node.id('dir')
6293 local GLYPH = node.id('glyph')
6294
6295 local function insert_implicit(head, state, outer)
6296   local new_state = state
6297   if state.sim and state.eim and state.sim ~= state.eim then
6298     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6299     local d = node.new(DIR)
6300     d.dir = '+' .. dir
6301     node.insert_before(head, state.sim, d)
6302     local d = node.new(DIR)
6303     d.dir = '-' .. dir
6304     node.insert_after(head, state.eim, d)
6305   end
6306   new_state.sim, new_state.eim = nil, nil
6307   return head, new_state
6308 end
6309
6310 local function insert_numeric(head, state)
6311   local new
6312   local new_state = state
6313   if state.san and state.ean and state.san ~= state.ean then
6314     local d = node.new(DIR)
6315     d.dir = '+TLT'
6316     _, new = node.insert_before(head, state.san, d)
6317     if state.san == state.sim then state.sim = new end
6318     local d = node.new(DIR)
6319     d.dir = '-TLT'
6320     _, new = node.insert_after(head, state.ean, d)
6321     if state.ean == state.eim then state.eim = new end
6322   end
6323   new_state.san, new_state.ean = nil, nil
6324   return head, new_state
6325 end
6326
6327 -- TODO - \hbox with an explicit dir can lead to wrong results
6328 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6329 -- was s made to improve the situation, but the problem is the 3-dir
6330 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6331 -- well.
6332
6333 function Babel.bidi(head, ispar, hdir)
6334   local d -- d is used mainly for computations in a loop
6335   local prev_d = ''
6336   local new_d = false
6337
6338   local nodes = {}
6339   local outer_first = nil
6340   local inmath = false
6341
6342   local glue_d = nil
6343   local glue_i = nil
6344
6345   local has_en = false
6346   local first_et = nil
6347
6348   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6349
6350   local save_outer

```

```

6351 local temp = node.get_attribute(head, ATDIR)
6352 if temp then
6353     temp = temp % 3
6354     save_outer = (temp == 0 and 'l') or
6355                 (temp == 1 and 'r') or
6356                 (temp == 2 and 'al')
6357 elseif ispar then -- Or error? Shouldn't happen
6358     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6359 else -- Or error? Shouldn't happen
6360     save_outer = ('TRT' == hdir) and 'r' or 'l'
6361 end
6362 -- when the callback is called, we are just _after_ the box,
6363 -- and the textdir is that of the surrounding text
6364 -- if not ispar and hdir ~= tex.textdir then
6365 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6366 -- end
6367 local outer = save_outer
6368 local last = outer
6369 -- 'al' is only taken into account in the first, current loop
6370 if save_outer == 'al' then save_outer = 'r' end
6371
6372 local fontmap = Babel.fontmap
6373
6374 for item in node.traverse(head) do
6375
6376     -- In what follows, #node is the last (previous) node, because the
6377     -- current one is not added until we start processing the neutrals.
6378
6379     -- three cases: glyph, dir, otherwise
6380     if item.id == GLYPH
6381         or (item.id == 7 and item.subtype == 2) then
6382
6383         local d_font = nil
6384         local item_r
6385         if item.id == 7 and item.subtype == 2 then
6386             item_r = item.replace -- automatic discs have just 1 glyph
6387         else
6388             item_r = item
6389         end
6390         local chardata = characters[item_r.char]
6391         d = chardata and chardata.d or nil
6392         if not d or d == 'nsm' then
6393             for nn, et in ipairs(ranges) do
6394                 if item_r.char < et[1] then
6395                     break
6396                 elseif item_r.char <= et[2] then
6397                     if not d then d = et[3]
6398                     elseif d == 'nsm' then d_font = et[3]
6399                     end
6400                     break
6401                 end
6402             end
6403         end
6404         d = d or 'l'
6405
6406         -- A short 'pause' in bidi for mapfont
6407         d_font = d_font or d
6408         d_font = (d_font == 'l' and 0) or
6409                 (d_font == 'nsm' and 0) or

```



