

Babel

Version 3.53.2290
2021/02/21

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	35
1.20	Hyphenation and line breaking	37
1.21	Selection based on BCP 47 tags	39
1.22	Selecting scripts	40
1.23	Selecting directions	41
1.24	Language attributes	45
1.25	Hooks	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	50
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	61

5	Identification and loading of required files	62
6	locale directory	62
7	Tools	62
7.1	Multiple languages	67
7.2	The Package File (\LaTeX , babel.sty)	67
7.3	base	69
7.4	Conditional loading of shorthands	71
7.5	Cross referencing macros	73
7.6	Marks	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	78
7.9	Basic bidi support	80
7.10	Local Language Configuration	85
8	The kernel of Babel (babel.def, common)	89
8.1	Tools	89
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	128
9.12.1	Quotation marks	128
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	177
13.7	Layout	188
13.8	Auto bidi with basic and basic-r	192
14	Data for CJK	203
15	The ‘nil’ language	203
16	Support for Plain T_EX (plain.def)	204
16.1	Not renaming hyphen.tex	204
16.2	Emulating some L _A T _E X features	204
16.3	General tools	205
16.4	Encoding related macros	208
17	Acknowledgements	211

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’	28
Package babel Info: The following fonts are not babel standard families	29

Part I

User guide

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

EXAMPLE Here is a simple full example for “traditional” \TeX engines (see below for `xetex` and `luatex`). The packages `fontenc` and `inputenc` do not belong to `babel`, but they are included in the example because typically you will need them. 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). To set hyphenation exceptions, use `\babelhyphenation` (see below).

1.9 More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with the option `fontenc`.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 discretionary and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

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

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 [$\langle\text{language}\rangle$, $\langle\text{language}\rangle$, ...]{ $\langle\text{shorthand}\rangle$ }{ $\langle\text{code}\rangle$ }

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

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

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

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

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

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

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

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

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

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

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

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).⁶ Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

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

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

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

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.

```
\newcommand{\myipa}[1]{\{\languageshortands{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.

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

⁷Thanks to Enrico Gregorio

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

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=** `<char><char>... | 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 \TeX 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 \TeX macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from `varioref` and `ifthen`).

With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in $\epsilon\text{T}_{\text{E}}\text{X}$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

- math=** active | normal
- Shorthands are mainly intended for text, not for math. By setting this option with the value `normal` they are deactivated in math mode (default is `active`) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.
- config=** $\langle file \rangle$
- Load $\langle file \rangle$.`cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).
- main=** $\langle language \rangle$
- Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.
- headfoot=** $\langle language \rangle$
- 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 `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** **New 3.9l** No warnings and no *infos* are written to the log file.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 `generic` (for traditional $\text{T}_{\text{E}}\text{X}$, LICR and ASCII strings), `unicode` (for engines like `xetex` and `luatex`) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (`T1`, `T2A`, `LGR`, `L7X`...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ tools, so use it only as a last resort).
- hyphenmap=** off | first | select | other | other*
- New 3.9g** Sets the behavior of case mapping for hyphenation, provided the language defines it.¹⁰ It can take the following values:

off deactivates this feature and no case mapping is applied;

⁹You can use alternatively the package `silence`.

¹⁰Turned off in plain.

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:

```
\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.

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.

1.13 ini files

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

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

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them by means of \babelprovide. In other words, \babelprovide is mainly meant for auxiliary tasks, and as alternative when the ldf, for some reason, does work as expected.

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

LUATEX/XETEX

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{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 bid 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 lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{၁၀ ၁၁ ၁၂ ၁၃ ၁၄} % 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}	dav	Taita
agq	Aghem	de-AT	German ^{ul}
ak	Akan	de-CH	German ^{ul}
am	Amharic ^{ul}	de	German ^{ul}
ar	Arabic ^{ul}	dje	Zarma
ar-DZ	Arabic ^{ul}	dsb	Lower Sorbian ^{ul}
ar-MA	Arabic ^{ul}	dua	Duala
ar-SY	Arabic ^{ul}	dyo	Jola-Fonyi
as	Assamese	dz	Dzongkha
asa	Asu	ebu	Embu
ast	Asturian ^{ul}	ee	Ewe
az-Cyrl	Azerbaijani	el	Greek ^{ul}
az-Latn	Azerbaijani	el-polyton	Polytonic Greek ^{ul}
az	Azerbaijani ^{ul}	en-AU	English ^{ul}
bas	Basaa	en-CA	English ^{ul}
be	Belarusian ^{ul}	en-GB	English ^{ul}
bem	Bemba	en-NZ	English ^{ul}
bez	Bena	en-US	English ^{ul}
bg	Bulgarian ^{ul}	en	English ^{ul}
bm	Bambara	eo	Esperanto ^{ul}
bn	Bangla ^{ul}	es-MX	Spanish ^{ul}
bo	Tibetan ^u	es	Spanish ^{ul}
brx	Bodo	et	Estonian ^{ul}
bs-Cyrl	Bosnian	eu	Basque ^{ul}
bs-Latn	Bosnian ^{ul}	ewo	Ewondo
bs	Bosnian ^{ul}	fa	Persian ^{ul}
ca	Catalan ^{ul}	ff	Fulah
ce	Chechen	fi	Finnish ^{ul}
cgg	Chiga	fil	Filipino
chr	Cherokee	fo	Faroese
ckb	Central Kurdish	fr	French ^{ul}
cop	Coptic	fr-BE	French ^{ul}
cs	Czech ^{ul}	fr-CA	French ^{ul}
cu	Church Slavic	fr-CH	French ^{ul}
cu-Cyrs	Church Slavic	fr-LU	French ^{ul}
cu-Glag	Church Slavic	fur	Friulian ^{ul}
cy	Welsh ^{ul}	fy	Western Frisian
da	Danish ^{ul}	ga	Irish ^{ul}

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	mgh	Meta'
he	Hebrew ^{ul}	mgo	Meta'
hi	Hindi ^u	mk	Macedonian ^{ul}
hr	Croatian ^{ul}	ml	Malayalam ^{ul}
hsb	Upper Sorbian ^{ul}	mn	Mongolian
hu	Hungarian ^{ul}	mr	Marathi ^{ul}
hy	Armenian ^u	ms-BN	Malay ¹
ia	Interlingua ^{ul}	ms-SG	Malay ¹
id	Indonesian ^{ul}	ms	Malay ^{ul}
ig	Igbo	mt	Maltese
ii	Sichuan Yi	mua	Mundang
is	Icelandic ^{ul}	my	Burmese
it	Italian ^{ul}	mzn	Mazanderani
ja	Japanese	naq	Nama
jgo	Ngomba	nb	Norwegian Bokmål ^{ul}
jmc	Machame	nd	North Ndebele
ka	Georgian ^{ul}	ne	Nepali
kab	Kabyle	nl	Dutch ^{ul}
kam	Kamba	nmg	Kwasio
kde	Makonde	nn	Norwegian Nynorsk ^{ul}
kea	Kabuverdianu	nnh	Ngiemboon
khq	Koyra Chiini	nus	Nuer
ki	Kikuyu	nyn	Nyankole
kk	Kazakh	om	Oromo
kkj	Kako	or	Odia
kl	Kalaallisut	os	Ossetic
kln	Kalenjin	pa-Arab	Punjabi
km	Khmer	pa-Guru	Punjabi
kn	Kannada ^{ul}	pa	Punjabi
ko	Korean	pl	Polish ^{ul}
kok	Konkani	pms	Piedmontese ^{ul}
ks	Kashmiri	ps	Pashto
ksb	Shambala	pt-BR	Portuguese ^{ul}
ksf	Bafia	pt-PT	Portuguese ^{ul}
ksh	Colognian	pt	Portuguese ^{ul}
kw	Cornish	qu	Quechua
ky	Kyrgyz	rm	Romansh ^{ul}
lag	Langi	rn	Rundi
lb	Luxembourgish	ro	Romanian ^{ul}
lg	Ganda	rof	Rombo
lkt	Lakota	ru	Russian ^{ul}
ln	Lingala	rw	Kinyarwanda
lo	Lao ^{ul}	rwk	Rwa
lrc	Northern Luri	sa-Beng	Sanskrit
		sa-Deva	Sanskrit

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

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

aghem	arabic-morocco
akan	arabic-MA
albanian	arabic-syria
american	arabic-SY
amharic	armenian
ancientgreek	assamese
arabic	asturian
arabic-algeria	asu
arabic-DZ	australian

austrian	churchsslavic-glagolitic
azerbaijani-cyrillic	colognian
azerbaijani-cyrl	cornish
azerbaijani-latin	croatian
azerbaijani-latn	czech
azerbaijani	danish
bafia	duala
bambara	dutch
basaa	dzongkha
basque	embu
belarusian	english-au
bemba	english-australia
ben	english-ca
bengali	english-canada
bodo	english-gb
bosnian-cyrillic	english-newzealand
bosnian-cyrl	english-nz
bosnian-latin	english-unitedkingdom
bosnian-latn	english-unitedstates
bosnian	english-us
brazilian	english
breton	esperanto
british	estonian
bulgarian	ewe
burmese	ewondo
canadian	faroes
cantonese	filipino
catalan	finnish
centralatlantamazight	french-be
centralkurdish	french-belgium
chechen	french-ca
cherokee	french-canada
chiga	french-ch
chinese-hans-hk	french-lu
chinese-hans-mo	french-luxembourg
chinese-hans-sg	french-switzerland
chinese-hans	french
chinese-hant-hk	friulian
chinese-hant-mo	fulah
chinese-hant	galician
chinese-simplified-hongkongsarchina	ganda
chinese-simplified-macausarchina	georgian
chinese-simplified-singapore	german-at
chinese-simplified	german-austria
chinese-traditional-hongkongsarchina	german-ch
chinese-traditional-macausarchina	german-switzerland
chinese-traditional	german
chinese	greek
churchslavic	gujarati
churchslavic-cyrs	gusii
churchslavic-oldcyrillic ¹³	hausa-gh
churchsslavic-glag	hausa-ghana

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

hausa-ne	malay-singapore
hausa-niger	malay
hausa	malayalam
hawaiian	maltese
hebrew	manx
hindi	marathi
hungarian	masai
icelandic	mazanderani
igbo	meru
inarisami	meta
indonesian	mexican
interlingua	mongolian
irish	morisyen
italian	mundang
japanese	nama
jolafonyi	nepali
kabuverdianu	newzealand
kabyle	ngiemboon
kako	ngomba
kalaallisut	norsk
kalenjin	northernluri
kamba	northernsami
kannada	northndebele
kashmiri	norwegianbokmal
kazakh	norwegiannynorsk
khmer	nswissgerman
kikuyu	nuer
kinyarwanda	nyankole
konkani	nynorsk
korean	occitan
koyraborosenni	oriya
koyrachiini	oromo
kwasio	ossetic
kyrgyz	pashto
lakota	persian
langi	piedmontese
lao	polish
latvian	polytonicgreek
lingala	portuguese-br
lithuanian	portuguese-brazil
lowersorbian	portuguese-portugal
lsorbian	portuguese-pt
lubakatanga	portuguese
luo	punjabi-arab
luxembourgish	punjabi-arabic
luyia	punjabi-gurmukhi
macedonian	punjabi-guru
machame	punjabi
makhuwameetto	quechua
makonde	romanian
malagasy	romansh
malay-bn	rombo
malay-brunei	rundi
malay-sg	russian

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

Modifying and adding values to ini files

New 3.39 There is a way to modify the values of ini files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use

something like `numbers/digits.native=abcdefghijkl`. Keys may be added, too. Without `import` you may modify the identification keys.

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

1.14 Selecting fonts

New 3.15 Babel provides a high level interface on top of `fontspec` to select fonts. There is no need to load `fontspec` explicitly – babel does it for you with the first `\babelfont`.¹⁴

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

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:

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

(In this particular case, instead of the captions group you may need to modify the `captions.list` 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 added 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= $\langle\text{language-tag}\rangle$

Loads only the strings. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Same for `\Alph`.

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

onchar= `ids | fonts`

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

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

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

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

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

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

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):

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

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

The styles are:

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

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

1.18 Dates

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

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

By default the calendar is the Gregorian, but a ini files may define strings for other calendars (currently ar, ar-*, he, fa, hi.) In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like 30. *Çileyä Pêşîn 2019*, but with variant=iza fa it prints 31'ê *Çileyä Pêşînê 2019*.

1.19 Accessing language info

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

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

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

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

\localeinfo $\{\langle field \rangle\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\localeid

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\babelposthyphenation` {*hyphenrules-name*}{*lua-pattern*}{*replacement*}

New 3.37-3.39 *With luatex* it is now possible to define non-standard hyphenation rules, like `f-f → ff-f`, repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. No rules are currently provided by default, but they can be defined as shown in the following example, where `{1}` is the first captured char (between `()` in the pattern):

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

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

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

¹⁵With `luatex` exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and `babel` only provides the most basic tools.

See the [babel wiki](#) for a more detailed description and some examples. It also describes 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: `fr-Latn-FR` → `fr-Latn` → `fr-FR` → `fr`. Languages with the same resolved name are considered the same. Case is normalized

before, so that `fr-latn-fr` \rightarrow `fr-Latn-FR`. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

`autoload.bcp47` with values `on` and `off`.

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

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

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

```
\babeladjust{ bcp47.toname = on }
```

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

1.22 Selecting scripts

Currently `babel` provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the

Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.¹⁶

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

`\ensureascii` $\langle text \rangle$

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

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

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

1.23 Selecting directions

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

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

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

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

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.

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}
```

```

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

\begin{document}

Most Arabic speakers consider the two varieties to be two registers
of one language, although the two registers can be referred to in
Arabic as \textit{fuṣṣḥā l-‘aṣr} (MSA) and
\textit{fuṣṣḥā 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`.

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

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

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

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

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

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

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

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

extras is used for miscellaneous readjustments which do not fit into the previous groups. Currently redefines in `luatex` `\underline` and `\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 `rl` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

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

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

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

New 3.17 Something like:

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

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

```
\footnote{(\foreignlanguage{\language name}{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 name}{\{ \}}%  
\BabelFootnote{\localfootnote}{\language name}{\{ \}}%  
\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:

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.26 Languages supported by babel with ldf files

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

Afrikaans afrikaans

Azerbaijani azerbaijani

Basque basque

Breton breton

Bulgarian bulgarian

Catalan catalan

Croatian croatian

Czech czech

Danish danish

Dutch dutch

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

Esperanto esperanto

Estonian estonian

Finnish finnish

French french, francais, canadien, acadian

Galician galician

German austrian, german, germanb, ngerman, naustrian

Greek greek, polutonikogreek

Hebrew hebrew

Icelandic icelandic

Indonesian indonesian (bahasa, indon, bahasai)

Interlingua interlingua

Irish Gaelic irish

Italian italian

Latin latin

Lower Sorbian lowersorbian

Malay malay, melayu (bahasam)

North Sami samin

Norwegian norsk, nynorsk

Polish polish

Portuguese portuguese, brazilian (portuges, brazil)¹⁹

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

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:

```

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

```

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

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

1.29 Tips, workarounds, known issues and notes

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

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

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

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

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

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

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.

- 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 \TeX internals.

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

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

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

1.31 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

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

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

Labels

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

2 Loading languages with language.dat

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

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

2.1 Format

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

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

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

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

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

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

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

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

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

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date<lang>` but not `\captions<lang>` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

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

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

3.1 Guidelines for contributed languages

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

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

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

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

²⁶But not removed, for backward compatibility.

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Skeleton

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

```

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

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

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

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

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

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

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

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

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

\EndBabelCommands

```



```

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

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

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

`\declare@shorthand`

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

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

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

3.5 Support for saving macro definitions

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

`\babel@save`

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

`\babel@savevariable`

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

²⁷This mechanism was introduced by Bernd Raichle.

anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the *<variable>*.

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

3.6 Support for extending macros

`\addto` The macro `\addto{<control sequence>}{<TeX code>}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`. Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

3.7 Macros common to a number of languages

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

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

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

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

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

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

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

²⁸In future releases further categories may be added.

```

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

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

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

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

\EndBabelCommands

```

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

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

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

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

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

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

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

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

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

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

\SetStringLoop {<macro-name>}{<string-list>}

A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with abday):

```
\SetStringLoop{abmon#1name}{en,fb,mr,ab,my,jn,jl,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
```

#1 is replaced by the roman numeral.

\SetCase [*<map-list>*]{<toupper-code>}{<tolower-code>}

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let\BB\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A *<map-list>* is a series of macros using the internal format of \@uclclist (eg, \bb\BB\cc\CC). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in L^AT_EX, we can set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap {<to-lower-macros>}

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

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.53.2290>>
2 <<date=2021/02/21>>
```

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in \LaTeX is executed twice, but we need them when defining options and

babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

35 \def\bbl@tempa#1{%
36   \long\def\bbl@trim##1##2{%
37     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38   \def\bbl@trim@c{%
39     \ifx\bbl@trim@a\@sptoken

```

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


```

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

```

\bb1@ifunset To check if a macro is defined, we create a new macro, which does the same as `\@ifundefined`. However, in an ϵ -tex engine, it is based on `\ifcsname`, which is more efficient, and do not waste memory.

```

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

```

\bb1@ifblank A tool from url, by Donald Arseneau, which tests if a string is empty or space. The companion macros tests if a macro is defined with some ‘real’ value, ie, not `\relax` and not empty,

```

68 \def\bb1@ifblank#1{%
69 \bb1@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bb1@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bb1@ifset#1#2#3{%
72 \bb1@ifunset{#1}{#3}{\bb1@exp{\bb1@ifblank{#1}}{#3}{#2}}}

```

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

```

73 \def\bb1@forkv#1#2{%
74 \def\bb1@kvcmd##1##2##3{#2}%
75 \bb1@kvnext#1,\@nil,}
76 \def\bb1@kvnext#1,{%
77 \ifx\@nil#1\relax\else
78 \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
79 \expandafter\bb1@kvnext
80 \fi}
81 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
82 \bb1@trim@def\bb1@forkv@a{#1}%
83 \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@forkv@a}}{#2}{#4}}

```

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

```

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

```

\bbl@replace

```

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

```

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

```

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

```

Two further tools. \bbl@samestring first expand its arguments and then compare their expansion

(sanitized, so that the catcodes do not matter). `\bbl@engine` takes the following values: 0 is pdf \TeX , 1 is \LaTeX , 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 L^AT_EX 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 (L^AT_EX, `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\bbl@trace[1]{\message{^^J[ #1 ]}}%
196    \let\bbl@debug\@firstofone
197    \ifx\directlua\@undefined\else
198      \directlua{ Babel = Babel or {}
199                Babel.debug = true }%
200    \fi}
201 {\providecommand\bbl@trace[1]{}%
202  \let\bbl@debug\@gobble
203  \ifx\directlua\@undefined\else
204    \directlua{ Babel = Babel or {}
205              Babel.debug = false }%
206  \fi}
207 <<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\bbbl@add\csname#1.ldf-h@@k\endcsname}%

```

If the format created a list of loaded languages (in `\bbbl@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\bbbl@languages\undefined\else
274   \begingroup
275     \catcode`\^^I=12
276     \@ifpackagewith{babel}{showlanguages}{%
277       \begingroup
278         \def\bbbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
279         \wlog{<*languages>}%
280         \bbbl@languages
281         \wlog{</languages>}%
282       \endgroup}{%
283     \endgroup
284     \def\bbbl@elt#1#2#3#4{%
285       \ifnum#2=\z@
286         \gdef\bbbl@nulllanguage{#1}%
287         \def\bbbl@elt##1##2##3##4{%
288           \fi}%
289       \bbbl@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 \bbbl@trace{Defining option 'base'}
292 \@ifpackagewith{babel}{base}{%
293   \let\bbbl@onlyswitch\@empty
294   \let\bbbl@provide@locale\relax
295   \input babel.def
296   \let\bbbl@onlyswitch\@undefined
297   \ifx\directlua\@undefined
298     \DeclareOption*{\bbbl@patterns{\CurrentOption}}%
299   \else
300     \input luababel.def
301     \DeclareOption*{\bbbl@patterns@lua{\CurrentOption}}%
302   \fi
303   \DeclareOption{base}{}%
304   \DeclareOption{showlanguages}{}%
305   \ProcessOptions
306   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
307   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
308   \global\let\@ifl@ter@\@ifl@ter
309   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
310   \endinput}{%
311 % \end{macrocode}
312 %
313 % \subsection{\texttt{key=value} options and other general option}
314 %
315 %   The following macros extract language modifiers, and only real
316 %   package options are kept in the option list. Modifiers are saved

```

```

317%   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
318%   no modifiers have been given, the former is |\relax|. How
319%   modifiers are handled are left to language styles; they can use
320%   |\in@|, loop them with |\@for| or load |keyval|, for example.
321%
322%   \begin{macrocode}
323 \bbl@trace{key=value and another general options}
324 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
325 \def\bbl@tempb#1.#2{% Remove trailing dot
326   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
327 \def\bbl@tempd#1.#2@nnil{% TODO. Refactor lists?
328   \ifx\@empty#2%
329     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
330   \else
331     \in@{,provide,},{, #1,}%
332     \ifin@
333       \edef\bbl@tempc{%
334         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
335     \else
336       \in@{=}{ #1}%
337       \ifin@
338         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
339       \else
340         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
341         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
342       \fi
343     \fi
344   \fi}
345 \let\bbl@tempc\@empty
346 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
347 \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

```

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

```

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

```

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

key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

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

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

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

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

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

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

```
392 \ProcessOptions*
```

7.4 Conditional loading of shorthands

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

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

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

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

```
407   \def\bbl@ifshorthand#1{%
```



```

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

```

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

```

414 \edef\bbl@opt@shorthands{%
415 \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

447 \input babel.def

```

7.5 Cross referencing macros

The \TeX book states:

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

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

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

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

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

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

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

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

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

```

482   \fi}
483 \fi

\ref    The same holds for the macro \ref that references a label and \pageref to reference a page. We
\pageref 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@redefineroast\ref#1{%
487     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
488   \bbl@redefineroast\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 BiBTeX 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 \bible is needed we define \bible in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bible. This new definition is then activated.

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

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

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

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

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

When a document is run for the first time, no .aux file is available, and \bible 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@activestrue\org@bibitem{#1}\@safe@activesfalse}
525 \else
526   \let\org@nocite\nocite
527   \let\org@citex\citex
528   \let\org@bible\bible
529   \let\org@bibitem\@bibitem
530 \fi
```

7.6 Marks

\markright Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \markright and \markboth somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used. We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

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

```

546      \bbl@ifblank{#1}%
547      {\org@markright{}}%
548      {\toks@{#1}}%
549      \bbl@exp{%
550          \org@markright{\protect\foreignlanguage{\language}%
551              {\protect\bbl@restore@actives\the\toks@}}}%
\markboth The definition of \markboth is equivalent to that of \markright, except that we need two token
\@mkboth 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 ifbbl@single, end \IfBabelLayout

```

7.7 Preventing clashes with other packages

7.7.1 ifthen

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

```

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

```

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

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

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

```

571 \bbl@trace{Preventing clashes with other packages}
572 \bbl@xin@{R}\bbl@opt@safe
573 \ifin@
574 \AtBeginDocument{%
575     \@ifpackageloaded{ifthen}{%
576         \bbl@redefine@long\ifthenelse#1#2#3{%

```

```

577 \let\bbl@temp@pref\pageref
578 \let\pageref\org@pageref
579 \let\bbl@temp@ref\ref
580 \let\ref\org@ref
581 \@safe@activetrue
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 to prevent problems when an active character ends up in the argument of `\vref`. The same needs to happen for `\vrefpagemum`.

```

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

```

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

```

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

```

7.7.3 hline

`\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 \TeX .

```

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]^{}
630         \string\DeclareFontFamily{#1}{#2}{}^{}
631         \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^{}
632         \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^{}
633         \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^{}
634         \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^{}
635         \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^{}
636         \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^{}
637         \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^{}
638         \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^{}
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 have been loaded by traversing `\@filelist` to search for `<enc>enc.def`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

```

\ensureascii
643 \bb1@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@{,#1enc.def,},{, \@filelist,}}%
654     \fi}%
655   \ifin@ % if a text non-ascii has been loaded
656     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
657     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
658     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
659     \def\bbl@tempb#1\@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
660     \def\bbl@tempc#1ENC.DEF#2\@{\%
661       \ifx\@empty#2\else
662         \bbl@ifunset{T@#1}%
663         {}%
664         {\bbl@xin@{,#1,},{, \BabelNonASCII, \BabelNonText,}%
665         \ifin@
666           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
667           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
668         \else
669           \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
670         \fi}%
671       \fi}%
672   \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@ de mas??
673   \bbl@xin@{,\cf@encoding,},{, \BabelNonASCII, \BabelNonText,}%
674   \ifin@%
675     \edef\ensureascii#1{%
676       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
677   \fi
678 \fi}

```

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

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

```

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

```

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

```

680 \AtBeginDocument{%
681   \ifpackageloaded{fontspec}%
682     {\xdef\latinencoding{%
683       \ifx\UTFfencname\@undefined
684         EU\ifcase\bbl@engine\or2\or1\fi
685       \else
686         \UTFfencname
687       \fi}}%
688   {\gdef\latinencoding{OT1}}%
689   \ifx\cf@encoding\bbl@t@one
690     \xdef\latinencoding{\bbl@t@one}%

```



```

691 \else
692 \ifx\@fontenc@load@list\@undefined
693 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
694 \else
695 \def\@elt#1{,#1,}%
696 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
697 \let\@elt\relax
698 \bbl@xin@{,T1,}\bbl@tempa
699 \ifin@
700 \xdef\latinencoding{\bbl@t@one}%
701 \fi
702 \fi
703 \fi}}

```

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

```

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

```

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

```

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

```

7.9 Basic bidi support

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

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

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

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

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

```

712 \ifodd\bbl@engine
713 \def\bbl@activate@preotf{%
714 \let\bbl@activate@preotf\relax % only once
715 \directlua{

```

```

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

```

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

```

751 \bbl@trace{Loading basic (internal) bidi support}
752 \ifodd\bbl@engine
753   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
754     \let\bbl@beforeforeign\leavevmode
755     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
756     \RequirePackage{luatexbase}
757     \bbl@activate@preotf
758     \directlua{
759       require('babel-data-bidi.lua')
760       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
761         require('babel-bidi-basic.lua')
762       \or
763         require('babel-bidi-basic-r.lua')
764     }
765     % TODO - to locale_props, not as separate attribute
766     \newattribute\bbl@attr@dir
767     % TODO. I don't like it, hackish:
768     \bbl@exp{\output{\bodydir\pagedir\the\output}}
769     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
770   \fi\fi
771 \else

```

```

772 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
773   \bbl@error
774   {The bidi method 'basic' is available only in\%
775     luatex. I'll continue with 'bidi=default', so\%
776     expect wrong results}%
777   {See the manual for further details.}%
778   \let\bbl@beforeforeign\leavevmode
779   \AtEndOfPackage{%
780     \EnableBabelHook{babel-bidi}%
781     \bbl@xebidipar}
782 \fi\fi
783 \def\bbl@loadxebidi#1{%
784   \ifx\RTLfootnotetext\@undefined
785     \AtEndOfPackage{%
786       \EnableBabelHook{babel-bidi}%
787       \ifx\fontspec\@undefined
788         \bbl@loadfontspec % bidi needs fontspec
789       \fi
790       \usepackage#1{bidi}}%
791   \fi}
792 \ifnum\bbl@bidimode>200
793   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
794     \bbl@tentative{bidi=bidi}
795     \bbl@loadxebidi{}
796   \or
797     \bbl@loadxebidi{[rldocument]}
798   \or
799     \bbl@loadxebidi{}
800   \fi
801 \fi
802 \fi
803 \ifnum\bbl@bidimode=\@ne
804   \let\bbl@beforeforeign\leavevmode
805   \ifodd\bbl@engine
806     \newattribute\bbl@attr@dir
807     \bbl@exp{\output{\bodydir\pagedir\the\output}}%
808   \fi
809   \AtEndOfPackage{%
810     \EnableBabelHook{babel-bidi}%
811     \ifodd\bbl@engine\else
812       \bbl@xebidipar
813     \fi}
814 \fi

```

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

```

815 \bbl@trace{Macros to switch the text direction}
816 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
817 \def\bbl@rscripts{% TODO. Base on codes ??
818   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
819   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
820   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
821   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
822   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
823   Old South Arabian,}%
824 \def\bbl@provide@dirs#1{%
825   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
826   \ifin@
827     \global\bbl@csarg\chardef{wdir@#1}\@ne

```

```

828 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
829 \ifin@
830 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
831 \fi
832 \else
833 \global\bbl@csarg\chardef{wdir@#1}\z@
834 \fi
835 \ifodd\bbl@engine
836 \bbl@csarg\ifcase{wdir@#1}%
837 \directlua{ Babel.locale_props[\the\localeid].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@sys@\languagename}{\bbl@provide@sys{\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 r1
868 \ifcase#3\relax
869 \ifcase\bbl@getluadir{#1}\relax\else
870 #2 TLT\relax
871 \fi
872 \else
873 \ifcase\bbl@getluadir{#1}\relax
874 #2 TRT\relax
875 \fi
876 \fi}
877 \def\bbl@texmdir#1{%
878 \bbl@setluadir{tex}\texmdir{#1}%
879 \chardef\bbl@thetexmdir#1\relax
880 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
881 \def\bbl@pardir#1{%
882 \bbl@setluadir{par}\pardir{#1}%
883 \chardef\bbl@thepardir#1\relax}
884 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}

```

```

885 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
886 \def\bbl@dirparastext{\paddir\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\bbl@mathboxdir{%
890   \ifcase\bbl@thetextdir\relax
891     \everyhbox{\textdir TLT\relax}%
892   \else
893     \everyhbox{\textdir TRT\relax}%
894   \fi}
895 \frozen@everymath\expandafter{%
896   \expandafter\bbl@mathboxdir\the\frozen@everymath}
897 \frozen@everydisplay\expandafter{%
898   \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
899 \else % pdftex=0, xetex=2
900   \newcount\bbl@dirlevel
901   \chardef\bbl@thetextdir\z@
902   \chardef\bbl@thepaddir\z@
903   \def\bbl@textdir#1{%
904     \ifcase#1\relax
905       \chardef\bbl@thetextdir\z@
906       \bbl@textdir@i\beginL\endL
907     \else
908       \chardef\bbl@thetextdir\@ne
909       \bbl@textdir@i\beginR\endR
910     \fi}
911   \def\bbl@textdir@i#1#2{%
912     \ifhmode
913       \ifnum\currentgrouplevel>\z@
914         \ifnum\currentgrouplevel=\bbl@dirlevel
915           \bbl@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 % 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@paddir#1{\chardef\bbl@thepaddir#1\relax}
940   \let\bbl@bodydir\@gobble
941   \let\bbl@pagedir\@gobble
942   \def\bbl@dirparastext{\chardef\bbl@thepaddir\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@xeeverypar{%
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@xeeverypar\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@pardir#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@vdef\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{\bbl@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 L^AT_EX, some of it is for the L^AT_EX case only.

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

8.1 Tools

```

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 $\text{\TeX} 2_{\epsilon}$ style file. So, In $\text{\TeX} 2.09$ 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\LdfInit\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{\<version>}
1204 \def\bbl@date{\<date>}
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 \MakeXXXcase. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note l@ is encapsulated, so that its case does not change.

```

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}{\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{\edef#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@bcpcase#2\@empty\@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@bcpcase#2\@empty\@empty\@@\bbl@tempb
1258 \bbl@bcpcase#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 % Still necessary. TODO
1287 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1288 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1289 \ifbbl@bcpallowed
1290 \expandafter\ifx\csname date\language\endcsname\relax
1291 \expandafter
1292 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1293 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1294 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1295 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1296 \expandafter\ifx\csname date\language\endcsname\relax
1297 \let\bbl@initoload\bbl@bcp
1298 \bbl@exp{\@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1299 \let\bbl@initoload\relax
1300 \fi
1301 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
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@hymapset=\@cclv\let\bbl@hymapset\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@files
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@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
1421 % set name
1422 \edef\language{#1}%
1423 \bbl@fixname\language
1424 % TODO. name@map must be here?

```



```

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{\lccode##1=##2\relax}%
1489 \ifnum\bbl@hymapsel>4\else
1490 \csname\language\name @bbl@hyphenmap\endcsname
1491 \fi
1492 \chardef\bbl@opt@hyphenmap\z@
1493 \else
1494 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1495 \csname\language\name @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=\@cc1v\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   \begingroup
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\BabelText\@firstofone
```

```

1549 \foreign@language{#1}%
1550 \bbl@usehooks{foreign*}{}%
1551 \bbl@dirparastext
1552 \BabelText{#2}% Still in vertical mode!
1553 {\par}%
1554 \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`.

```

1555 \def\foreign@language#1{%
1556 % set name
1557 \edef\language#1}%
1558 \ifbbl@usedategroup
1559 \bbl@add\bbl@select@opts{,date,}%
1560 \bbl@usedategroupfalse
1561 \fi
1562 \bbl@fixname\language
1563 % TODO. name@map here?
1564 \bbl@provide@locale
1565 \bbl@iflanguage\language{%
1566 \expandafter\ifx\csname date\language\endcsname\relax
1567 \bbl@warning % TODO - why a warning, not an error?
1568 {Unknown language `#1'. Either you have\\%
1569 misspelled its name, it has not been installed,\\%
1570 or you requested it in a previous run. Fix its name,\\%
1571 install it or just rerun the file, respectively. In\\%
1572 some cases, you may need to remove the aux file.\\%
1573 I'll proceed, but expect wrong results.\\%
1574 Reported}%
1575 \fi
1576 % set type
1577 \let\bbl@select@type\@ne
1578 \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.

```

1579 \let\bbl@hyphlist\@empty
1580 \let\bbl@hyphenation@\relax
1581 \let\bbl@pttnlist\@empty
1582 \let\bbl@patterns@\relax
1583 \let\bbl@hymapsel=\@cclv
1584 \def\bbl@patterns#1{%
1585 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1586 \csname l@#1\endcsname
1587 \edef\bbl@tempa{#1}%
1588 \else
1589 \csname l@#1:\f@encoding\endcsname
1590 \edef\bbl@tempa{#1:\f@encoding}%
1591 \fi
1592 \@expandtwoargs\bbl@usehooks{patterns}{\bbl@tempa}}%
1593 % > luatex

```

```

1594 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1595   \beginngroup
1596     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1597     \ifin@else
1598       \@expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
1599     \hyphenation{%
1600       \bbl@hyphenation@
1601       \@ifundefined{bbl@hyphenation@#1}%
1602         \@empty
1603         {\space\csname bbl@hyphenation@#1\endcsname}}%
1604     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1605   \fi
1606   \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*`.

```

1607 \def\hyphenrules#1{%
1608   \edef\bbl@tempf{#1}%
1609   \bbl@fixname\bbl@tempf
1610   \bbl@iflanguage\bbl@tempf{%
1611     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1612     \ifx\languageshorthands\@undefined\else
1613       \languageshorthands{none}%
1614     \fi
1615     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1616       \set@hyphenmins\tw@\thr@\relax
1617     \else
1618       \expandafter\expandafter\expandafter\set@hyphenmins
1619       \csname\bbl@tempf hyphenmins\endcsname\relax
1620     \fi}}
1621 \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.

```

1622 \def\providehyphenmins#1#2{%
1623   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1624     \@namedef{#1hyphenmins}{#2}%
1625   \fi}

```

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

```

1626 \def\set@hyphenmins#1#2{%
1627   \lefthyphenmin#1\relax
1628   \righthyphenmin#2\relax}

```

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

```

1629 \ifx\ProvidesFile\@undefined
1630   \def\ProvidesLanguage#1[#2 #3 #4]{%
1631     \wlog{Language: #1 #4 #3 <#2>}%
1632   }
1633 \else
1634   \def\ProvidesLanguage#1{%

```

```

1635 \begingroup
1636 \catcode`\ 10 %
1637 \@makeother\/%
1638 \@ifnextchar[%]
1639 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1640 \def\@provideslanguage#1[#2]{%
1641 \wlog{Language: #1 #2}%
1642 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1643 \endgroup}
1644 \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`.

```
1645 \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.

```
1646 \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’:

```

1647 \providecommand\setlocale{%
1648 \bbl@error
1649 {Not yet available}%
1650 {Find an armchair, sit down and wait}}
1651 \let\uselocale\setlocale
1652 \let\locale\setlocale
1653 \let\selectlocale\setlocale
1654 \let\localename\setlocale
1655 \let\textlocale\setlocale
1656 \let\textlanguage\setlocale
1657 \let\languagegetext\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 $\TeX 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.
Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

1658 \edef\bbl@nulllanguage{\string\language=0}
1659 \ifx\PackageError\undefined % TODO. Move to Plain
1660 \def\bbl@error#1#2{%
1661 \begingroup
1662 \newlinechar=`^^J
1663 \def\{^^J(babel) }%
1664 \errhelp{#2}\errmessage{\{#1}%
1665 \endgroup}
1666 \def\bbl@warning#1{%
1667 \begingroup
1668 \newlinechar=`^^J
1669 \def\{^^J(babel) }%
1670 \message{\{#1}%

```

```

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

Here ended switch.def.