```

6410             (d_font == 'r' and 1) or
6411             (d_font == 'al' and 2) or
6412             (d_font == 'an' and 2) or nil
6413 if d_font and fontmap and fontmap[d_font][item_r.font] then
6414     item_r.font = fontmap[d_font][item_r.font]
6415 end
6416
6417 if new_d then
6418     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6419     if inmath then
6420         attr_d = 0
6421     else
6422         attr_d = node.get_attribute(item, ATDIR)
6423         attr_d = attr_d % 3
6424     end
6425     if attr_d == 1 then
6426         outer_first = 'r'
6427         last = 'r'
6428     elseif attr_d == 2 then
6429         outer_first = 'r'
6430         last = 'al'
6431     else
6432         outer_first = 'l'
6433         last = 'l'
6434     end
6435     outer = last
6436     has_en = false
6437     first_et = nil
6438     new_d = false
6439 end
6440
6441 if glue_d then
6442     if (d == 'l' and 'l' or 'r') ~= glue_d then
6443         table.insert(nodes, {glue_i, 'on', nil})
6444     end
6445     glue_d = nil
6446     glue_i = nil
6447 end
6448
6449 elseif item.id == DIR then
6450     d = nil
6451     new_d = true
6452
6453 elseif item.id == node.id'glue' and item.subtype == 13 then
6454     glue_d = d
6455     glue_i = item
6456     d = nil
6457
6458 elseif item.id == node.id'math' then
6459     inmath = (item.subtype == 0)
6460
6461 else
6462     d = nil
6463 end
6464
6465 -- AL <= EN/ET/ES      -- W2 + W3 + W6
6466 if last == 'al' and d == 'en' then
6467     d = 'an'          -- W3
6468 elseif last == 'al' and (d == 'et' or d == 'es') then

```

```

6469     d = 'on'          -- W6
6470 end
6471
6472 -- EN + CS/ES + EN      -- W4
6473 if d == 'en' and #nodes >= 2 then
6474     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6475         and nodes[#nodes-1][2] == 'en' then
6476         nodes[#nodes][2] = 'en'
6477     end
6478 end
6479
6480 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6481 if d == 'an' and #nodes >= 2 then
6482     if (nodes[#nodes][2] == 'cs')
6483         and nodes[#nodes-1][2] == 'an' then
6484         nodes[#nodes][2] = 'an'
6485     end
6486 end
6487
6488 -- ET/EN                  -- W5 + W7->l / W6->on
6489 if d == 'et' then
6490     first_et = first_et or (#nodes + 1)
6491 elseif d == 'en' then
6492     has_en = true
6493     first_et = first_et or (#nodes + 1)
6494 elseif first_et then      -- d may be nil here !
6495     if has_en then
6496         if last == 'l' then
6497             temp = 'l'    -- W7
6498         else
6499             temp = 'en'   -- W5
6500         end
6501     else
6502         temp = 'on'      -- W6
6503     end
6504     for e = first_et, #nodes do
6505         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6506     end
6507     first_et = nil
6508     has_en = false
6509 end
6510
6511 -- Force mathdir in math if ON (currently works as expected only
6512 -- with 'l')
6513 if inmath and d == 'on' then
6514     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6515 end
6516
6517 if d then
6518     if d == 'al' then
6519         d = 'r'
6520         last = 'al'
6521     elseif d == 'l' or d == 'r' then
6522         last = d
6523     end
6524     prev_d = d
6525     table.insert(nodes, {item, d, outer_first})
6526 end
6527

```

```

6528     outer_first = nil
6529
6530 end
6531
6532 -- TODO -- repeated here in case EN/ET is the last node. Find a
6533 -- better way of doing things:
6534 if first_et then      -- dir may be nil here !
6535     if has_en then
6536         if last == 'l' then
6537             temp = 'l'    -- W7
6538         else
6539             temp = 'en'   -- W5
6540         end
6541     else
6542         temp = 'on'      -- W6
6543     end
6544     for e = first_et, #nodes do
6545         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6546     end
6547 end
6548
6549 -- dummy node, to close things
6550 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6551
6552 ----- NEUTRAL -----
6553
6554 outer = save_outer
6555 last = outer
6556
6557 local first_on = nil
6558
6559 for q = 1, #nodes do
6560     local item
6561
6562     local outer_first = nodes[q][3]
6563     outer = outer_first or outer
6564     last = outer_first or last
6565
6566     local d = nodes[q][2]
6567     if d == 'an' or d == 'en' then d = 'r' end
6568     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6569
6570     if d == 'on' then
6571         first_on = first_on or q
6572     elseif first_on then
6573         if last == d then
6574             temp = d
6575         else
6576             temp = outer
6577         end
6578         for r = first_on, q - 1 do
6579             nodes[r][2] = temp
6580             item = nodes[r][1]    -- MIRRORING
6581             if Babel.mirroring_enabled and item.id == GLYPH
6582                 and temp == 'r' and characters[item.char] then
6583                 local font_mode = font.fonts[item.font].properties.mode
6584                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6585                     item.char = characters[item.char].m or item.char
6586                 end

```