1715 \ifx\directlua\@undefined\else
1716 \ifx\bbl@luapatterns\@undefined
1717 \input luababel.def
1718 \fi
1719 \fi
1720 <<Basic macros>>
1721 \bbl@trace{Compatibility with language.def}
1722 \ifx\bbl@languages\@undefined
1723 \ifx\directlua\@undefined
1724 \openin1 = language.def % TODO. Remove hardcoded number
1725 \ifeof1
1726 \closein1
1727 \message{I couldn't find the file language.def}

```

```

1728 \else
1729 \closein1
1730 \begingroup
1731 \def\addlanguage#1#2#3#4#5{%
1732 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1733 \global\expandafter\let\csname l@#1\expandafter\endcsname
1734 \csname lang@#1\endcsname
1735 \fi}%
1736 \def\uselanguage#1{%
1737 \input language.def
1738 \endgroup
1739 \fi
1740 \fi
1741 \chardef\l@english\z@
1742 \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.

```

1743 \def\addto#1#2{%
1744 \ifx#1\@undefined
1745 \def#1{#2}%
1746 \else
1747 \ifx#1\relax
1748 \def#1{#2}%
1749 \else
1750 {\toks@\expandafter{#1#2}%
1751 \xdef#1{the\toks@}}%
1752 \fi
1753 \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?

```

1754 \def\bbl@withactive#1#2{%
1755 \begingroup
1756 \lccode`~=`#2\relax
1757 \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`.

```

1758 \def\bbl@redefine#1{%
1759 \edef\bbl@tempa{\bbl@stripslash#1}%
1760 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1761 \expandafter\def\csname\bbl@tempa\endcsname{
1762 \@onlypreamble\bbl@redefine

```

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

```

1763 \def\bbl@redefine@long#1{%
1764 \edef\bbl@tempa{\bbl@stripslash#1}%
1765 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1766 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1767 \@onlypreamble\bbl@redefine@long

```


`\bbl@redefinero bust` 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`.

```
1768 \def\bbl@redefinero bust#1{%
1769   \edef\bbl@tempa{\bbl@stripslash#1}%
1770   \bbl@ifunset{\bbl@tempa\space}%
1771   {\expandafter\let\csname org\bbl@tempa\endcsname#1%
1772    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1773   {\bbl@exp{\let\<org\bbl@tempa>\<\bbl@tempa\space>}}}%
1774   \@namedef{\bbl@tempa\space}}
1775 \@onlypreamble\bbl@redefinero bust
```

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.

```
1776 \bbl@trace{Hooks}
1777 \newcommand\AddBabelHook[3][{}]{%
1778   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1779   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1780   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1781   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1782   {\bbl@csarg\bbl@add{\ev@#3@#1}{\bbl@elth{#2}}}%
1783   {\bbl@csarg\let{\ev@#2@#3@#1}\relax}%
1784   \bbl@csarg\newcommand{\ev@#2@#3@#1}{\bbl@tempb}}
1785 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{\hk@#1}\@firstofone}
1786 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{\hk@#1}\@gobble}
1787 \def\bbl@usehooks#1#2{%
1788   \def\bbl@elth##1{%
1789     \bbl@cs{\hk@##1}{\bbl@cs{\ev@##1@#1}{#2}}%
1790     \bbl@cs{\ev@#1@}%
1791     \ifx\language\@undefined\else % Test required for Plain (?)
1792       \def\bbl@elth##1{%
1793         \bbl@cs{\hk@##1}{\bbl@cl{\ev@##1@#1}{#2}}%
1794         \bbl@cl{\ev@#1}%
1795       \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).

```
1796 \def\bbl@evargs{,% <- don't delete this comma
1797   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1798   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1799   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1800   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1801   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.

```

1802 \bbl@trace{Defining babelensure}
1803 \newcommand\babelensure[2][{}]{% TODO - revise test files
1804   \AddBabelHook{babel-ensure}{afterextras}{%
1805     \ifcase\bbl@select@type
1806       \bbl@cl{e}%
1807     \fi}%
1808 \begingroup
1809   \let\bbl@ens@include\@empty
1810   \let\bbl@ens@exclude\@empty
1811   \def\bbl@ens@fontenc{\relax}%
1812   \def\bbl@tempb##1{%
1813     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1814   \edef\bbl@tempa{\bbl@tempb##1\@empty}%
1815   \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1816   \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1817   \def\bbl@tempc{\bbl@ensure}%
1818   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1819     \expandafter{\bbl@ens@include}}%
1820   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1821     \expandafter{\bbl@ens@exclude}}%
1822   \toks@\expandafter{\bbl@tempc}%
1823   \bbl@exp{%
1824 \endgroup
1825 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1826 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1827 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1828   \ifx##1\undefined % 3.32 - Don't assume the macro exists
1829     \edef##1{\noexpand\bbl@nocaption
1830       {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1831   \fi
1832   \ifx##1\@empty\else
1833     \in@{##1}{#2}%
1834     \ifin@ \else
1835       \bbl@ifunset{bbl@ensure@\language\name}%
1836       {\bbl@exp{%
1837         \\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1838           \\foreignlanguage{\language\name}%
1839           {\ifx\relax#3\else
1840             \\fontencoding{#3}\\selectfont
1841             \fi
1842             #####1}}}%
1843       }%
1844       \toks@\expandafter{##1}%
1845       \edef##1{%
1846         \bbl@csarg\noexpand{ensure@\language\name}%
1847         {\the\toks@}}%
1848     \fi
1849     \expandafter\bbl@tempb
1850   \fi}%
1851 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1852 \def\bbl@tempa##1{% elt for include list
1853   \ifx##1\@empty\else
1854     \bbl@csarg\in@{ensure@\language\name\expandafter}\expandafter{##1}%
1855     \ifin@ \else
1856       \bbl@tempb##1\@empty
1857     \fi
1858     \expandafter\bbl@tempa
1859   \fi}%
1860 \bbl@tempa#1\@empty}

```

```

1861 \def\bbl@captionslist{%
1862 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1863 \contentsname\listfigurename\listtablename\indexname\figurename
1864 \tablename\partname\enclname\ccname\headtoname\pagename\seenname
1865 \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 @-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.

```

1866 \bbl@trace{Macros for setting language files up}
1867 \def\bbl@ldfinit{%
1868 \let\bbl@screset\@empty
1869 \let\BabelStrings\bbl@opt@string
1870 \let\BabelOptions\@empty
1871 \let\BabelLanguages\relax
1872 \ifx\originalTeX\@undefined
1873 \let\originalTeX\@empty
1874 \else
1875 \originalTeX
1876 \fi}
1877 \def\LdfInit#1#2{%
1878 \chardef\atcatcode=\catcode`\@
1879 \catcode`\@=11\relax
1880 \chardef\eqcatcode=\catcode`\=
1881 \catcode`\==12\relax
1882 \expandafter\if\expandafter\@backslashchar
1883 \expandafter\@car\string#2\@nil
1884 \ifx#2\@undefined\else
1885 \ldf@quit{#1}%
1886 \fi
1887 \else
1888 \expandafter\ifx\csname#2\endcsname\relax\else
1889 \ldf@quit{#1}%
1890 \fi
1891 \fi
1892 \bbl@ldfinit}

```

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

```

1893 \def\ldf@quit#1{%
1894 \expandafter\main@language\expandafter{#1}%
1895 \catcode`\@=\atcatcode \let\atcatcode\relax

```

```

1896 \catcode`\==\eqcatcode \let\eqcatcode\relax
1897 \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.

```

1898 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1899 \bbl@afterlang
1900 \let\bbl@afterlang\relax
1901 \let\BabelModifiers\relax
1902 \let\bbl@screset\relax}%
1903 \def\ldf@finish#1{%
1904 \ifx\loadlocalcfg@undefined\else % For LaTeX 209
1905 \loadlocalcfg{#1}%
1906 \fi
1907 \bbl@afterldf{#1}%
1908 \expandafter\main@language\expandafter{#1}%
1909 \catcode`\@=\atcatcode \let\atcatcode\relax
1910 \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 `ltxex`.

```

1911 \@onlypreamble\LdfInit
1912 \@onlypreamble\ldf@quit
1913 \@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.

```

1914 \def\main@language#1{%
1915 \def\bbl@main@language{#1}%
1916 \let\language\bbl@main@language % TODO. Set localename
1917 \bbl@id@assign
1918 \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`.

```

1919 \def\bbl@beforestart{%
1920 \bbl@usehooks{beforestart}}}%
1921 \global\let\bbl@beforestart\relax}
1922 \AtBeginDocument{%
1923 \@nameuse{bbl@beforestart}%
1924 \if@filesw
1925 \providecommand\babel@aux[2]{}%
1926 \immediate\write\@mainaux{%
1927 \string\providecommand\string\babel@aux[2]{}%
1928 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1929 \fi
1930 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1931 \ifbbl@single % must go after the line above.
1932 \renewcommand\selectlanguage[1]{}%
1933 \renewcommand\foreignlanguage[2]{#2}%
1934 \global\let\babel@aux\@gobbletwo % Also as flag
1935 \fi
1936 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1937 \def\select@language#1{%
1938   \ifcase\bbbl@select@type
1939     \bbbl@ifsamestring\language#1\{\select@language{#1}}%
1940   \else
1941     \select@language{#1}%
1942   \fi}

```

9.5 Shorthands

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

```

1943 \bbbl@trace{Shorhands}
1944 \def\bbbl@add@special#1{% 1:a macro like "\", \?, etc.
1945   \bbbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1946   \bbbl@ifunset{@sanitize}\{\bbbl@add\@sanitize{\@makeother#1}}%
1947   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1948     \begingroup
1949       \catcode`#1\active
1950       \nfss@catcodes
1951       \ifnum\catcode`#1=\active
1952         \endgroup
1953       \bbbl@add\nfss@catcodes{\@makeother#1}%
1954     \else
1955       \endgroup
1956     \fi
1957   \fi}

```

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

```

1958 \def\bbbl@remove@special#1{%
1959   \begingroup
1960     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1961       \else\noexpand##1\noexpand##2\fi}%
1962     \def\do{\x\do}%
1963     \def\@makeother{\x\@makeother}%
1964     \edef\x{\endgroup
1965       \def\noexpand\dospecials{\dospecials}%
1966       \expandafter\ifx\curname @sanitize\endcsname\relax\else
1967         \def\noexpand\@sanitize{\@sanitize}%
1968       \fi}%
1969     \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 `\bbbl@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).

```

1970 \def\bbl@active@def#1#2#3#4{%
1971   \namedef{#3#1}{%
1972     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1973       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1974     \else
1975       \bbl@afterfi\csname#2@sh@#1\endcsname
1976     \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.

```

1977   \long\@namedef{#3@arg#1}##1{%
1978     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1979       \bbl@afterelse\csname#4#1\endcsname##1%
1980     \else
1981       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1982     \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.

```

1983 \def\@initiate@active@char#1{%
1984   \bbl@ifunset{active@char\string#1}%
1985   {\bbl@withactive
1986     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1987   {}}

```

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`).

```

1988 \def\@initiate@active@char#1#2#3{%
1989   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1990   \ifx#1\@undefined
1991     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1992   \else
1993     \bbl@csarg\let{oridef@#2}#1%
1994     \bbl@csarg\edef{oridef@#2}{%
1995       \let\noexpand#1%
1996       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1997   \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*”).

```

1998   \ifx#1#3\relax
1999     \expandafter\let\csname normal@char#2\endcsname#3%
2000   \else
2001     \bbl@info{Making #2 an active character}%
2002     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2003     \@namedef{normal@char#2}{%
2004       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
2005   \else

```

```

2006     \@namedef{normal@char#2}{#3}%
2007     \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).

```

2008     \bbl@restoreactive{#2}%
2009     \AtBeginDocument{%
2010         \catcode`#2\active
2011         \if@filesw
2012             \immediate\write\@mainaux{\catcode`\string#2\active}%
2013         \fi}%
2014     \expandafter\bbl@add@special\csname#2\endcsname
2015     \catcode`#2\active
2016     \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}).

```

2017     \let\bbl@tempa\@firstoftwo
2018     \if\string^#2%
2019         \def\bbl@tempa{\noexpand\textormath}%
2020     \else
2021         \ifx\bbl@mathnormal\@undefined\else
2022             \let\bbl@tempa\bbl@mathnormal
2023         \fi
2024     \fi
2025     \expandafter\edef\csname active@char#2\endcsname{%
2026         \bbl@tempa
2027         {\noexpand\if@safe@actives
2028             \noexpand\expandafter
2029             \expandafter\noexpand\csname normal@char#2\endcsname
2030             \noexpand\else
2031                 \noexpand\expandafter
2032                 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2033             \noexpand\fi}%
2034         {\expandafter\noexpand\csname normal@char#2\endcsname}}}%
2035     \bbl@csarg\edef{doactive#2}{%
2036         \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!).

```

2037     \bbl@csarg\edef{active@#2}{%
2038         \noexpand\active@prefix\noexpand#1%
2039         \expandafter\noexpand\csname active@char#2\endcsname}%
2040     \bbl@csarg\edef{normal@#2}{%
2041         \noexpand\active@prefix\noexpand#1%
2042         \expandafter\noexpand\csname normal@char#2\endcsname}%
2043     \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.

```
2044 \bbl@active@def#2\user@group{user@active}{language@active}%
2045 \bbl@active@def#2\language@group{language@active}{system@active}%
2046 \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.

```
2047 \expandafter\edef\csname\user@group @sh#2@@\endcsname
2048 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2049 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
2050 {\expandafter\noexpand\csname user@active#2\endcsname}%
```

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

```
2051 \if\string'#2%
2052 \let\prim@s\bbl@prim@s
2053 \let\active@math@prime#1%
2054 \fi
2055 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}
```

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

```
2056 <<*More package options>> ≡
2057 \DeclareOption{math=active}{}
2058 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
2059 <</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.

```
2060 \@ifpackagewith{babel}{KeepShorthandsActive}%
2061 {\let\bbl@restoreactive\@gobble}%
2062 {\def\bbl@restoreactive#1{%
2063 \bbl@exp{%
2064 \\\AfterBabelLanguage\\CurrentOption
2065 {\catcode`#1=\the\catcode`#1\relax}%
2066 \\\AtEndOfPackage
2067 {\catcode`#1=\the\catcode`#1\relax}}}%
2068 \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.

```
2069 \def\bbl@sh@select#1#2{%
2070 \expandafter\ifx\csname#1@sh#2@sel\endcsname\relax
2071 \bbl@afterelse\bbl@scndcs
2072 \else
2073 \bbl@afterfi\csname#1@sh#2@sel\endcsname
2074 \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.

```

2075 \begingroup
2076 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2077 {\gdef\active@prefix#1{%
2078   \ifx\protect\@typeset@protect
2079   \else
2080     \ifx\protect\@unexpandable@protect
2081       \noexpand#1%
2082     \else
2083       \protect#1%
2084     \fi
2085     \expandafter\@gobble
2086   \fi}}
2087 {\gdef\active@prefix#1{%
2088   \ifincsname
2089     \string#1%
2090     \expandafter\@gobble
2091   \else
2092     \ifx\protect\@typeset@protect
2093     \else
2094       \ifx\protect\@unexpandable@protect
2095         \noexpand#1%
2096       \else
2097         \protect#1%
2098       \fi
2099       \expandafter\expandafter\expandafter\@gobble
2100     \fi
2101   \fi}}
2102 \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>`.

```

2103 \newif\if@safe@actives
2104 \@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.

```

2105 \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`.

```

2106 \def\bbl@activate#1{%
2107   \bbl@withactive{\expandafter\let\expandafter}#1%
2108   \csname bbl@active@\string#1\endcsname}
2109 \def\bbl@deactivate#1{%
2110   \bbl@withactive{\expandafter\let\expandafter}#1%
2111   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2112 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2113 \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.

```

2114 \def\babel@texpdf#1#2#3#4{%
2115   \ifx\texorpdfstring\@undefined
2116     \textormath{#1}{#2}%
2117   \else
2118     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2119     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2120   \fi}
2121 %
2122 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2123 \def\@decl@short#1#2#3\@nil#4{%
2124   \def\bbl@tempa{#3}%
2125   \ifx\bbl@tempa\@empty
2126     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2127     \bbl@ifunset{#1@sh@\string#2@}{}%
2128     {\def\bbl@tempa{#4}%
2129      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2130      \else
2131        \bbl@info
2132        {Redefining #1 shorthand \string#2\\%
2133         in language \CurrentOption}%
2134      \fi}%
2135     \@namedef{#1@sh@\string#2@}{#4}%
2136   \else
2137     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2138     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2139     {\def\bbl@tempa{#4}%
2140      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2141      \else
2142        \bbl@info
2143        {Redefining #1 shorthand \string#2\string#3\\%
2144         in language \CurrentOption}%
2145      \fi}%
2146     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2147   \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.

```

2148 \def\textormath{%
2149   \ifmmode
2150     \expandafter\@secondoftwo
2151   \else
2152     \expandafter\@firstoftwo
2153   \fi}

```

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

```

2154 \def\user@group{user}
2155 \def\language@group{english} % TODO. I don't like defaults
2156 \def\system@group{system}

\useshorthands This is the user level macro. It initializes and activates the character for use as a shorthand character
(ie, it's active in the preamble). Languages can deactivate shorthands, so a starred version is also
provided which activates them always after the language has been switched.

2157 \def\useshorthands{%
2158   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2159 \def\bbl@usesh@s#1{%
2160   \bbl@usesh@x
2161   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2162   {#1}}
2163 \def\bbl@usesh@x#1#2{%
2164   \bbl@ifshorthand{#2}%
2165   {\def\user@group{user}%
2166     \initiate@active@char{#2}%
2167     #1%
2168     \bbl@activate{#2}}%
2169   {\bbl@error
2170     {Cannot declare a shorthand turned off (\string#2)}
2171     {Sorry, but you cannot use shorthands which have been\\%
2172       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.

2173 \def\user@language@group{user@\language@group}
2174 \def\bbl@set@user@generic#1#2{%
2175   \bbl@ifunset{user@generic@active#1}%
2176   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2177     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2178     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2179       \expandafter\noexpand\csname normal@char#1\endcsname}%
2180     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2181       \expandafter\noexpand\csname user@active#1\endcsname}}%
2182   \@empty}
2183 \newcommand\defineshorthand[3][user]{%
2184   \edef\bbl@tempa{\zap@space#1 \@empty}%
2185   \bbl@for\bbl@tempb\bbl@tempa{%
2186     \if*\expandafter\@car\bbl@tempb\@nil
2187       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2188       \@expandtwoargs
2189       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2190     \fi
2191     \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].

2192 \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".

2193 \def\aliasshorthand#1#2{%

```

```

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

```

\@notshorthand

```

2211 \def\@notshorthand#1{%
2212   \bbl@error{%
2213     The character '\string #1' should be made a shorthand character;\\%
2214     add the command \string\usesshorthands\string{#1\string} to
2215     the preamble.\\%
2216     I will ignore your instruction}%
2217   {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.

```

2218 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2219 \DeclareRobustCommand*\shorthandoff{%
2220   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2221 \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.

```

2222 \def\bbl@switch@sh#1#2{%
2223   \ifx#2\@nnil\else
2224     \bbl@ifunset{\bbl@active@\string#2}%
2225     {\bbl@error
2226       {I cannot switch '\string#2' on or off--not a shorthand}%
2227       {This character is not a shorthand. Maybe you made\\%
2228         a typing mistake? I will ignore your instruction}}}%
2229     {\ifcase#1%
2230       \catcode`#212\relax
2231     \or
2232       \catcode`#2\active
2233     \or
2234       \csname bbl@oricat@\string#2\endcsname
2235       \csname bbl@oridef@\string#2\endcsname
2236     \fi}%
2237     \bbl@afterfi\bbl@switch@sh#1%
2238   \fi}

```

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

```

2239 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2240 \def\bbl@putsh#1{%
2241   \bbl@ifunset{\bbl@active@\string#1}%
2242   {\bbl@putsh@i#1\@empty\@nnil}%
2243   {\csname bbl@active@\string#1\endcsname}}
2244 \def\bbl@putsh@i#1#2\@nnil{%
2245   \csname\language@group @sh@\string#1@%
2246   \ifx\@empty#2\else\string#2\fi\endcsname}
2247 \ifx\bbl@opt@shorthands\@nnil\else
2248   \let\bbl@s@initiate@active@char\initiate@active@char
2249   \def\initiate@active@char#1{%
2250     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2251   \let\bbl@s@switch@sh\bbl@switch@sh
2252   \def\bbl@switch@sh#1#2{%
2253     \ifx#2\@nnil\else
2254       \bbl@afterfi
2255       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2256       \fi}
2257   \let\bbl@s@activate\bbl@activate
2258   \def\bbl@activate#1{%
2259     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2260   \let\bbl@s@deactivate\bbl@deactivate
2261   \def\bbl@deactivate#1{%
2262     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2263 \fi

```

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

```

2264 \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.

```

2265 \def\bbl@prim@s{%
2266   \prime\futurelet\@let@token\bbl@pr@m@s}
2267 \def\bbl@if@primes#1#2{%
2268   \ifx#1\@let@token
2269     \expandafter\@firstoftwo
2270   \else\ifx#2\@let@token
2271     \bbl@afterelse\expandafter\@firstoftwo
2272   \else
2273     \bbl@afterfi\expandafter\@secondoftwo
2274   \fi\fi}
2275 \begingroup
2276 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2277 \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
2278 \lowercase{%
2279   \gdef\bbl@pr@m@s{%
2280     \bbl@if@primes"'"%
2281     \pr@@@s
2282     {\bbl@if@primes*\^pr@@@t\egroup}}}
2283 \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).

```
2284 \initiate@active@char{~}
2285 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2286 \bbl@activate{~}
```

\OT1dpos 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.

```
2287 \expandafter\def\csname OT1dpos\endcsname{127}
2288 \expandafter\def\csname T1dpos\endcsname{4}
```

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

```
2289 \ifx\f@encoding\undefined
2290 \def\f@encoding{OT1}
2291 \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.

```
2292 \bbl@trace{Language attributes}
2293 \newcommand\languageattribute[2]{%
2294 \def\bbl@tempc{#1}%
2295 \bbl@fixname\bbl@tempc
2296 \bbl@iflanguage\bbl@tempc{%
2297 \bbl@vforeach{#2}{%
```

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

```
2298 \ifx\bbl@known@attrs\undefined
2299 \in@false
2300 \else
2301 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
2302 \fi
2303 \ifin@
2304 \bbl@warning{%
2305 You have more than once selected the attribute '##1'\%
2306 for language #1. Reported}%
2307 \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.

```
2308 \bbl@exp{%
2309 \bbl@add@list\bbl@known@attrs{\bbl@tempc-##1}}%
2310 \edef\bbl@tempa{\bbl@tempc-##1}%
2311 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2312 {\csname\bbl@tempc @attr@##1\endcsname}%
2313 {\@attrerr{\bbl@tempc}{##1}}%
2314 \fi}}
2315 \@onlypreamble\languageattribute
```

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

```
2316 \newcommand*{\@attrerr}[2]{%
2317   \bbl@error
2318   {The attribute #2 is unknown for language #1.}%
2319   {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}`.

```
2320 \def\bbl@declare@ttribute#1#2#3{%
2321   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2322   \ifin@
2323     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2324   \fi
2325   \bbl@add@list\bbl@attributes{#1-#2}%
2326   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

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

```
2327 \def\bbl@ifattributeset#1#2#3#4{%
2328   \ifx\bbl@known@attribs\@undefined
2329     \in@false
2330   \else
2331     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2332   \fi
2333   \ifin@
2334     \bbl@afterelse#3%
2335   \else
2336     \bbl@afterfi#4%
2337   \fi}
```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```
2338 \def\bbl@ifknown@ttrib#1#2{%
2339   \let\bbl@tempa\@secondoftwo
2340   \bbl@loopx\bbl@tempb{#2}{%
2341     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2342     \ifin@
2343       \let\bbl@tempa\@firstoftwo
2344     \else
2345       \fi}%
2346   \bbl@tempa}
```

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

```
2347 \def\bbl@clear@ttribs{%
2348   \ifx\bbl@attributes\@undefined\else
2349     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2350       \expandafter\bbl@clear@ttrib\bbl@tempa.
2351     }%
2352     \let\bbl@attributes\@undefined
```

```

2353 \fi}
2354 \def\bbl@clear@ttrib#1-#2.{%
2355 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2356 \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`

```

2357 \bbl@trace{Macros for saving definitions}
2358 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2359 \newcount\babel@savecnt
2360 \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.

```

2361 \def\babel@save#1{%
2362 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2363 \toks@\expandafter{\originalTeX\let#1=}%
2364 \bbl@exp{%
2365 \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2366 \advance\babel@savecnt@ne}
2367 \def\babel@savevariable#1{%
2368 \toks@\expandafter{\originalTeX #1=}%
2369 \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.

```

2370 \def\bbl@frenchspacing{%
2371 \ifnum\the\sfcodes\.=\@m
2372 \let\bbl@nonfrenchspacing\relax
2373 \else
2374 \frenchspacing
2375 \let\bbl@nonfrenchspacing\nonfrenchspacing
2376 \fi}
2377 \let\bbl@nonfrenchspacing\nonfrenchspacing
2378 \let\bbl@elt\relax
2379 \edef\bbl@fs@chars{%
2380 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2381 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2382 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}

```

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

9.8 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the macros `\text{<tag>}` and `\<tag>`. Definitions are first expanded so that they don't contain `\csname` but the actual macro.

```

2383 \bbl@trace{Short tags}
2384 \def\babeltags#1{%
2385   \edef\bbl@tempa{\zap@space#1 \@empty}%
2386   \def\bbl@tempb##1=##2\@{
2387     \edef\bbl@tempc{%
2388       \noexpand\newcommand
2389       \expandafter\noexpand\csname ##1\endcsname{%
2390         \noexpand\protect
2391         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2392       \noexpand\newcommand
2393       \expandafter\noexpand\csname text##1\endcsname{%
2394         \noexpand\foreignlanguage{##2}}
2395       \bbl@tempc}%
2396   \bbl@for\bbl@tempa\bbl@tempa{%
2397     \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.

```

2398 \bbl@trace{Hyphens}
2399 \@onlypreamble\babelhyphenation
2400 \AtEndOfPackage{%
2401   \newcommand\babelhyphenation[2][\@empty]{%
2402     \ifx\bbl@hyphenation@relax
2403       \let\bbl@hyphenation@\@empty
2404     \fi
2405     \ifx\bbl@hyphlist\@empty\else
2406       \bbl@warning{%
2407         You must not intermingle \string\selectlanguage\space and\%
2408         \string\babelhyphenation\space or some exceptions will not\%
2409         be taken into account. Reported}%
2410     \fi
2411     \ifx\@empty#1%
2412       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2413     \else
2414       \bbl@vforeach{#1}{%
2415         \def\bbl@tempa{##1}%
2416         \bbl@fixname\bbl@tempa
2417         \bbl@iflanguage\bbl@tempa{%
2418           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2419             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2420             {}%
2421             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2422             #2}}}%
2423     \fi}

```

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

```

2424 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}

```

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

```

2425 \def\bbl@t@one{T1}
2426 \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`.

```

2427 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2428 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2429 \def\bbl@hyphen{%
2430   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2431 \def\bbl@hyphen@i#1#2{%
2432   \bbl@ifunset{bbl@hy@#1#2@empty}%
2433   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2434   {\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.

```

2435 \def\bbl@usehyphen#1{%
2436   \leavevmode
2437   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2438   \nobreak\hskip\z@skip}
2439 \def\bbl@@usehyphen#1{%
2440   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2441 \def\bbl@hyphenchar{%
2442   \ifnum\hyphenchar\font=\m@ne
2443     \babellnullhyphen
2444   \else
2445     \char\hyphenchar\font
2446   \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.

```

2447 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2448 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2449 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2450 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2451 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2452 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2453 \def\bbl@hy@repeat{%
2454   \bbl@usehyphen{%
2455     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2456 \def\bbl@hy@@repeat{%
2457   \bbl@usehyphen{%
2458     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2459 \def\bbl@hy@empty{\hskip\z@skip}
2460 \def\bbl@hy@@empty{\discretionary{}{}{}}

```

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

```

2461 \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.

```
2462 \bbl@trace{Multiencoding strings}
2463 \def\bbl@tglobal#1{\global\let#1#1}
2464 \def\bbl@recatcode#1{% TODO. Used only once?
2465   \@tempcnta="7F
2466   \def\bbl@tempa{%
2467     \ifnum\@tempcnta>"FF\else
2468       \catcode\@tempcnta=#1\relax
2469       \advance\@tempcnta\@ne
2470       \expandafter\bbl@tempa
2471     \fi}%
2472   \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).

```
2473 \ifpackagewith{babel}{nocase}%
2474   {\let\bbl@patchuclc\relax}%
2475   {\def\bbl@patchuclc{%
2476     \global\let\bbl@patchuclc\relax
2477     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2478     \gdef\bbl@uclc##1{%
2479       \let\bbl@encoded\bbl@encoded@uclc
2480       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2481       {##1}%
2482       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2483         \csname\language @bbl@uclc\endcsname}%
2484       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2485     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2486     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2487 \<<*More package options>> ≡
2488 \DeclareOption{nocase}{}
2489 \<</More package options>>
```

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

```
2490 \<<*More package options>> ≡
2491 \let\bbl@opt@strings\@nnil % accept strings=value
2492 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2493 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2494 \def\BabelStringsDefault{generic}
2495 \<</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.

```

2496 \@onlypreamble\StartBabelCommands
2497 \def\StartBabelCommands{%
2498   \begingroup
2499   \bbl@recatcode{11}%
2500   <<Macros local to BabelCommands>>
2501   \def\bbl@provstring##1##2{%
2502     \providecommand##1{##2}%
2503     \bbl@tglobal##1}%
2504   \global\let\bbl@scafter\@empty
2505   \let\StartBabelCommands\bbl@startcmds
2506   \ifx\BabelLanguages\relax
2507     \let\BabelLanguages\CurrentOption
2508   \fi
2509   \begingroup
2510   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2511   \StartBabelCommands}
2512 \def\bbl@startcmds{%
2513   \ifx\bbl@screset\@nnil\else
2514     \bbl@usehooks{stopcommands}{}%
2515   \fi
2516   \endgroup
2517   \begingroup
2518   \@ifstar
2519     {\ifx\bbl@opt@strings\@nnil
2520       \let\bbl@opt@strings\BabelStringsDefault
2521     \fi
2522     \bbl@startcmds@i}%
2523   \bbl@startcmds@i}
2524 \def\bbl@startcmds@i#1#2{%
2525   \edef\bbl@L{\zap@space#1 \@empty}%
2526   \edef\bbl@G{\zap@space#2 \@empty}%
2527   \bbl@startcmds@ii}
2528 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

2529 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2530   \let\SetString\@gobbletwo
2531   \let\bbl@stringdef\@gobbletwo
2532   \let\AfterBabelCommands\@gobble
2533   \ifx\@empty#1%
2534     \def\bbl@sc@label{generic}%
2535     \def\bbl@encstring##1##2{%
2536       \ProvideTextCommandDefault##1{##2}%
2537       \bbl@tglobal##1%
2538       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2539     \let\bbl@sctest\in@true
2540   \else
2541     \let\bbl@sc@charset\space % <- zapped below

```

```

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

```

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

```

2588 \def\bbl@forlang#1##2{%
2589 \bbl@for#1\bbl@L{%
2590 \bbl@xin@{, #1,}{, \BabelLanguages,}%
2591 \ifin@#2\relax\fi}}

```

```

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

```

2614 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2615   \bbl@forlang\bbl@tempa{%
2616     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2617     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2618     {\bbl@exp{%
2619       \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2620     }%
2621     \def\BabelString{#2}%
2622     \bbl@usehooks{stringprocess}{}%
2623     \expandafter\bbl@stringdef
2624     \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.

```

2625 \ifx\bbl@opt@strings\relax
2626   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2627   \bbl@patchuclc
2628   \let\bbl@encoded\relax
2629   \def\bbl@encoded@uclc#1{%
2630     \@inmathwarn#1%
2631     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2632       \expandafter\ifx\csname ?\string#1\endcsname\relax
2633         \TextSymbolUnavailable#1%
2634       \else
2635         \csname ?\string#1\endcsname
2636       \fi
2637     \else
2638       \csname\cf@encoding\string#1\endcsname

```

```

2639     \fi}
2640 \else
2641   \def\bbl@scset#1#2{\def#1{#2}}
2642 \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.

```

2643 <<*Macros local to BabelCommands>> ≡
2644 \def\SetStringLoop##1##2{%
2645   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2646   \count@\z@
2647   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2648     \advance\count@\@ne
2649     \toks@\expandafter{\bbl@tempa}%
2650     \bbl@exp{%
2651       \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2652       \count@=\the\count@\relax}}}%
2653 <</Macros local to BabelCommands>>

```

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

```

2654 \def\bbl@aftercmds#1{%
2655   \toks@\expandafter{\bbl@scafter#1}%
2656   \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.

```

2657 <<*Macros local to BabelCommands>> ≡
2658 \newcommand\SetCase[3][]{%
2659   \bbl@patchuclc
2660   \bbl@forlang\bbl@tempa{%
2661     \expandafter\bbl@encstring
2662     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2663     \expandafter\bbl@encstring
2664     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2665     \expandafter\bbl@encstring
2666     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2667 <</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.

```

2668 <<*Macros local to BabelCommands>> ≡
2669 \newcommand\SetHyphenMap[1]{%
2670   \bbl@forlang\bbl@tempa{%
2671     \expandafter\bbl@stringdef
2672     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2673 <</Macros local to BabelCommands>>

```

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

```

2674 \newcommand\BabelLower[2]{% one to one.
2675   \ifnum\lccode#1=#2\else
2676     \babel@savevariable{\lccode#1}%
2677     \lccode#1=#2\relax
2678   \fi}
2679 \newcommand\BabelLowerMM[4]{% many-to-many
2680   \@tempcnta=#1\relax

```

```

2681 \@tempcntb=#4\relax
2682 \def\bbl@tempa{%
2683   \ifnum\@tempcnta>#2\else
2684     \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2685     \advance\@tempcnta#3\relax
2686     \advance\@tempcntb#3\relax
2687     \expandafter\bbl@tempa
2688   \fi}%
2689 \bbl@tempa}
2690 \newcommand\BabelLowerMO[4]{% many-to-one
2691   \@tempcnta=#1\relax
2692   \def\bbl@tempa{%
2693     \ifnum\@tempcnta>#2\else
2694       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2695       \advance\@tempcnta#3
2696       \expandafter\bbl@tempa
2697     \fi}%
2698   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2699 <<{*More package options}>> ≡
2700 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2701 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2702 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2703 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@}
2704 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2705 <</More package options>>

```

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

```

2706 \AtEndOfPackage{%
2707   \ifx\bbl@opt@hyphenmap\undefined
2708     \bbl@xin@{,}{\bbl@language@opts}%
2709     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2710   \fi}

```

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

```

2711 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2712   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2713 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2714   \bbl@trim@def\bbl@tempa{#2}%
2715   \bbl@xin@{.template}{\bbl@tempa}%
2716   \ifin@
2717     \bbl@ini@captions@template{#3}{#1}%
2718   \else
2719     \edef\bbl@tempd{%
2720       \expandafter\expandafter\expandafter
2721       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2722     \bbl@xin@
2723       {\expandafter\string\csname #2name\endcsname}%
2724     {\bbl@tempd}%
2725     \ifin@ % Renew caption
2726       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2727     \ifin@
2728       \bbl@exp{%
2729         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2730         {\bbl@scset\<#2name>\<#1#2name>}%
2731       }%

```



```

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

```

9.11 Macros common to a number of languages

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

```

2766 \bbl@trace{Macros related to glyphs}
2767 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2768 \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2769 \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.

```

2770 \def\save@sf@q#1{\leavevmode
2771 \begingroup
2772 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2773 \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.

```
2774 \ProvideTextCommand{\quotedblbase}{OT1}{%
2775   \save@sf@q{\set@low@box{\textquotedblright\}%
2776     \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.

```
2777 \ProvideTextCommandDefault{\quotedblbase}{%
2778   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2779 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2780   \save@sf@q{\set@low@box{\textquoteright\}%
2781     \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.

```
2782 \ProvideTextCommandDefault{\quotesinglbase}{%
2783   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2784 \ProvideTextCommand{\guillemetleft}{OT1}{%
2785   \ifmmode
2786     \ll
2787   \else
2788     \save@sf@q{\nobreak
2789       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2790   \fi}
2791 \ProvideTextCommand{\guillemetright}{OT1}{%
2792   \ifmmode
2793     \gg
2794   \else
2795     \save@sf@q{\nobreak
2796       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2797   \fi}
2798 \ProvideTextCommand{\guillemotleft}{OT1}{%
2799   \ifmmode
2800     \ll
2801   \else
2802     \save@sf@q{\nobreak
2803       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2804   \fi}
2805 \ProvideTextCommand{\guillemotright}{OT1}{%
2806   \ifmmode
2807     \gg
2808   \else
2809     \save@sf@q{\nobreak
2810       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2811   \fi}
```

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

```
2812 \ProvideTextCommandDefault{\guillemetleft}{%
2813   \UseTextSymbol{OT1}{\guillemetleft}}
2814 \ProvideTextCommandDefault{\guillemetright}{%
2815   \UseTextSymbol{OT1}{\guillemetright}}
2816 \ProvideTextCommandDefault{\guillemotleft}{%
2817   \UseTextSymbol{OT1}{\guillemotleft}}
2818 \ProvideTextCommandDefault{\guillemotright}{%
2819   \UseTextSymbol{OT1}{\guillemotright}}
```

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

`\guilsinglright`

```

2820 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2821   \ifmmode
2822     <%
2823   \else
2824     \save@sf@q{\nobreak
2825       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2826   \fi}
2827 \ProvideTextCommand{\guilsinglright}{OT1}{%
2828   \ifmmode
2829     >%
2830   \else
2831     \save@sf@q{\nobreak
2832       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2833   \fi}

```

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

```

2834 \ProvideTextCommandDefault{\guilsinglleft}{%
2835   \UseTextSymbol{OT1}{\guilsinglleft}}
2836 \ProvideTextCommandDefault{\guilsinglright}{%
2837   \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.