```

6587         end
6588     end
6589     first_on = nil
6590 end
6591
6592     if d == 'r' or d == 'l' then last = d end
6593 end
6594
6595 ----- IMPLICIT, REORDER -----
6596
6597 outer = save_outer
6598 last = outer
6599
6600 local state = {}
6601 state.has_r = false
6602
6603 for q = 1, #nodes do
6604
6605     local item = nodes[q][1]
6606
6607     outer = nodes[q][3] or outer
6608
6609     local d = nodes[q][2]
6610
6611     if d == 'nsm' then d = last end          -- W1
6612     if d == 'en' then d = 'an' end
6613     local isdir = (d == 'r' or d == 'l')
6614
6615     if outer == 'l' and d == 'an' then
6616         state.san = state.san or item
6617         state.ean = item
6618     elseif state.san then
6619         head, state = insert_numeric(head, state)
6620     end
6621
6622     if outer == 'l' then
6623         if d == 'an' or d == 'r' then      -- im -> implicit
6624             if d == 'r' then state.has_r = true end
6625             state.sim = state.sim or item
6626             state.eim = item
6627         elseif d == 'l' and state.sim and state.has_r then
6628             head, state = insert_implicit(head, state, outer)
6629         elseif d == 'l' then
6630             state.sim, state.eim, state.has_r = nil, nil, false
6631         end
6632     else
6633         if d == 'an' or d == 'l' then
6634             if nodes[q][3] then -- nil except after an explicit dir
6635                 state.sim = item -- so we move sim 'inside' the group
6636             else
6637                 state.sim = state.sim or item
6638             end
6639             state.eim = item
6640         elseif d == 'r' and state.sim then
6641             head, state = insert_implicit(head, state, outer)
6642         elseif d == 'r' then
6643             state.sim, state.eim = nil, nil
6644         end
6645     end

```

```

6646
6647   if isdir then
6648       last = d           -- Don't search back - best save now
6649   elseif d == 'on' and state.san then
6650       state.san = state.san or item
6651       state.ean = item
6652   end
6653
6654 end
6655
6656 return node.prev(head) or head
6657 end
6658 </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.

```

6659 < *nil>
6660 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6661 \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.

```

6662 \ifx\l@nil\undefined
6663   \newlanguage\l@nil
6664   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
6665   \let\bbl@elt\relax
6666   \edef\bbl@languages{% Add it to the list of languages
6667     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
6668 \fi

```

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

```

6669 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
6670 \let\captionnil\@empty
6671 \let\datenil\@empty

```

```
6672 \ldf@finish{nil}
6673 \</nil>
```

16.1 Not renaming hyphen.tex

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).

```

6674 (*bplain | blplain)
6675 \catcode`\{=1 % left brace is begin-group character
6676 \catcode`\}=2 % right brace is end-group character
6677 \catcode`\#=6 % hash mark is macro parameter character

```

```
6678 \openin 0 hyphen.cfg
6679 \ifeof0
6680 \else
6681   \let\input
```

```

6682 \def\input #1 {%
6683   \let\input\@
6684   \a hyphen.cfg
6685   \let\@undefined
6686 }
6687 \fi
6688 </bplain | bplain>

```

```
6689 <bplain>\a plain.tex
6690 <blplain>\a lplain.tex
```

```
6691 <bplain>\def\fmtname{babel-plain}
6692 <blplain>\def\fmtname{babel-lplain}
```

205

16.2 Emulating some L^AT_EX features

The following code duplicates or emulates parts of L^AT_EX 2_ε that are needed for babel.

```
6693 <<*Emulate LaTeX>> ≡
6694 % == Code for plain ==
6695 \def\@empty{}
6696 \def\loadlocalcfg#1{%
6697   \openin0#1.cfg
6698   \ifeof0
6699     \closein0
6700   \else
6701     \closein0
6702     {\immediate\write16{*****}%
6703      \immediate\write16{* Local config file #1.cfg used}%
6704      \immediate\write16{*}%
6705     }
6706     \input #1.cfg\relax
6707   \fi
6708   \@endofldf}
```

16.3 General tools

A number of L^AT_EX macro's that are needed later on.