```

2838 \DeclareTextCommand{\ij}{OT1}{%
2839   i\kern-0.02em\bbl@allowhyphens j}
2840 \DeclareTextCommand{\IJ}{OT1}{%
2841   I\kern-0.02em\bbl@allowhyphens J}
2842 \DeclareTextCommand{\ij}{T1}{\char188}
2843 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2844 \ProvideTextCommandDefault{\ij}{%
2845   \UseTextSymbol{OT1}{\ij}}
2846 \ProvideTextCommandDefault{\IJ}{%
2847   \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).

```

2848 \def\crrtic@{\hrule height0.1ex width0.3em}
2849 \def\crttic@{\hrule height0.1ex width0.33em}
2850 \def\ddj@{%
2851   \setbox0\hbox{d}\dimen@=\ht0
2852   \advance\dimen@1ex
2853   \dimen@.45\dimen@
2854   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2855   \advance\dimen@ii.5ex
2856   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\box{\crrtic@}}}}
2857 \def\DDJ@{%
2858   \setbox0\hbox{D}\dimen@=.55\ht0
2859   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2860   \advance\dimen@ii.15ex % correction for the dash position
2861   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2862   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@

```

```

2863 \leavevmode\rlap{\raise\dimen@hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2864 %
2865 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2866 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2867 \ProvideTextCommandDefault{\dj}{%
2868 \UseTextSymbol{OT1}{\dj}}
2869 \ProvideTextCommandDefault{\DJ}{%
2870 \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.

```

2871 \DeclareTextCommand{\SS}{OT1}{SS}
2872 \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 2873 \ProvideTextCommandDefault{\glq}{%
2874 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2875 \ProvideTextCommand{\grq}{T1}{%
2876 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2877 \ProvideTextCommand{\grq}{TU}{%
2878 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2879 \ProvideTextCommand{\grq}{OT1}{%
2880 \save@sf@q{\kern-.0125em
2881 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2882 \kern.07em\relax}}
2883 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

\glqq The ‘german’ double quotes.

```

\grqq 2884 \ProvideTextCommandDefault{\glqq}{%
2885 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2886 \ProvideTextCommand{\grqq}{T1}{%
2887 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2888 \ProvideTextCommand{\grqq}{TU}{%
2889 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2890 \ProvideTextCommand{\grqq}{OT1}{%
2891 \save@sf@q{\kern-.07em
2892 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2893 \kern.07em\relax}}
2894 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}

```

\flq The ‘french’ single guillemets.

```

\frq 2895 \ProvideTextCommandDefault{\flq}{%
2896 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2897 \ProvideTextCommandDefault{\frq}{%
2898 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

```

\flqq The 'french' double guillemets.
\frqq
2899 \ProvideTextCommandDefault{\flqq}{%
2900   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2901 \ProvideTextCommandDefault{\frqq}{%
2902   \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

9.12.4 Umlauts and tremas

The command `\` needs to have a different effect for different languages. For German for instance, the ‘umlaut’ should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

`\umlauthigh` To be able to provide both positions of `\` we provide two commands to switch the positioning, the default will be `\umlauthigh` (the normal positioning).

```

2903 \def\uumlauthigh{%
2904   \def\bbl@umlauta##1{\leavevmode\bgroup%
2905     \expandafter\accent\csname\fontencoding dqpos\endcsname
2906     ##1\bbl@allowhyphens\egroup}%
2907   \let\bbl@umlaute\bbl@umlauta}
2908 \def\uumlautlow{%
2909   \def\bbl@umlauta{\protect\lower@umlaut}}
2910 \def\uumlautelow{%
2911   \def\bbl@umlaute{\protect\lower@umlaut}}
2912 \umlauthigh

```

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

```

2913 \expandafter\ifx\csname U@D\endcsname\relax
2914   \csname newdimen\endcsname\U@D
2915 \fi

```

The following code fools \TeX 's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

2916 \def\lower@umlaut#1{%
2917   \leavevmode\bgroup
2918     \U@D 1ex%
2919     {\setbox\z@\hbox{%
2920       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2921       \dimen@ -.45ex\advance\dimen@\ht\z@
2922       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2923     \expandafter\accent\csname\fontencoding dqpos\endcsname
2924     \fontdimen5\font\U@D #1%
2925   \egroup}

```

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

```

2926 \AtBeginDocument{%

```

```

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

```

2938 \ifx\l@english\@undefined
2939 \chardef\l@english\z@
2940 \fi
2941 % The following is used to cancel rules in ini files (see Amharic).
2942 \ifx\l@babelnohyphens\@undefined
2943 \newlanguage\l@babelnohyphens
2944 \fi

```

9.13 Layout

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

```

2945 \bbl@trace{Bidi layout}
2946 \providecommand\IfBabelLayout[3]{#3}%
2947 \newcommand\BabelPatchSection[1]{%
2948   \@ifundefined{#1}{}{%
2949     \bbl@exp{\let\bbl@ss@#1>\<#1>}%
2950     \@namedef{#1}{%
2951       \ifstar\bbl@presec@#1}%
2952       {\@dblarg\bbl@presec@x{#1}}}%
2953 \def\bbl@presec@x#1[#2]#3{%
2954   \bbl@exp{%
2955     \\\select@language@x{\bbl@main@language}%
2956     \\\bbl@cs{sspre@#1}%
2957     \\\bbl@cs{ss@#1}%
2958     [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2959     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2960     \\\select@language@x{\languagename}}%
2961 \def\bbl@presec@#1#2{%
2962   \bbl@exp{%
2963     \\\select@language@x{\bbl@main@language}%
2964     \\\bbl@cs{sspre@#1}%
2965     \\\bbl@cs{ss@#1}*%
2966     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2967     \\\select@language@x{\languagename}}%
2968 \IfBabelLayout{sectioning}%
2969   {\BabelPatchSection{part}%
2970    \BabelPatchSection{chapter}%
2971    \BabelPatchSection{section}%
2972    \BabelPatchSection{subsection}%
2973    \BabelPatchSection{subsubsection}%
2974    \BabelPatchSection{paragraph}%
2975    \BabelPatchSection{subparagraph}%
2976    \def\babel@toc#1{%
2977      \select@language@x{\bbl@main@language}}}%

```

```

2978 \IfBabelLayout{captions}%
2979 {\BabelPatchSection{caption}}{}

```

9.14 Load engine specific macros

```

2980 \bbl@trace{Input engine specific macros}
2981 \ifcase\bbl@engine
2982 \input txtbabel.def
2983 \or
2984 \input luababel.def
2985 \or
2986 \input xebabel.def
2987 \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.

```

2988 \bbl@trace{Creating languages and reading ini files}
2989 \newcommand\babelprovide[2][]{%
2990 \let\bbl@savelangname\language
2991 \edef\bbl@savelocaleid{\the\localeid}%
2992 % Set name and locale id
2993 \edef\language{#2}%
2994 % \global\@namedef\bbl@lcname@#2}{#2}%
2995 \bbl@id@assign
2996 \let\bbl@KVP@captions\@nil
2997 \let\bbl@KVP@date\@nil
2998 \let\bbl@KVP@import\@nil
2999 \let\bbl@KVP@main\@nil
3000 \let\bbl@KVP@script\@nil
3001 \let\bbl@KVP@language\@nil
3002 \let\bbl@KVP@hyphenrules\@nil
3003 \let\bbl@KVP@mapfont\@nil
3004 \let\bbl@KVP@maparabic\@nil
3005 \let\bbl@KVP@mapdigits\@nil
3006 \let\bbl@KVP@intraspace\@nil
3007 \let\bbl@KVP@intrapenalty\@nil
3008 \let\bbl@KVP@onchar\@nil
3009 \let\bbl@KVP@alph\@nil
3010 \let\bbl@KVP@Alph\@nil
3011 \let\bbl@KVP@labels\@nil
3012 \bbl@csarg\let{KVP@labels*}\@nil
3013 \global\let\bbl@inidata\@empty
3014 \bbl@forkv{#1}{% TODO - error handling
3015 \in@{/{}}{##1}%
3016 \ifin@
3017 \bbl@renewinikey##1\@{##2}%
3018 \else
3019 \bbl@csarg\def{KVP@##1}{##2}%
3020 \fi}%
3021 % == init ==
3022 \ifx\bbl@screset\@undefined
3023 \bbl@ldfinit
3024 \fi
3025 % ==
3026 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3027 \bbl@ifunset{date#2}%

```

```

3028 {\let\bbkflag\@empty}% new
3029 {\ifx\bbk@KVP@hyphenrules\@nil\else
3030   \let\bbkflag\@empty
3031   \fi
3032   \ifx\bbk@KVP@import\@nil\else
3033     \let\bbkflag\@empty
3034     \fi}%
3035 % == import, captions ==
3036 \ifx\bbk@KVP@import\@nil\else
3037   \bbk@exp{\bbk@ifblank{\bbk@KVP@import}}%
3038   {\ifx\bbk@initload\relax
3039     \begingroup
3040       \def\BabelBeforeIni##1##2{\gdef\bbk@KVP@import{##1}\endinput}%
3041       \bbk@input@texini{##2}%
3042     \endgroup
3043     \else
3044       \xdef\bbk@KVP@import{\bbk@initload}%
3045     \fi}%
3046   {}%
3047 \fi
3048 \ifx\bbk@KVP@captions\@nil
3049   \let\bbk@KVP@captions\bbk@KVP@import
3050 \fi
3051 % Load ini
3052 \bbk@ifunset{date#2}%
3053   {\bbk@provide@new{##2}}%
3054   {\bbk@ifblank{##1}%
3055     }% With \bbk@load@basic below
3056   {\bbk@provide@renew{##2}}}%
3057 % Post tasks
3058 % -----
3059 % == ensure captions ==
3060 \ifx\bbk@KVP@captions\@nil\else
3061   \bbk@ifunset{\bbk@extracaps#2}%
3062     {\bbk@exp{\bbk@babelensure[exclude=\today]{##2}}}%
3063     {\toks@%
3064       \expandafter\expandafter\expandafter
3065       {\csname \bbk@extracaps#2\endcsname}%
3066       \bbk@exp{\bbk@babelensure[exclude=\today,include=\the\toks@]{##2}}%
3067       \bbk@ifunset{\bbk@ensure@\language}%
3068       {\bbk@exp{%
3069         \\\DeclareRobustCommand\<\bbk@ensure@\language>[1]{%
3070           \\\foreignlanguage{\language}%
3071             {###1}}}%
3072       }%
3073       \bbk@exp{%
3074         \\\bbk@toglobal\<\bbk@ensure@\language>%
3075         \\\bbk@toglobal\<\bbk@ensure@\language\space>%
3076       }%
3077 \fi
3078 % ==
3079 % At this point all parameters are defined if 'import'. Now we
3080 % execute some code depending on them. But what about if nothing was
3081 % imported? We just set the basic parameters, but still loading the
3082 % whole ini file.
3083 \bbk@load@basic{##2}%
3084 % == script, language ==
3085 % Override the values from ini or defines them
3086 \ifx\bbk@KVP@script\@nil\else
3087   \bbk@csarg\edef{sname#2}{\bbk@KVP@script}%
3088 \fi

```



```

3087 \ifx\bb1@KVP@language\@nil\else
3088   \bb1@csarg\edef{lname@#2}{\bb1@KVP@language}%
3089 \fi
3090 % == onchar ==
3091 \ifx\bb1@KVP@onchar\@nil\else
3092   \bb1@luahyphenate
3093   \directlua{
3094     if Babel.locale_mapped == nil then
3095       Babel.locale_mapped = true
3096       Babel.linebreaking.add_before(Babel.locale_map)
3097       Babel.loc_to_scr = {}
3098       Babel.chr_to_loc = Babel.chr_to_loc or {}
3099     end}%
3100   \bb1@xin@{ ids }{ \bb1@KVP@onchar\space}%
3101 \ifin@
3102   \ifx\bb1@starthyphens\@undefined % Needed if no explicit selection
3103     \AddBabelHook{babel-onchar}{beforestart}{\bb1@starthyphens}%
3104   \fi
3105   \bb1@exp{\bb1@add\bb1@starthyphens
3106     {\bb1@patterns@lua{\language}}}%
3107   % TODO - error/warning if no script
3108   \directlua{
3109     if Babel.script_blocks['\bb1@cl{sbc}'] then
3110       Babel.loc_to_scr[\the\localeid] =
3111         Babel.script_blocks['\bb1@cl{sbc}']
3112       Babel.locale_props[\the\localeid].lc = \the\localeid\space
3113       Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@\language}\space
3114     end
3115   }%
3116 \fi
3117 \bb1@xin@{ fonts }{ \bb1@KVP@onchar\space}%
3118 \ifin@
3119   \bb1@ifunset{bb1@lsys@\language}{\bb1@provide@lsys{\language}}}%
3120   \bb1@ifunset{bb1@wdir@\language}{\bb1@provide@dirs{\language}}}%
3121   \directlua{
3122     if Babel.script_blocks['\bb1@cl{sbc}'] then
3123       Babel.loc_to_scr[\the\localeid] =
3124         Babel.script_blocks['\bb1@cl{sbc}']
3125     end}%
3126   \ifx\bb1@mapselect\@undefined
3127     \AtBeginDocument{%
3128       \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
3129     {\selectfont}}%
3130   \def\bb1@mapselect{%
3131     \let\bb1@mapselect\relax
3132     \edef\bb1@prefontid{\fontid\font}}%
3133   \def\bb1@mapdir##1{%
3134     {\def\language{##1}%
3135       \let\bb1@ifrestoring\@firstoftwo % To avoid font warning
3136       \bb1@switchfont
3137       \directlua{
3138         Babel.locale_props[\the\csname bb1@id@##1\endcsname]%
3139           [\bb1@prefontid'] = \fontid\font\space}}}%
3140   \fi
3141   \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%
3142 \fi
3143 % TODO - catch non-valid values
3144 \fi
3145 % == mapfont ==

```

```

3146 % For bidi texts, to switch the font based on direction
3147 \ifx\bb1@KVP@mapfont\@nil\else
3148   \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}{}%
3149   {\bb1@error{Option '\bb1@KVP@mapfont' unknown for\%
3150     mapfont. Use 'direction'.%
3151     {See the manual for details.}}}%
3152   \bb1@ifunset{\bb1@lsys@\language}\bb1@provide@lsys{\language}{}%
3153   \bb1@ifunset{\bb1@wdir@\language}\bb1@provide@dirs{\language}{}%
3154   \ifx\bb1@mapselect\@undefined
3155     \AtBeginDocument{%
3156       \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}%
3157       {\selectfont}}%
3158     \def\bb1@mapselect{%
3159       \let\bb1@mapselect\relax
3160       \edef\bb1@prefontid{\fontid\font}%
3161       \def\bb1@mapdir##1{%
3162         {\def\language##1}%
3163         \let\bb1@ifrestoring\@firstoftwo % avoid font warning
3164         \bb1@switchfont
3165         \directlua{Babel.fontmap
3166           [\the\csname \bb1@wdir@##1\endcsname]%
3167           [\bb1@prefontid]=\fontid\font}}}%
3168     \fi
3169     \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%
3170   \fi
3171 % == Line breaking: intraspace, intrapenalty ==
3172 % For CJK, East Asian, Southeast Asian, if interspace in ini
3173 \ifx\bb1@KVP@intraspace\@nil\else % We can override the ini or set
3174   \bb1@csarg\edef{intsp@#2}{\bb1@KVP@intraspace}%
3175 \fi
3176 \bb1@provide@intraspace
3177 % == Line breaking: hyphenate.other.locale/.script==
3178 \ifx\bb1@lbkflag\@empty
3179   \bb1@ifunset{\bb1@hyotl@\language}{}%
3180   {\bb1@csarg\bb1@replace{hyotl@\language}{ }{,}%
3181     \bb1@startcommands*\language}%
3182   \bb1@csarg\bb1@foreach{hyotl@\language}{%
3183     \ifcase\bb1@engine
3184       \ifnum#1<257
3185         \SetHyphenMap{\BabelLower{##1}{##1}}%
3186       \fi
3187     \else
3188       \SetHyphenMap{\BabelLower{##1}{##1}}%
3189     \fi}%
3190   \bb1@endcommands}%
3191   \bb1@ifunset{\bb1@hyots@\language}{}%
3192   {\bb1@csarg\bb1@replace{hyots@\language}{ }{,}%
3193     \bb1@csarg\bb1@foreach{hyots@\language}{%
3194       \ifcase\bb1@engine
3195         \ifnum#1<257
3196           \global\lccode#1=##1\relax
3197         \fi
3198       \else
3199         \global\lccode#1=##1\relax
3200       \fi}}%
3201 \fi
3202 % == Counters: maparabic ==
3203 % Native digits, if provided in ini (TeX level, xe and lua)
3204 \ifcase\bb1@engine\else

```

```

3205 \bbl@ifunset{\bbl@dgnat@\language\name}{}%
3206 {\expandafter\ifx\csname bbl@dgnat@\language\name\endcsname\@empty\else
3207 \expandafter\expandafter\expandafter
3208 \bbl@setdigits\csname bbl@dgnat@\language\name\endcsname
3209 \ifx\bbl@KVP@maparabic\@nil\else
3210 \ifx\bbl@latinarabic\@undefined
3211 \expandafter\let\expandafter\@arabic
3212 \csname bbl@counter@\language\name\endcsname
3213 \else % ie, if layout=counters, which redefines \@arabic
3214 \expandafter\let\expandafter\bbl@latinarabic
3215 \csname bbl@counter@\language\name\endcsname
3216 \fi
3217 \fi
3218 \fi}%
3219 \fi
3220 % == Counters: mapdigits ==
3221 % Native digits (lua level).
3222 \ifodd\bbl@engine
3223 \ifx\bbl@KVP@mapdigits\@nil\else
3224 \bbl@ifunset{\bbl@dgnat@\language\name}{}%
3225 {\RequirePackage{luatexbase}%
3226 \bbl@activate@preotf
3227 \directlua{
3228 Babel = Babel or {} %% -> presets in luababel
3229 Babel.digits_mapped = true
3230 Babel.digits = Babel.digits or {}
3231 Babel.digits[\the\localeid] =
3232 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3233 if not Babel.numbers then
3234 function Babel.numbers(head)
3235 local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3236 local GLYPH = node.id'glyph'
3237 local inmath = false
3238 for item in node.traverse(head) do
3239 if not inmath and item.id == GLYPH then
3240 local temp = node.get_attribute(item, LOCALE)
3241 if Babel.digits[temp] then
3242 local chr = item.char
3243 if chr > 47 and chr < 58 then
3244 item.char = Babel.digits[temp][chr-47]
3245 end
3246 end
3247 elseif item.id == node.id'math' then
3248 inmath = (item.subtype == 0)
3249 end
3250 end
3251 return head
3252 end
3253 end
3254 }}%
3255 \fi
3256 \fi
3257 % == Counters: alph, Alph ==
3258 % What if extras<lang> contains a \babel@save\@alph? It won't be
3259 % restored correctly when exiting the language, so we ignore
3260 % this change with the \bbl@alph@saved trick.
3261 \ifx\bbl@KVP@alph\@nil\else
3262 \toks@\expandafter\expandafter\expandafter{%
3263 \csname extras\language\name\endcsname}%

```