```
6709 \long\def\@firstofone#1{#1}
6710 \long\def\@firstoftwo#1#2{#1}
6711 \long\def\@secondoftwo#1#2{#2}
6712 \def\@nnil{\@nil}
6713 \def\@gobbletwo#1#2{}
6714 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6715 \def\@star@or@long#1{%
6716   \@ifstar
6717   {\let\l@ngrel@x\relax#1}%
6718   {\let\l@ngrel@x\long#1}}
6719 \let\l@ngrel@x\relax
6720 \def\@car#1#2\@nil{#1}
6721 \def\@cdr#1#2\@nil{#2}
6722 \let\@typeset@protect\relax
6723 \let\protected@edef\edef
6724 \long\def\@gobble#1{}
6725 \edef\@backslashchar{\expandafter\@gobble\string\}
6726 \def\strip@prefix#1>{}
6727 \def\g@addto@macro#1#2{%
6728   \toks@\expandafter{#1#2}%
6729   \xdef#1{\the\toks@}}
6730 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6731 \def\@nameuse#1{\csname #1\endcsname}
6732 \def\@ifundefined#1{%
6733   \expandafter\ifx\csname#1\endcsname\relax
6734     \expandafter\@firstoftwo
6735   \else
6736     \expandafter\@secondoftwo
6737   \fi}
6738 \def\@expandtwoargs#1#2#3{%
6739   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6740 \def\zap@space#1 #2{%
6741   #1%
6742   \ifx#2\@empty\else\expandafter\zap@space\fi
6743   #2}
```

```
6744 \let\bbl@trace\@gobble
```

$\LaTeX 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
6745 \ifx\@preamblecmds\@undefined
6746   \def\@preamblecmds{}
6747 \fi
6748 \def\@onlypreamble#1{%
6749   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6750     \@preamblecmds\do#1}}
6751 \@onlypreamble\@onlypreamble
```

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

```
6752 \def\begindocument{%
6753   \@begindocumenthook
6754   \global\let\@begindocumenthook\@undefined
6755   \def\do##1{\global\let##1\@undefined}%
6756   \@preamblecmds
6757   \global\let\do\noexpand}
6758 \ifx\@begindocumenthook\@undefined
6759   \def\@begindocumenthook{}
6760 \fi
6761 \@onlypreamble\@begindocumenthook
6762 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```
6763 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6764 \@onlypreamble\AtEndOfPackage
6765 \def\@endofldf{}
6766 \@onlypreamble\@endofldf
6767 \let\bbl@afterlang\@empty
6768 \chardef\bbl@opt@hyphenmap\z@
```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```
6769 \catcode`\&=\z@
6770 \ifx&\if@files\@undefined
6771   \expandafter\let\csname if@files\expandafter\endcsname
6772     \csname iffalse\endcsname
6773 \fi
6774 \catcode`\&=4
```

Mimick \LaTeX 's commands to define control sequences.

```
6775 \def\newcommand{\@star@or@long\new@command}
6776 \def\new@command#1{%
6777   \@testopt{\@newcommand#1}0}
6778 \def\@newcommand#1[#2]{%
6779   \@ifnextchar [{\@xargdef#1[#2]}%
6780     {\@argdef#1[#2]}}
6781 \long\def\@argdef#1[#2]#3{%
6782   \@yargdef#1\@ne{#2}{#3}}
6783 \long\def\@xargdef#1[#2][#3]#4{%
6784   \expandafter\def\expandafter#1\expandafter{%
6785     \expandafter\@protected@testopt\expandafter #1%
6786     \csname\string#1\expandafter\endcsname{#3}}%
6787   \expandafter\@yargdef \csname\string#1\endcsname
6788   \tw@{#2}{#4}}
```