```

3264 \bbl@exp{%
3265 \def\<extras\language>{%
3266 \let\\bbl@alph@savd\\@alph
3267 \the\toks@
3268 \let\\@alph\\bbl@alph@savd
3269 \\babel@save\\@alph
3270 \let\\@alph<bbl@cntr@\\bbl@KVP@alph @\language>}}%
3271 \fi
3272 \ifx\bbl@KVP@Alph@nil\else
3273 \toks@\expandafter\expandafter\expandafter{%
3274 \csname extras\language\endcsname}%
3275 \bbl@exp{%
3276 \def\<extras\language>{%
3277 \let\\bbl@Alph@savd\\@Alph
3278 \the\toks@
3279 \let\\@Alph\\bbl@Alph@savd
3280 \\babel@save\\@Alph
3281 \let\\@Alph<bbl@cntr@\\bbl@KVP@Alph @\language>}}%
3282 \fi
3283 % == require.babel in ini ==
3284 % To load or reload the babel-*.tex, if require.babel in ini
3285 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3286 \bbl@ifunset{bbl@rqtex@\language}}{%
3287 {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3288 \let\BabelBeforeIni@gobbletwo
3289 \chardef\atcatcode=\catcode`\@
3290 \catcode`\@=11\relax
3291 \bbl@input@texini{\bbl@cs{rqtex@\language}}%
3292 \catcode`\@=\atcatcode
3293 \let\atcatcode\relax
3294 \fi}%
3295 \fi
3296 % == main ==
3297 \ifx\bbl@KVP@main@nil % Restore only if not 'main'
3298 \let\language\bbl@savelangname
3299 \chardef\localeid\bbl@savelocaleid\relax
3300 \fi}

```

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

```

3301 \def\bbl@provide@new#1{%
3302 \@namedef{date#1}}{% marks lang exists - required by \StartBabelCommands
3303 \@namedef{extras#1}}{%
3304 \@namedef{noextras#1}}{%
3305 \bbl@startcommands*{#1}{captions}%
3306 \ifx\bbl@KVP@captions@nil % and also if import, implicit
3307 \def\bbl@tempb##1{% elt for \bbl@captionslist
3308 \ifx##1@empty\else
3309 \bbl@exp{%
3310 \\SetString\\##1{%
3311 \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3312 \expandafter\bbl@tempb
3313 \fi}%
3314 \expandafter\bbl@tempb\bbl@captionslist\@empty
3315 \else
3316 \ifx\bbl@initoload\relax
3317 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
3318 \else
3319 \bbl@read@ini{\bbl@initoload}2% % Same
3320 \fi

```

```

3321 \fi
3322 \StartBabelCommands*{#1}{date}%
3323 \ifx\bbbl@KVP@import\@nil
3324 \bbbl@exp{%
3325 \\\SetString\\today{\\bbbl@nocaption{today}{#1today}}}%
3326 \else
3327 \bbbl@savetoday
3328 \bbbl@savedate
3329 \fi
3330 \bbbl@endcommands
3331 \bbbl@load@basic{#1}%
3332 % == hyphenmins == (only if new)
3333 \bbbl@exp{%
3334 \gdef\<#1hyphenmins>{%
3335 {\bbbl@ifunset{bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
3336 {\bbbl@ifunset{bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}%
3337 % == hyphenrules ==
3338 \bbbl@provide@hyphens{#1}%
3339 % == frenchspacing == (only if new)
3340 \bbbl@ifunset{bbbl@frspc@#1}{}%
3341 {\edef\bbbl@tempa{\bbbl@cl{frspc}}}%
3342 \edef\bbbl@tempa{\expandafter\@car\bbbl@tempa\@nil}%
3343 \if u\bbbl@tempa % do nothing
3344 \else\if n\bbbl@tempa % non french
3345 \expandafter\bbbl@add\csname extras#1\endcsname{%
3346 \let\bbbl@elt\bbbl@fs@elt@i
3347 \bbbl@fs@chars}%
3348 \else\if y\bbbl@tempa % french
3349 \expandafter\bbbl@add\csname extras#1\endcsname{%
3350 \let\bbbl@elt\bbbl@fs@elt@ii
3351 \bbbl@fs@chars}%
3352 \fi\fi\fi}%
3353 %
3354 \ifx\bbbl@KVP@main\@nil\else
3355 \expandafter\main@language\expandafter{#1}%
3356 \fi}
3357 % A couple of macros used above, to avoid hashes #####...
3358 \def\bbbl@fs@elt@i#1#2#3{%
3359 \ifnum\sfcode`#1=#2\relax
3360 \babel@savevariable{\sfcode`#1}%
3361 \sfcode`#1=#3\relax
3362 \fi}%
3363 \def\bbbl@fs@elt@ii#1#2#3{%
3364 \ifnum\sfcode`#1=#3\relax
3365 \babel@savevariable{\sfcode`#1}%
3366 \sfcode`#1=#2\relax
3367 \fi}%
3368 %
3369 \def\bbbl@provide@renew#1{%
3370 \ifx\bbbl@KVP@captions\@nil\else
3371 \StartBabelCommands*{#1}{captions}%
3372 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
3373 \EndBabelCommands
3374 \fi
3375 \ifx\bbbl@KVP@import\@nil\else
3376 \StartBabelCommands*{#1}{date}%
3377 \bbbl@savetoday
3378 \bbbl@savedate
3379 \EndBabelCommands

```

```

3380 \fi
3381 % == hyphenrules ==
3382 \ifx\bbbl@lbkflag\@empty
3383 \bbbl@provide@hyphens{#1}%
3384 \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.

```

3385 \def\bbbl@load@basic#1{%
3386 \bbbl@ifunset{bbbl@inidata@\language}\relax
3387 {\getlocaleproperty\bbbl@tempa{\language}{identification/load.level}%
3388 \ifcase\bbbl@tempa
3389 \bbbl@csarg\let{lname@\language}\relax
3390 \fi}%
3391 \bbbl@ifunset{bbbl@lname@#1}%
3392 {\def\BabelBeforeIni##1##2{%
3393 \begingroup
3394 \let\bbbl@ini@captions@aux\@gobbletwo
3395 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3396 \bbbl@read@ini{##1}1%
3397 \ifx\bbbl@initoload\relax\endinput\fi
3398 \endgroup}%
3399 \begingroup % boxed, to avoid extra spaces:
3400 \ifx\bbbl@initoload\relax
3401 \bbbl@input@texini{#1}%
3402 \else
3403 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
3404 \fi
3405 \endgroup}%
3406 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3407 \def\bbbl@provide@hyphens#1{%
3408 \let\bbbl@tempa\relax
3409 \ifx\bbbl@KVP@hyphenrules\@nil\else
3410 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3411 \bbbl@foreach\bbbl@KVP@hyphenrules{%
3412 \ifx\bbbl@tempa\relax % if not yet found
3413 \bbbl@ifsamestring{##1}{+}%
3414 {\bbbl@exp{\addlanguage\<#1>}}}%
3415 {}%
3416 \bbbl@ifunset{l@#1}%
3417 {}%
3418 {\bbbl@exp{\let\bbbl@tempa\<#1>}}}%
3419 \fi}%
3420 \fi
3421 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
3422 \ifx\bbbl@KVP@import\@nil
3423 \ifx\bbbl@initoload\relax\else
3424 \bbbl@exp{% and hyphenrules is not empty
3425 \bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
3426 {}%
3427 {\let\bbbl@tempa\<#1>\bbbl@cl{hyphr}>}}%
3428 \fi
3429 \else % if importing
3430 \bbbl@exp{% and hyphenrules is not empty
3431 \bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
3432 {}%

```

```

3433      {\let\\bbl@tempa<l@bbl@c{hyphr}>}}%
3434      \fi
3435      \fi
3436      \bbl@ifunset{bbl@tempa}%          ie, relax or undefined
3437      {\bbl@ifunset{l@#1}%              no hyphenrules found - fallback
3438       {\bbl@exp{\\addialect<l@#1>\language}}%
3439       {}}%                             so, l@<lang> is ok - nothing to do
3440      {\bbl@exp{\\addialect<l@#1>bbl@tempa}}}% found in opt list or ini

```

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

```

3441 \def\bbl@input@texini#1{%
3442   \bbl@bsphack
3443   \bbl@exp{%
3444     \catcode`\\%=14 \catcode`\\=0
3445     \catcode`\\{=1 \catcode`\\}=2
3446     \lowercase{\InputIfFileExists{babel-#1.tex}}}%
3447     \catcode`\\%= \the\catcode`\% \relax
3448     \catcode`\\= \the\catcode`\% \relax
3449     \catcode`\\{= \the\catcode`\{ \relax
3450     \catcode`\\}= \the\catcode`\} \relax}%
3451   \bbl@esphack}

```

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

```

3452 \def\bbl@inline#1\bbl@inline{%
3453   \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@% ]
3454   \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}%
3455   \def\bbl@iniskip#1\@@{%          if starts with ;
3456   \def\bbl@inistore#1=#2\@@{%      full (default)
3457     \bbl@trim@def\bbl@tempa{#1}%
3458     \bbl@trim\toks@{#2}%
3459     \bbl@ifunset{bbl@KVP@bbl@section/\bbl@tempa}%
3460     {\bbl@exp{%
3461       \\g@addto@macro\\bbl@inidata{%
3462         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3463     {}}%
3464   \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
3465     \bbl@trim@def\bbl@tempa{#1}%
3466     \bbl@trim\toks@{#2}%
3467     \bbl@xin@{.identification.}{.\bbl@section.}%
3468     \ifin@
3469       \bbl@exp{\\g@addto@macro\\bbl@inidata{%
3470         \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3471     \fi}%

```

Now, the 'main loop', which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

```

3472 \ifx\bbl@readstream\undefined
3473   \csname newread\endcsname\bbl@readstream
3474   \fi
3475   \def\bbl@read@ini#1#2{%
3476     \openin\bbl@readstream=babel-#1.ini
3477     \ifeof\bbl@readstream
3478       \bbl@error

```

```

3479      {There is no ini file for the requested language\\%
3480      (#1). Perhaps you misspelled it or your installation\\%
3481      is not complete.}%
3482      {Fix the name or reinstall babel.}%
3483 \else
3484   % Store ini data in \bbl@inidata
3485   \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3486   \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3487   \bbl@info{Importing
3488           \ifcase#2font and identification \or basic \fi
3489           data for \language\\%
3490           from babel-#1.ini. Reported}%
3491   \ifnum#2=\z@
3492     \global\let\bbl@inidata\empty
3493     \let\bbl@inistore\bbl@inistore@min    % Remember it's local
3494   \fi
3495   \def\bbl@section{identification}%
3496   \bbl@exp{\bbl@inistore tag.ini=#1\\%}
3497   \bbl@inistore load.level=#2\\%
3498   \loop
3499   \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3500     \endlinechar\m@ne
3501     \read\bbl@readstream to \bbl@line
3502     \endlinechar\^^M
3503     \ifx\bbl@line\empty\else
3504       \expandafter\bbl@iniline\bbl@line\bbl@iniline
3505     \fi
3506   \repeat
3507   % Process stored data
3508   \bbl@csarg\xdef{lini@language}{#1}%
3509   \let\bbl@savestrings\empty
3510   \let\bbl@savetoday\empty
3511   \let\bbl@savestate\empty
3512   \def\bbl@elt##1##2##3{%
3513     \def\bbl@section{##1}%
3514     \in@{=date.}{##1}% Find a better place
3515     \ifin@
3516       \bbl@ini@calendar{##1}%
3517     \fi
3518     \global\bbl@csarg\let{bbl@KVP###1/##2}\relax
3519     \bbl@ifunset{bbl@inikv###1}{}%
3520     {\csname bbl@inikv###1\endcsname{##2}{##3}}}%
3521   \bbl@inidata
3522   % 'Export' data
3523   \bbl@ini@exports{#2}%
3524   \global\bbl@csarg\let{inidata@language}\bbl@inidata
3525   \global\let\bbl@inidata\empty
3526   \bbl@exp{\bbl@add@list\bbl@ini@loaded{language}}%
3527   \bbl@tglobal\bbl@ini@loaded
3528 \fi}

```

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

```

3529 \def\bbl@ini@calendar#1{%
3530   \lowercase{\def\bbl@tempa{=#1=}}%
3531   \bbl@replace\bbl@tempa{=date.gregorian}{}%
3532   \bbl@replace\bbl@tempa{=date.}{}%
3533   \in@{.licr=}{#1}%
3534   \ifin@
3535     \ifcase\bbl@engine

```



```

3536 \bbl@replace\bbl@tempa{.licr={}}%
3537 \else
3538 \let\bbl@tempa\relax
3539 \fi
3540 \fi
3541 \ifx\bbl@tempa\relax\else
3542 \bbl@replace\bbl@tempa{=}{}%
3543 \bbl@exp{%
3544 \def<\bbl@inikv@#1>####1####2{%
3545 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3546 \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).

```

3547 \def\bbl@renewinikey#1/#2\@#3{%
3548 \edef\bbl@tempa{\zap@space #1 \@empty}% section
3549 \edef\bbl@tempb{\zap@space #2 \@empty}% key
3550 \bbl@trim\toks@{#3}% value
3551 \bbl@exp{%
3552 \global\let<\bbl@KVP@\bbl@tempa/\bbl@tempb>\\@empty % just a flag
3553 \\\g@addto@macro\\bbl@inidata{%
3554 \\\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.

```

3555 \def\bbl@exportkey#1#2#3{%
3556 \bbl@ifunset{\bbl@kv@#2}%
3557 {\bbl@csarg\gdef{#1@\language}\@empty}%
3558 {\xexpandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
3559 \bbl@csarg\gdef{#1@\language}\@empty}%
3560 \else
3561 \bbl@exp{\global\let<\bbl@#1@\language>\<\bbl@kv@#2>}%
3562 \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.

```

3563 \def\bbl@iniwarning#1{%
3564 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3565 {\bbl@warning{%
3566 From babel-\bbl@cs{lini@\language}.ini:\%
3567 \bbl@cs{@kv@identification.warning#1}\%
3568 Reported }}}
3569 %
3570 \def\bbl@ini@exports#1{%
3571 % Identification always exported
3572 \bbl@iniwarning}%
3573 \ifcase\bbl@engine
3574 \bbl@iniwarning{.pdflatex}%
3575 \or
3576 \bbl@iniwarning{.lualatex}%
3577 \or
3578 \bbl@iniwarning{.xelatex}%
3579 \fi%
3580 \bbl@exportkey{elname}{identification.name.english}{}%
3581 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
3582 {\csname\bbl@elname@\language\endcsname}}%

```

```

3583 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
3584 \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
3585 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3586 \bbl@exportkey{esname}{identification.script.name}{}%
3587 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3588   {\csname bbl@esname\language\endcsname}}%
3589 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
3590 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3591 % Also maps bcp47 -> languagename
3592 \ifbbl@bcptoname
3593   \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
3594 \fi
3595 % Conditional
3596 \ifnum#1>\z@      % 0 = only info, 1, 2 = basic, (re)new
3597   \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
3598   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3599   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3600   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3601   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3602   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3603   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3604   \bbl@exportkey{intsp}{typography.intraspaces}{}%
3605   \bbl@exportkey{chrng}{characters.ranges}{}%
3606   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3607   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3608   \ifnum#1=\tw@    % only (re)new
3609     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3610     \bbl@toglobal\bbl@savetoday
3611     \bbl@toglobal\bbl@savestate
3612     \bbl@savestrings
3613   \fi
3614 \fi}

```

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

```

3615 \def\bbl@inikv#1#2{%      key=value
3616   \toks@{#2}%             This hides #'s from ini values
3617   \bbl@csarg\xdef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3618 \let\bbl@inikv@identification\bbl@inikv
3619 \let\bbl@inikv@typography\bbl@inikv
3620 \let\bbl@inikv@characters\bbl@inikv
3621 \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’.

```

3622 \def\bbl@inikv@counters#1#2{%
3623   \bbl@ifsamestring{#1}{digits}%
3624   {\bbl@error{The counter name 'digits' is reserved for mapping\\
3625     decimal digits}%
3626     {Use another name.}}%
3627   }%
3628   \def\bbl@tempc{#1}%
3629   \bbl@trim@def{\bbl@tempc*}{#2}%
3630   \in@{.1$}{#1$}%
3631   \ifin@
3632     \bbl@replace\bbl@tempc{.1}{}%
3633     \bbl@csarg\protected\xdef{cntr@\bbl@tempc @\language}{%

```

```

3634 \noexpand\bb1@alphanumeric{\bb1@tempc}}%
3635 \fi
3636 \in@{.F.}{#1}%
3637 \ifin@else\in@{.S.}{#1}\fi
3638 \ifin@
3639 \bb1@csarg\protected@xdef{cntr@#1@\language}\bb1@tempb*}%
3640 \else
3641 \toks@{}% Required by \bb1@buildifcase, which returns \bb1@tempa
3642 \expandafter\bb1@buildifcase\bb1@tempb* \ \ % Space after \
3643 \bb1@csarg{\global\expandafter\let}{cntr@#1@\language}\bb1@tempa
3644 \fi}

```

Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

```

3645 \ifcase\bb1@engine
3646 \bb1@csarg\def{inikv@captions.licr}#1#2{%
3647 \bb1@ini@captions@aux{#1}{#2}}
3648 \else
3649 \def\bb1@inikv@captions#1#2{%
3650 \bb1@ini@captions@aux{#1}{#2}}
3651 \fi

```

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

```

3652 \def\bb1@ini@captions@template#1#2{% string language tempa=capt-name
3653 \bb1@replace\bb1@tempa{.template}{}}%
3654 \def\bb1@toreplace{#1}{}%
3655 \bb1@replace\bb1@toreplace{[ ]}{\nobreakspace}}%
3656 \bb1@replace\bb1@toreplace{[ ]}{\csname}%
3657 \bb1@replace\bb1@toreplace{[ ]}{\csname the}%
3658 \bb1@replace\bb1@toreplace{[ ]}{name\endcsname}}%
3659 \bb1@replace\bb1@toreplace{[ ]}{\endcsname}}%
3660 \bb1@xin@{,\bb1@tempa,}{,chapter,appendix,part,}%
3661 \ifin@
3662 \@nameuse{bb1@patch\bb1@tempa}%
3663 \global\bb1@csarg\let{\bb1@tempa fmt@#2}\bb1@toreplace
3664 \fi
3665 \bb1@xin@{,\bb1@tempa,}{,figure,table,}%
3666 \ifin@
3667 \toks@\expandafter{\bb1@toreplace}%
3668 \bb1@exp{\gdef\<fnum@\bb1@tempa>{\the\toks@}}%
3669 \fi}
3670 \def\bb1@ini@captions@aux#1#2{%
3671 \bb1@trim@def\bb1@tempa{#1}%
3672 \bb1@xin@{.template}{\bb1@tempa}%
3673 \ifin@
3674 \bb1@ini@captions@template{#2}\language
3675 \else
3676 \bb1@ifblank{#2}%
3677 {\bb1@exp{%
3678 \toks@{\bb1@nocaption{\bb1@tempa}{\language\bb1@tempa name}}}%
3679 {\bb1@trim\toks@{#2}}}%
3680 \bb1@exp{%
3681 \bb1@add\bb1@savestrings{%
3682 \SetString\<\bb1@tempa name>{\the\toks@}}}%
3683 \toks@\expandafter{\bb1@captionslist}%
3684 \bb1@exp{\in@{\<\bb1@tempa name>}{\the\toks@}}%
3685 \ifin@else
3686 \bb1@exp{%

```

```

3687      \\bbl@add\<bbl@extracaps@\language\>\<bbl@tempa name\>%
3688      \\bbl@tglobal\<bbl@extracaps@\language\>%
3689      \fi
3690      \fi}

```

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

```

3691 \def\bbl@list@the{%
3692   part,chapter,section,subsection,subsubsection,paragraph,%
3693   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3694   table,page,footnote,mpfootnote,mpfn}
3695 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3696   \bbl@ifunset{bbl@map@#1@\language}%
3697     {\@nameuse{#1}}%
3698     {\@nameuse{bbl@map@#1@\language}}}%
3699 \def\bbl@inikv@labels#1#2{%
3700   \in@{.map}{#1}%
3701   \ifin@
3702     \ifx\bbl@KVP@labels\@nil\else
3703       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3704       \ifin@
3705         \def\bbl@tempc{#1}%
3706         \bbl@replace\bbl@tempc{.map}{}%
3707         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3708         \bbl@exp{%
3709           \gdef\<bbl@map@\bbl@tempc @\language\>%
3710             {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3711         \bbl@foreach\bbl@list@the{%
3712           \bbl@ifunset{the##1}{}%
3713           {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3714             \bbl@exp{%
3715               \\bbl@sreplace\<the##1>%
3716               {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3717             \\bbl@sreplace\<the##1>%
3718             {\<\empty @\bbl@tempc>\<c@##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3719           \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3720             \toks@\expandafter\expandafter\expandafter{%
3721               \csname the##1\endcsname}%
3722             \expandafter\def\csname the##1\endcsname{{\the\toks@}}%
3723             \fi}}%
3724       \fi
3725     \fi
3726   %
3727   \else
3728     %
3729     % The following code is still under study. You can test it and make
3730     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3731     % language dependent.
3732     \in@{enumerate.}{#1}%
3733     \ifin@
3734       \def\bbl@tempa{#1}%
3735       \bbl@replace\bbl@tempa{enumerate.}{}%
3736       \def\bbl@toreplace{#2}%
3737       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3738       \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3739       \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}}%
3740       \toks@\expandafter{\bbl@toreplace}%
3741       \bbl@exp{%
3742         \\bbl@add\<extras\language\>%
3743         \\babel@save\<labelenum\romannumeral\bbl@tempa>%

```

```

3744 \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3745 \\\bbl@toglobal<extras\language>%
3746 \fi
3747 \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.

```

3748 \def\bbl@chapttype{chapter}
3749 \ifx\@makechapterhead\@undefined
3750 \let\bbl@patchchapter\relax
3751 \else\ifx\thechapter\@undefined
3752 \let\bbl@patchchapter\relax
3753 \else\ifx\ps@headings\@undefined
3754 \let\bbl@patchchapter\relax
3755 \else
3756 \def\bbl@patchchapter{%
3757 \global\let\bbl@patchchapter\relax
3758 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3759 \bbl@toglobal\appendix
3760 \bbl@sreplace\ps@headings
3761 {\@chapapp\ thechapter}%
3762 {\bbl@chapterformat}%
3763 \bbl@toglobal\ps@headings
3764 \bbl@sreplace\chaptermark
3765 {\@chapapp\ thechapter}%
3766 {\bbl@chapterformat}%
3767 \bbl@toglobal\chaptermark
3768 \bbl@sreplace\@makechapterhead
3769 {\@chapapp\space\thechapter}%
3770 {\bbl@chapterformat}%
3771 \bbl@toglobal\@makechapterhead
3772 \gdef\bbl@chapterformat{%
3773 \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3774 {\@chapapp\space\thechapter}
3775 {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}}
3776 \let\bbl@patchappendix\bbl@patchchapter
3777 \fi\fi\fi
3778 \ifx\@part\@undefined
3779 \let\bbl@patchpart\relax
3780 \else
3781 \def\bbl@patchpart{%
3782 \global\let\bbl@patchpart\relax
3783 \bbl@sreplace\@part
3784 {\partname\nobreakspace\thepart}%
3785 {\bbl@partformat}%
3786 \bbl@toglobal\@part
3787 \gdef\bbl@partformat{%
3788 \bbl@ifunset{\bbl@partfmt@\language}%
3789 {\partname\nobreakspace\thepart}
3790 {\@nameuse{\bbl@partfmt@\language}}}}
3791 \fi

```

Date. TODO. Document

```

3792 % Arguments are _not_ protected.
3793 \let\bbl@calendar\@empty
3794 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3795 \def\bbl@localedate#1#2#3#4{%

```

```

3796 \begingroup
3797   \ifx\@empty#1\@empty\else
3798     \let\bbl@ld@calendar\@empty
3799     \let\bbl@ld@variant\@empty
3800     \edef\bbl@tempa{\zap@space#1 \@empty}%
3801     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3802     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3803     \edef\bbl@calendar{%
3804       \bbl@ld@calendar
3805       \ifx\bbl@ld@variant\@empty\else
3806         .\bbl@ld@variant
3807       \fi}%
3808     \bbl@replace\bbl@calendar{gregorian}{}%
3809     \fi
3810     \bbl@cased
3811     {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3812 \endgroup
3813 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3814 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3815   \bbl@trim@def\bbl@tempa{#1.#2}%
3816   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3817   {\bbl@trim@def\bbl@tempa{#3}%
3818     \bbl@trim\toks@{#5}%
3819     \@temptokena\expandafter{\bbl@savedate}%
3820     \bbl@exp{% Reverse order - in ini last wins
3821       \def\\bbl@savedate{%
3822         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3823         \the\@temptokena}}}%
3824   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3825     {\lowercase{\def\bbl@tempb{#6}}%
3826       \bbl@trim@def\bbl@toreplace{#5}%
3827       \bbl@TG@@date
3828       \bbl@ifunset{\bbl@date@\language @}%
3829       {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3830        % TODO. Move to a better place.
3831        \bbl@exp{%
3832          \gdef\<\language date>{\\protect\<\language date >}}%
3833          \gdef\<\language date >####1####2####3{%
3834            \\bbl@usedategroupttrue
3835            \<bbl@ensure@\language >%
3836              \\localedate{####1}{####2}{####3}}}%
3837          \\bbl@add\\bbl@savetoday{%
3838            \\SetString\\today{%
3839              \<\language date>%
3840              {\\the\year}{\\the\month}{\\the\day}}}}}%
3841     {}%
3842   \ifx\bbl@tempb\@empty\else
3843     \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3844   \fi}%
3845   {}}}

```

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.

```

3846 \let\bbl@calendar\@empty
3847 \newcommand\BabelDateSpace{\nobreakspace}
3848 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3849 \newcommand\BabelDated[1]{\number#1}
3850 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}

```

```

3851 \newcommand\BabelDateM[1]{\number#1}
3852 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3853 \newcommand\BabelDateMMMM[1]{\%
3854 \csname month\romannumeral#1\bb1@calendar name\endcsname}}%
3855 \newcommand\BabelDatey[1]{\number#1}%
3856 \newcommand\BabelDateyy[1]{\%
3857 \ifnum#1<10 0\number#1 %
3858 \else\ifnum#1<100 \number#1 %
3859 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3860 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3861 \else
3862 \bb1@error
3863 {Currently two-digit years are restricted to the\
3864 range 0-9999.}%
3865 {There is little you can do. Sorry.}%
3866 \fi\fi\fi\fi}}
3867 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3868 \def\bb1@replace@finish@iii#1{%
3869 \bb1@exp{\def\#1####1####2####3{\the\toks@}}
3870 \def\bb1@TG@date{%
3871 \bb1@replace\bb1@toreplace{[ ]}{\BabelDateSpace{}}%
3872 \bb1@replace\bb1@toreplace{[. ]}{\BabelDateDot{}}%
3873 \bb1@replace\bb1@toreplace{[d]}{\BabelDated{####3}}%
3874 \bb1@replace\bb1@toreplace{[dd]}{\BabelDatedd{####3}}%
3875 \bb1@replace\bb1@toreplace{[M]}{\BabelDateM{####2}}%
3876 \bb1@replace\bb1@toreplace{[MM]}{\BabelDateMM{####2}}%
3877 \bb1@replace\bb1@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3878 \bb1@replace\bb1@toreplace{[y]}{\BabelDatey{####1}}%
3879 \bb1@replace\bb1@toreplace{[yy]}{\BabelDateyy{####1}}%
3880 \bb1@replace\bb1@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3881 \bb1@replace\bb1@toreplace{[y|]}{\bb1@datecctr[####1|]}%
3882 \bb1@replace\bb1@toreplace{[m|]}{\bb1@datecctr[####2|]}%
3883 \bb1@replace\bb1@toreplace{[d|]}{\bb1@datecctr[####3|]}%
3884 % Note after \bb1@replace \toks@ contains the resulting string.
3885 TODO - Using this implicit behavior doesn't seem a good idea.
3886 \bb1@replace@finish@iii\bb1@toreplace}
3887 \def\bb1@datecctr{\expandafter\bb1@xdatecctr\expandafter}
3888 \def\bb1@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.

```

3889 \def\bb1@provide@lsys#1{%
3890 \bb1@ifunset{bb1@lname@#1}%
3891 {\bb1@load@info{#1}}%
3892 {}%
3893 \bb1@csarg\let{lsys@#1}\@empty
3894 \bb1@ifunset{bb1@sname@#1}{\bb1@csarg\gdef{sname@#1}{Default}}}%
3895 \bb1@ifunset{bb1@sotf@#1}{\bb1@csarg\gdef{sotf@#1}{DFLT}}}%
3896 \bb1@csarg\bb1@add@list{lsys@#1}{Script=\bb1@cs{sname@#1}}%
3897 \bb1@ifunset{bb1@lname@#1}{%
3898 {\bb1@csarg\bb1@add@list{lsys@#1}{Language=\bb1@cs{lname@#1}}}%
3899 \ifcase\bb1@engine\or\or
3900 \bb1@ifunset{bb1@prehc@#1}{%
3901 {\bb1@exp{\bb1@ifblank{\bb1@cs{prehc@#1}}}%
3902 {}%
3903 {\ifx\bb1@xenoxyph\@undefined
3904 \let\bb1@xenoxyph\bb1@xenoxyph@d
3905 \ifx\AtBeginDocument\@notprerr
3906 \expandafter\@secondoftwo % to execute right now

```

```

3907         \fi
3908         \AtBeginDocument{%
3909             \expandafter\bbbl@add
3910             \csname selectfont \endcsname{\bbbl@xenohyph}%
3911             \expandafter\selectlanguage\expandafter{\language}%
3912             \expandafter\bbbl@tglobal\csname selectfont \endcsname}%
3913         \fi}%
3914 \fi
3915 \bbbl@csarg\bbbl@tglobal{lsys@#1}}
3916 \def\bbbl@xenohyph@d{%
3917     \bbbl@ifset{\bbbl@prehc@\language}%
3918     {\ifnum\hyphenchar\font=\defaultshyphenchar
3919         \iffontchar\font\bbbl@c1{\prehc}\relax
3920         \hyphenchar\font\bbbl@c1{\prehc}\relax
3921     \else\iffontchar\font"200B
3922         \hyphenchar\font"200B
3923     \else
3924         \bbbl@warning
3925         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3926         in the current font, and therefore the hyphen\\%
3927         will be printed. Try changing the fontspec's\\%
3928         'HyphenChar' to another value, but be aware\\%
3929         this setting is not safe (see the manual)}%
3930         \hyphenchar\font\defaultshyphenchar
3931     \fi\fi
3932     \fi}%
3933     {\hyphenchar\font\defaultshyphenchar}}
3934 % \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).

```

3935 \def\bbbl@load@info#1{%
3936     \def\BabelBeforeIni##1##2{%
3937         \begingroup
3938         \bbbl@read@ini{##1}0%
3939         \endinput           % babel- .tex may contain onlypreamble's
3940         \endgroup}%         boxed, to avoid extra spaces:
3941     {\bbbl@input@texini{#1}}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in \TeX . Non-digits characters are kept. The first macro is the generic “localized” command.

```

3942 \def\bbbl@setdigits#1#2#3#4#5{%
3943     \bbbl@exp{%
3944         \def\<\language digits>####1{%           ie, \langdigits
3945             \<\bbbl@digits@\language>####1\\\nil}%
3946             \let\<\bbbl@cntr@digits@\language>\<\language digits>%
3947             \def\<\language counter>####1{%       ie, \langcounter
3948                 \expandafter\<\bbbl@counter@\language>%
3949                 \csname c#####1\endcsname}%
3950             \def\<\bbbl@counter@\language>####1{% ie, \bbbl@counter@lang
3951                 \expandafter\<\bbbl@digits@\language>%
3952                 \number####1\\\nil}}}%
3953     \def\bbbl@tempa##1##2##3##4##5{%
3954         \bbbl@exp{%      Wow, quite a lot of hashes! :-(
3955             \def\<\bbbl@digits@\language>#####1{%
3956                 \ifx#####1\\\nil                % ie, \bbbl@digits@lang

```


[illegible]

```

3973 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3974   \ifx\\#1%                % \\ before, in case #1 is multiletter
3975     \bbl@exp{%
3976       \def\\bbl@tempa####1{%
3977         \<ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3978   \else
3979     \toks@\xexpandafter{\the\toks@\or #1}%
3980     \expandafter\bbl@buildifcase
3981   \fi}

```

```

3982 \newcommand\localenumber[2]{\bbl@cs{cntr@#1@\language}\#2}%
3983 \def\bbl@localecntr#1#2{\localenumber{#2}{#1}}
3984 \newcommand\localecounter[2]{%
3985   \expandafter\bbl@localecntr
3986   \expandafter{\number\csname c@#2\endcsname}{#1}}
3987 \def\bbl@alphanumeric#1#2{%
3988   \expandafter\bbl@alphanumeric@i\number#2 76543210\@@{#1}}
3989 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\@@#9{%
3990   \ifcase\car#8\@nil\or    % Currenty >10000, but prepared for bigger
3991     \bbl@alphanumeric@ii{#9}000000#1\or
3992     \bbl@alphanumeric@ii{#9}00000#1#2\or
3993     \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3994     \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3995     \bbl@alphanum@invalid{>9999}%
3996   \fi}
3997 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3998   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3999   {\bbl@cs{cntr@#1.4@\language}#5%
4000    \bbl@cs{cntr@#1.3@\language}#6%
4001    \bbl@cs{cntr@#1.2@\language}#7%
4002    \bbl@cs{cntr@#1.1@\language}#8%
4003    \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4004     \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
4005     {\bbl@cs{cntr@#1.S.321@\language}}%
4006   \fi}%
4007 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}

```

```

4008 \def\bbl@alphnum@invalid#1{%
4009   \bbl@error{Alphabetic numeral too large (#1)}%
4010   {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.

```

4011 \newcommand\localeinfo[1]{%
4012   \bbl@ifunset{\bbl@csname \bbl@info@#1\endcsname @\language}%
4013   {\bbl@error{I've found no info for the current locale.\%
4014     The corresponding ini file has not been loaded\%
4015     Perhaps it doesn't exist}%
4016     {See the manual for details.}}%
4017   {\bbl@cs{\csname \bbl@info@#1\endcsname @\language}}}%
4018   \@namedef{\bbl@info@name.locale}{\lcnam}
4019   \@namedef{\bbl@info@tag.ini}{\lini}
4020   \@namedef{\bbl@info@name.english}{\elname}
4021   \@namedef{\bbl@info@name.opentype}{\lname}
4022   \@namedef{\bbl@info@tag.bcp47}{\tbc}
4023   \@namedef{\bbl@info@language.tag.bcp47}{\lbc}
4024   \@namedef{\bbl@info@tag.opentype}{\lotf}
4025   \@namedef{\bbl@info@script.name}{\esname}
4026   \@namedef{\bbl@info@script.name.opentype}{\sname}
4027   \@namedef{\bbl@info@script.tag.bcp47}{\sbcp}
4028   \@namedef{\bbl@info@script.tag.opentype}{\sotf}
4029   \let\bbl@ensureinfo\@gobble
4030 \newcommand\BabelEnsureInfo{%
4031   \ifx\InputIfFileExists\@undefined\else
4032     \def\bbl@ensureinfo##1{%
4033       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}}%
4034   \fi
4035   \bbl@foreach\bbl@loaded{%
4036     \def\language{##1}%
4037     \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`.

```

4038 \newcommand\getlocaleproperty{%
4039   \@ifstar\bbl@getproperty@s\bbl@getproperty@x%
4040   \def\bbl@getproperty@s#1#2#3{%
4041     \let#1\relax
4042     \def\bbl@elt##1##2##3{%
4043       \bbl@ifsamestring{##1/##2}{##3}%
4044       {\providecommand#1{##3}%
4045       \def\bbl@elt####1####2####3{}}}%
4046     {}}%
4047     \bbl@cs{inidata@#2}}%
4048   \def\bbl@getproperty@x#1#2#3{%
4049     \bbl@getproperty@s{#1}{#2}{#3}%
4050     \ifx#1\relax
4051       \bbl@error
4052       {Unknown key for locale '#2':\%
4053       #3\%
4054       \string#1 will be set to \relax}%
4055       {Perhaps you misspelled it.}%
4056     \fi}
4057   \let\bbl@ini@loaded\@empty
4058 \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.