```

6789 \long\def\@yargdef#1#2#3{%
6790   \@tempcnta#3\relax
6791   \advance \@tempcnta \@ne
6792   \let\@hash@\relax
6793   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6794   \@tempcntb #2%
6795   \@whilenum \@tempcntb < \@tempcnta
6796   \do{%
6797     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6798     \advance\@tempcntb \@ne}%
6799   \let\@hash@###
6800   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
6801 \def\providecommand{\@star@or@long\provide@command}
6802 \def\provide@command#1{%
6803   \begingroup
6804   \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
6805   \endgroup
6806   \expandafter\@ifundefined\@gtempa
6807     {\def\reserved@a{\new@command#1}}%
6808     {\let\reserved@a\relax
6809      \def\reserved@a{\new@command\reserved@a}}%
6810   \reserved@a}%
6811 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6812 \def\declare@robustcommand#1{%
6813   \edef\reserved@a{\string#1}%
6814   \def\reserved@b{#1}%
6815   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6816   \edef#1{%
6817     \ifx\reserved@a\reserved@b
6818       \noexpand\x@protect
6819       \noexpand#1%
6820     \fi
6821     \noexpand\protect
6822     \expandafter\@gobble\string#1 \endcsname
6823   }%
6824   \expandafter\new@command\csname
6825     \expandafter\@gobble\string#1 \endcsname
6826 }
6827 \def\x@protect#1{%
6828   \ifx\protect\@typeset@protect\else
6829     \@x@protect#1%
6830   \fi
6831 }
6832 }
6833 \catcode`\&=\z@ % Trick to hide conditionals
6834 \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`.

```

6835 \def\bbl@tempa{\csname newif\endcsname&\ifin@}
6836 \catcode`\&=4
6837 \ifx\in@\@undefined
6838   \def\in@#1#2{%
6839     \def\in@@##1##2##3\in@@{%
6840       \ifx\in@@##2\in@false\else\in@true\fi}%
6841     \in@@#2#1\in@\in@@}
6842 \else
6843   \let\bbl@tempa\@empty

```

```
6844 \fi
6845 \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).

```
6846 \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.

```
6847 \def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\LaTeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```
6848 \ifx\@tempcnta\@undefined
6849   \csname newcount\endcsname\@tempcnta\relax
6850 \fi
6851 \ifx\@tempcntb\@undefined
6852   \csname newcount\endcsname\@tempcntb\relax
6853 \fi
```

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

```
6854 \ifx\bye\@undefined
6855   \advance\count10 by -2\relax
6856 \fi
6857 \ifx\@ifnextchar\@undefined
6858   \def\@ifnextchar#1#2#3{%
6859     \let\reserved@d=#1%
6860     \def\reserved@a{#2}\def\reserved@b{#3}%
6861     \futurelet\@let@token\@ifnch}
6862 \def\@ifnch{%
6863   \ifx\@let@token\@sptoken
6864     \let\reserved@c\@xifnch
6865   \else
6866     \ifx\@let@token\reserved@d
6867       \let\reserved@c\reserved@a
6868     \else
6869       \let\reserved@c\reserved@b
6870     \fi
6871   \fi
6872   \reserved@c}
6873 \def\:\let\@sptoken= } \: % this makes \@sptoken a space token
6874 \def\:\@xifnch\expandafter\def\:\futurelet\@let@token\@ifnch}
6875 \fi
6876 \def\@testopt#1#2{%
6877   \@ifnextchar[#{1}{#1[#2]}}
6878 \def\@protected@testopt#1{%
6879   \ifx\protect\@typeset@protect
6880     \expandafter\@testopt
6881   \else
6882     \@x@protect#1%
6883   \fi}
6884 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6885   #2\relax}\fi}
6886 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6887   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```
6888 \def\DeclareTextCommand{%
6889   \@dec@text@cmd\providecommand
6890 }
6891 \def\ProvideTextCommand{%
6892   \@dec@text@cmd\providecommand
6893 }
6894 \def\DeclareTextSymbol#1#2#3{%
6895   \@dec@text@cmd\chardef#1{#2}#3\relax
6896 }
6897 \def\@dec@text@cmd#1#2#3{%
6898   \expandafter\def\expandafter#2%
6899     \expandafter{%
6900       \csname#3-cmd\expandafter\endcsname
6901       \expandafter#2%
6902       \csname#3\string#2\endcsname
6903     }%
6904 %   \let\@ifdefinable\rc@ifdefinable
6905   \expandafter#1\csname#3\string#2\endcsname
6906 }
6907 \def\@current@cmd#1{%
6908   \ifx\protect\@typeset@protect\else
6909     \noexpand#1\expandafter\@gobble
6910   \fi
6911 }
6912 \def\@changed@cmd#1#2{%
6913   \ifx\protect\@typeset@protect
6914     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6915       \expandafter\ifx\csname ?\string#1\endcsname\relax
6916         \expandafter\def\csname ?\string#1\endcsname{%
6917           \@changed@x@err{#1}%
6918         }%
6919       \fi
6920       \global\expandafter\let
6921         \csname\cf@encoding\string#1\expandafter\endcsname
6922         \csname ?\string#1\endcsname
6923     \fi
6924     \csname\cf@encoding\string#1%
6925       \expandafter\endcsname
6926   \else
6927     \noexpand#1%
6928   \fi
6929 }
6930 \def\@changed@x@err#1{%
6931   \errhelp{Your command will be ignored, type <return> to proceed}%
6932   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6933 \def\DeclareTextCommandDefault#1{%
6934   \DeclareTextCommand#1?%
6935 }
6936 \def\ProvideTextCommandDefault#1{%
6937   \ProvideTextCommand#1?%
6938 }
6939 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6940 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6941 \def\DeclareTextAccent#1#2#3{%
6942   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6943 }
```

```

6944 \def\DeclareTextCompositeCommand#1#2#3#4{%
6945   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6946   \edef\reserved@b{\string##1}%
6947   \edef\reserved@c{%
6948     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6949   \ifx\reserved@b\reserved@c
6950     \expandafter\expandafter\expandafter\ifx
6951       \expandafter\@car\reserved@a\relax\relax\@nil
6952       \@text@composite
6953     \else
6954       \edef\reserved@b##1{%
6955         \def\expandafter\noexpand
6956           \csname#2\string#1\endcsname####1{%
6957             \noexpand\@text@composite
6958             \expandafter\noexpand\csname#2\string#1\endcsname
6959             ####1\noexpand\@empty\noexpand\@text@composite
6960             {##1}%
6961           }%
6962       }%
6963       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6964     \fi
6965     \expandafter\def\csname\expandafter\string\csname
6966       #2\endcsname\string#1-\string#3\endcsname{#4}
6967   \else
6968     \errhelp{Your command will be ignored, type <return> to proceed}%
6969     \errmessage{\string\DeclareTextCompositeCommand\space used on
6970       inappropriate command \protect#1}
6971   \fi
6972 }
6973 \def\@text@composite#1#2#3\@text@composite{%
6974   \expandafter\@text@composite@x
6975     \csname\string#1-\string#2\endcsname
6976 }
6977 \def\@text@composite@x#1#2{%
6978   \ifx#1\relax
6979     #2%
6980   \else
6981     #1%
6982   \fi
6983 }
6984 %
6985 \def\@strip@args#1:#2-#3\@strip@args{#2}
6986 \def\DeclareTextComposite#1#2#3#4{%
6987   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6988   \bgroup
6989     \lcode`\@=#4%
6990     \lowercase{%
6991       \egroup
6992       \reserved@a @%
6993     }%
6994 }
6995 %
6996 \def\UseTextSymbol#1#2{#2}
6997 \def\UseTextAccent#1#2#3{}
6998 \def\@use@text@encoding#1{}
6999 \def\DeclareTextSymbolDefault#1#2{%
7000   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7001 }
7002 \def\DeclareTextAccentDefault#1#2{%

```

```

7003 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7004 }
7005 \def\cf@encoding{OT1}

```

Currently we only use the $\text{\texttt{\textbackslash UseTextAccent}}$ method for accents for those that are known to be made active in *some* language definition file.

```

7006 \DeclareTextAccent{"}{OT1}{127}
7007 \DeclareTextAccent{'}{OT1}{19}
7008 \DeclareTextAccent{^}{OT1}{94}
7009 \DeclareTextAccent{`}{OT1}{18}
7010 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN $\text{\texttt{\textbackslash TeX}}$.

```

7011 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7012 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
7013 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
7014 \DeclareTextSymbol{\textquoteright}{OT1}{''}
7015 \DeclareTextSymbol{\i}{OT1}{16}
7016 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the $\text{\texttt{\textbackslash scriptsize}}$ control sequence to be available. Because plain $\text{\texttt{\textbackslash TeX}}$ doesn't have such a sophisticated font mechanism as $\text{\texttt{\textbackslash TeX}}$ has, we just `\let` it to `\sevenrm`.

```

7017 \ifx\scriptsize\undefined
7018 \let\scriptsize\sevenrm
7019 \fi
7020 % End of code for plain
7021 <</Emulate LaTeX>>

```

A proxy file:

```

7022 <(*plain)
7023 \input babel.def
7024 </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 $\text{\texttt{\textbackslash TeX}}$ styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The $\text{\texttt{\textbackslash TeX}}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *$\text{\texttt{\textbackslash TeX}}$, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: $\text{\texttt{\textbackslash TeX}}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German $\text{\texttt{\textbackslash TeX}}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International $\text{\texttt{\textbackslash TeX}}$ is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).