```
4059 \newcommand\babeladjust[1]{% TODO. Error handling.
4060   \bbl@forkv{#1}{%
4061     \bbl@ifunset{\bbl@ADJ@##1@##2}%
4062     {\bbl@cs{ADJ@##1}{##2}}%
4063     {\bbl@cs{ADJ@##1@##2}}}
4064 %
4065 \def\bbl@adjust@lua#1#2{%
4066   \ifvmode
4067     \ifnum\currentgrouplevel=\z@
4068       \directlua{ Babel.#2 }%
4069       \expandafter\expandafter\expandafter\@gobble
4070     \fi
4071   \fi
4072   {\bbl@error   % The error is gobbled if everything went ok.
4073     {Currently, #1 related features can be adjusted only\\%
4074       in the main vertical list.}%
4075     {Maybe things change in the future, but this is what it is.}}}
4076 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
4077   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4078 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
4079   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4080 \@namedef{\bbl@ADJ@bidi.text@on}{%
4081   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4082 \@namedef{\bbl@ADJ@bidi.text@off}{%
4083   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4084 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
4085   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4086 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
4087   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4088 %
4089 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
4090   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4091 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
4092   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4093 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
4094   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4095 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
4096   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4097 %
4098 \def\bbl@adjust@layout#1{%
4099   \ifvmode
4100     #1%
4101     \expandafter\@gobble
4102   \fi
4103   {\bbl@error   % The error is gobbled if everything went ok.
4104     {Currently, layout related features can be adjusted only\\%
4105       in vertical mode.}%
4106     {Maybe things change in the future, but this is what it is.}}}
4107 \@namedef{\bbl@ADJ@layout.tabular@on}{%
4108   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4109 \@namedef{\bbl@ADJ@layout.tabular@off}{%
4110   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4111 \@namedef{\bbl@ADJ@layout.lists@on}{%
4112   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4113 \@namedef{\bbl@ADJ@layout.lists@off}{%
```

```

4114 \bbl@adjust@layout{\let\list\bbl@OL@list}}
4115 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4116 \bbl@activateposthyphen}
4117 %
4118 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4119 \bbl@bcpallowedtrue}
4120 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4121 \bbl@bcpallowedfalse}
4122 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4123 \def\bbl@bcp@prefix{#1}}
4124 \def\bbl@bcp@prefix{bcp47-}
4125 \@namedef{bbl@ADJ@autoload.options}#1{%
4126 \def\bbl@autoload@options{#1}}
4127 \let\bbl@autoload@bcptoptions\empty
4128 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4129 \def\bbl@autoload@bcptoptions{#1}}
4130 \newif\ifbbl@bcptoname
4131 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4132 \bbl@bcptonametrue}
4133 \BabelEnsureInfo}
4134 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4135 \bbl@bcptonamefalse}
4136 % TODO: use babel name, override
4137 %
4138 % As the final task, load the code for lua.
4139 %
4140 \ifx\directlua\@undefined\else
4141 \ifx\bbl@luapatterns\@undefined
4142 \input luabel.def
4143 \fi
4144 \fi
4145 </core>

A proxy file for switch.def
4146 <*kernel>
4147 \let\bbl@onlyswitch\@empty
4148 \input babel.def
4149 \let\bbl@onlyswitch\@undefined
4150 </kernel>
4151 <*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.

```

4152 <<Make sure ProvidesFile is defined>>
4153 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
4154 \xdef\bbl@format{\jobname}
4155 \def\bbl@version{<<version>>}
4156 \def\bbl@date{<<date>>}

```

```

4157 \ifx\AtBeginDocument\@undefined
4158   \def\@empty{}
4159   \let\orig@dump\dump
4160   \def\dump{%
4161     \ifx\@ztryfc\@undefined
4162       \else
4163         \toks0=\expandafter{\@preamblecmds}%
4164         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4165         \def\@begindocumenthook{}%
4166       \fi
4167       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4168 \fi
4169 <<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.

```

4170 \def\process@line#1#2 #3 #4 {%
4171   \ifx=#1%
4172     \process@synonym{#2}%
4173   \else
4174     \process@language{#1#2}{#3}{#4}%
4175   \fi
4176   \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.

```

4177 \toks@{}
4178 \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.

```

4179 \def\process@synonym#1{%
4180   \ifnum\last@language=\m@ne
4181     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4182   \else
4183     \expandafter\chardef\csname l@#1\endcsname\last@language
4184     \wlog{\string\l@#1=\string\language\the\last@language}%
4185     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
4186     \csname\language\hyphenmins\endcsname
4187     \let\bbl@elt\relax
4188     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4189   \fi}

```

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

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

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

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

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting. Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```

4190 \def\process@language#1#2#3{%
4191   \expandafter\addlanguage\csname l@#1\endcsname
4192   \expandafter\language\csname l@#1\endcsname
4193   \edef\language{#1}%
4194   \bbl@hook@everylanguage{#1}%
4195   % > luatex
4196   \bbl@get@enc#1::@@@
4197   \begingroup
4198     \lefthyphenmin\m@ne
4199     \bbl@hook@loadpatterns{#2}%
4200     % > luatex
4201     \ifnum\lefthyphenmin=\m@ne
4202     \else
4203       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4204         \the\lefthyphenmin\the\righthyphenmin}%
4205     \fi
4206   \endgroup
4207   \def\bbl@tempa{#3}%
4208   \ifx\bbl@tempa\@empty\else
4209     \bbl@hook@loadexceptions{#3}%
4210     % > luatex
4211   \fi
4212   \let\bbl@elt\relax
4213   \edef\bbl@languages{%
4214     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4215   \ifnum\the\language=\z@
4216     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4217       \set@hyphenmins\tw@\thr@@\relax
4218     \else
4219       \expandafter\expandafter\expandafter\set@hyphenmins
4220       \csname #1hyphenmins\endcsname
4221     \fi
4222     \the\toks@
4223     \toks@{}%
4224   \fi}

```

`\bbl@get@enc` The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. It uses delimited arguments to achieve this.

```

4225 \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.

```

4226 \def\bbl@hook@everylanguage#1{
4227 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4228 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4229 \def\bbl@hook@loadkernel#1{%
4230 \def\addlanguage{\csname newlanguage\endcsname}%
4231 \def\adddialect##1##2{%
4232 \global\chardef##1##2\relax
4233 \wlog{\string##1 = a dialect from \string\language##2}}%
4234 \def\iflanguage##1{%
4235 \expandafter\ifx\csname l@##1\endcsname\relax
4236 \@nolanerr{##1}%
4237 \else
4238 \ifnum\csname l@##1\endcsname=\language
4239 \expandafter\expandafter\expandafter\@firstoftwo
4240 \else
4241 \expandafter\expandafter\expandafter\@secondoftwo
4242 \fi
4243 \fi}%
4244 \def\providehyphenmins##1##2{%
4245 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4246 \@namedef{##1hyphenmins}{##2}%
4247 \fi}%
4248 \def\set@hyphenmins##1##2{%
4249 \lefthyphenmin##1\relax
4250 \righthyphenmin##2\relax}%
4251 \def\selectlanguage{%
4252 \errhelp{Selecting a language requires a package supporting it}%
4253 \errmessage{Not loaded}}%
4254 \let\foreignlanguage\selectlanguage
4255 \let\otherlanguage\selectlanguage
4256 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4257 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4258 \def\setlocale{%
4259 \errhelp{Find an armchair, sit down and wait}%
4260 \errmessage{Not yet available}}%
4261 \let\uselocale\setlocale
4262 \let\locale\setlocale
4263 \let\selectlocale\setlocale
4264 \let\localename\setlocale
4265 \let\textlocale\setlocale
4266 \let\textlanguage\setlocale
4267 \let\languagetext\setlocale}
4268 \begingroup
4269 \def\AddBabelHook#1#2{%
4270 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4271 \def\next{\toks1}%
4272 \else
4273 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4274 \fi
4275 \next}
4276 \ifx\directlua\@undefined
4277 \ifx\XeTeXinputencoding\@undefined\else
4278 \input xebabel.def
4279 \fi
4280 \else
4281 \input luababel.def
4282 \fi
4283 \openin1 = babel-\bbl@format.cfg
4284 \ifeof1

```

```

4285 \else
4286 \input babel-\bbl@format.cfg\relax
4287 \fi
4288 \closein1
4289 \endgroup
4290 \bbl@hook@loadkernel{switch.def}

```

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

```

4291 \openin1 = language.dat

```

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

```

4292 \def\language{english}%
4293 \ifeof1
4294 \message{I couldn't find the file language.dat,\space
4295         I will try the file hyphen.tex}
4296 \input hyphen.tex\relax
4297 \chardef\l@english\z@
4298 \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.

```

4299 \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.

```

4300 \loop
4301 \endlinechar\m@ne
4302 \read1 to \bbl@line
4303 \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.

```

4304 \if T\ifeof1F\fi T\relax
4305 \ifx\bbl@line\@empty\else
4306 \edef\bbl@line{\bbl@line\space\space\space}%
4307 \expandafter\process@line\bbl@line\relax
4308 \fi
4309 \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.

```

4310 \begingroup
4311 \def\bbl@elt#1#2#3#4{%
4312 \global\language=#2\relax
4313 \gdef\language{#1}%
4314 \def\bbl@elt##1##2##3##4{}}%
4315 \bbl@languages
4316 \endgroup
4317 \fi
4318 \closein1

```

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

```

4319 \if/\the\toks@\else

```



```

4320 \errhelp{language.dat loads no language, only synonyms}
4321 \errmessage{Orphan language synonym}
4322 \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.

```

4323 \let\bbl@line\@undefined
4324 \let\process@line\@undefined
4325 \let\process@synonym\@undefined
4326 \let\process@language\@undefined
4327 \let\bbl@get@enc\@undefined
4328 \let\bbl@hyph@enc\@undefined
4329 \let\bbl@tempa\@undefined
4330 \let\bbl@hook@loadkernel\@undefined
4331 \let\bbl@hook@everylanguage\@undefined
4332 \let\bbl@hook@loadpatterns\@undefined
4333 \let\bbl@hook@loadexceptions\@undefined
4334 \let\patterns\@undefined

```

Here the code for `initTeX` 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].

```

4335 <<(*More package options)>> ≡
4336 \chardef\bbl@bidimode\z@
4337 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4338 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4339 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4340 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4341 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4342 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4343 <</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.

```

4344 <<(*Font selection)>> ≡
4345 \bbl@trace{Font handling with fontspec}
4346 \ifx\ExplSyntaxOn\@undefined\else
4347   \ExplSyntaxOn
4348   \catcode`\ =10
4349   \def\bbl@loadfontspec{%
4350     \usepackage{fontspec}%
4351     \expandafter
4352     \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4353       Font '\l_fontspec_fontname_tl' is using the\\%
4354       default features for language '##1'.\\%
4355       That's usually fine, because many languages\\%
4356       require no specific features, but if the output is\\%
4357       not as expected, consider selecting another font.}
4358     \expandafter
4359     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%
4360       Font '\l_fontspec_fontname_tl' is using the\\%

```

```

4361      default features for script '##2'.\\%
4362      That's not always wrong, but if the output is\\%
4363      not as expected, consider selecting another font.}}
4364 \ExplSyntaxOff
4365 \fi
4366 \@onlypreamble\babelfont
4367 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4368 \bbl@foreach{#1}{\%
4369 \expandafter\ifx\csname date##1\endcsname\relax
4370 \IfFileExists{babel-##1.tex}%
4371 {\babelprovide{##1}}%
4372 }%
4373 \fi}%
4374 \edef\bbl@tempa{#1}%
4375 \def\bbl@tempb{#2}% Used by \bbl@babelfont
4376 \ifx\fontspec\undefined
4377 \bbl@loadfontspec
4378 \fi
4379 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4380 \bbl@babelfont}
4381 \newcommand\bbl@babelfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
4382 \bbl@ifunset{\bbl@tempb family}%
4383 {\bbl@providedefam{\bbl@tempb}}%
4384 {\bbl@exp{%
4385 \\\bbl@sreplace\<\bbl@tempb family >%
4386 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4387 % For the default font, just in case:
4388 \bbl@ifunset{\bbl@lsys\<\language\>}{\bbl@provide@lsys{\language}}}%
4389 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4390 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4391 \bbl@exp{%
4392 \let\<\bbl@tempb dflt@\<\language\>\<\bbl@tempb dflt@>%
4393 \\\bbl@font@set\<\bbl@tempb dflt@\<\language\>%
4394 \<\bbl@tempb default>\<\bbl@tempb family>}}}%
4395 {\bbl@foreach\bbl@tempa{\< ie bbl@rmdflt@lang / *scrt
4396 \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:

```

4397 \def\bbl@providedefam#1{%
4398 \bbl@exp{%
4399 \\\newcommand\<#1default>{}% Just define it
4400 \\\bbl@add@list\<\bbl@font@fams{#1}%
4401 \\\DeclareRobustCommand\<#1family>{%
4402 \\\not@math@alphabet\<#1family>\relax
4403 \\\fontfamily\<#1default>\selectfont}%
4404 \\\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.

```

4405 \def\bbl@nostdfont#1{%
4406 \bbl@ifunset{\bbl@WFF@\<f@family>%
4407 {\bbl@csarg\gdef\<WFF@\<f@family>{}% Flag, to avoid dupl warns
4408 \bbl@infowarn{The current font is not a babel standard family:\\%
4409 #1%
4410 \fontname\font\\%
4411 There is nothing intrinsically wrong with this warning, and\\%
4412 you can ignore it altogether if you do not need these\\%
4413 families. But if they are used in the document, you should be\\%
4414 aware 'babel' will no set Script and Language for them, so\\%

```

```

4415     you may consider defining a new family with \string\babelfont.\%
4416     See the manual for further details about \string\babelfont.\%
4417     Reported}}
4418     {}}%
4419 \gdef\bbl@switchfont{%
4420   \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
4421   \bbl@exp{%   eg Arabic -> arabic
4422     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4423   \bbl@foreach\bbl@font@fams{%
4424     \bbl@ifunset{\bbl@##1dflt@\language}%      (1) language?
4425     {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}%    (2) from script?
4426     {\bbl@ifunset{\bbl@##1dflt@}%              2=F - (3) from generic?
4427     {}%                                          123=F - nothing!
4428     {\bbl@exp{%                                3=T - from generic
4429       \global\let<\bbl@##1dflt@\language>%
4430       \<\bbl@##1dflt@>}}}%
4431     {\bbl@exp{%                                2=T - from script
4432       \global\let<\bbl@##1dflt@\language>%
4433       \<\bbl@##1dflt@*\bbl@tempa>}}}%
4434     {}}%                                          1=T - language, already defined
4435   \def\bbl@tempa{\bbl@nostdfont{}}}%
4436   \bbl@foreach\bbl@font@fams{%   don't gather with prev for
4437     \bbl@ifunset{\bbl@##1dflt@\language}%
4438     {\bbl@cs{famrst@##1}%
4439     \global\bbl@csarg\let{famrst@##1}\relax}%
4440     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4441       \\bbl@add\\originalTeX{%
4442       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4443       \<##1default>\<##1family>{##1}}}%
4444       \\bbl@font@set<\bbl@##1dflt@\language>% the main part!
4445       \<##1default>\<##1family>}}}%
4446   \bbl@ifrestoring{{}\bbl@tempa}}}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

4447 \ifx\f@family\undefined\else   % if latex
4448 \ifcase\bbl@engine              % if pdftex
4449   \let\bbl@cckstdfonts\relax
4450 \else
4451   \def\bbl@cckstdfonts{%
4452     \begingroup
4453     \global\let\bbl@cckstdfonts\relax
4454     \let\bbl@tempa\@empty
4455     \bbl@foreach\bbl@font@fams{%
4456       \bbl@ifunset{\bbl@##1dflt@}%
4457       {\nameuse{##1family}%
4458       \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4459       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4460       \space\space\fontname\font\\}%
4461       \bbl@csarg\xdef{##1dflt@}{\f@family}%
4462       \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4463     {}}%
4464   \ifx\bbl@tempa\@empty\else
4465     \bbl@infowarn{The following font families will use the default\\%
4466     settings for all or some languages:\\%
4467     \bbl@tempa
4468     There is nothing intrinsically wrong with it, but\\%
4469     'babel' will no set Script and Language, which could\\%
4470     be relevant in some languages. If your document uses\\%

```

```

4471             these families, consider redefining them with \string\babelfont.\\%
4472             Reported}%
4473             \fi
4474         \endgroup}
4475     \fi
4476 \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.

```

4477 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4478     \bbl@xin@{<>}{#1}%
4479     \ifin@
4480         \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4481         \fi
4482     \bbl@exp{%
4483         \def\\#2#1% eg, \rmdefault{\bbl@rmdflt@lang}
4484         \\bbl@ifsamestring{#2}{\f@family}%
4485         {\\#3%
4486             \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4487         \let\\bbl@tempa\relax}%
4488     {}}
4489 % TODO - next should be global?, but even local does its job. I'm
4490 % still not sure -- must investigate:
4491 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4492     \let\bbl@tempe\bbl@mapselect
4493     \let\bbl@mapselect\relax
4494     \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4495     \let#4\@empty % Make sure \renewfontfamily is valid
4496     \bbl@exp{%
4497         \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4498         \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4499         {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4500         \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4501         {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4502         \\renewfontfamily\\#4%
4503         [\bbl@cs{lsys@\language\name},#2]{#3}% ie \bbl@exp{.}{#3}
4504     \begingroup
4505         #4%
4506         \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4507     \endgroup
4508     \let#4\bbl@temp@fam
4509     \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4510     \let\bbl@mapselect\bbl@tempe}%

```

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

```

4511 \def\bbl@font@rst#1#2#3#4{%
4512     \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.

```

4513 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4514 \newcommand\babelFSstore[2][{}]{%
4515     \bbl@ifblank{#1}%

```

```

4516     {\bbl@csarg\def\sname@#2}{Latin}}%
4517     {\bbl@csarg\def\sname@#2}{#1}}%
4518     \bbl@provide@dirs{#2}%
4519     \bbl@csarg\ifnum{wdir@#2}>\z@
4520         \let\bbl@beforeforeign\leavevmode
4521         \EnableBabelHook{babel-bidi}%
4522     \fi
4523     \bbl@foreach{#2}{%
4524         \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4525         \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4526         \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4527 \def\bbl@FSstore#1#2#3#4{%
4528     \bbl@csarg\edef{#2default#1}{#3}%
4529     \expandafter\addto\csname extras#1\endcsname{%
4530         \let#4#3%
4531         \ifx#3\f@family
4532             \edef#3{\csname bbl@#2default#1\endcsname}%
4533             \fontfamily{#3}\selectfont
4534         \else
4535             \edef#3{\csname bbl@#2default#1\endcsname}%
4536             \fi}%
4537     \expandafter\addto\csname noextras#1\endcsname{%
4538         \ifx#3\f@family
4539             \fontfamily{#4}\selectfont
4540         \fi
4541         \let#3#4}}
4542 \let\bbl@langfeatures\@empty
4543 \def\babelFSfeatures{% make sure \fontspec is redefined once
4544     \let\bbl@ori@fontspec\fontspec
4545     \renewcommand\fontspec[1][{}]{%
4546         \bbl@ori@fontspec[\bbl@langfeatures##1]}
4547     \let\babelFSfeatures\bbl@FSfeatures
4548     \babelFSfeatures}
4549 \def\bbl@FSfeatures#1#2{%
4550     \expandafter\addto\csname extras#1\endcsname{%
4551         \babel@save\bbl@langfeatures
4552         \edef\bbl@langfeatures{#2,}}
4553 <</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.

```

4554 <<{*Footnote changes}>> ≡
4555 \bbl@trace{Bidi footnotes}
4556 \ifnum\bbl@bidimode>\z@
4557     \def\bbl@footnote#1#2#3{%
4558         \@ifnextchar[%
4559             {\bbl@footnote@o{#1}{#2}{#3}}%
4560             {\bbl@footnote@x{#1}{#2}{#3}}}
4561     \long\def\bbl@footnote@x#1#2#3#4{%
4562         \bgroup
4563         \select@language@x{\bbl@main@language}%
4564         \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4565         \egroup}
4566     \long\def\bbl@footnote@o#1#2#3[#4]#5{%

```

```

4567 \bgroup
4568 \select@language@x{\bbl@main@language}%
4569 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#3}%
4570 \egroup}
4571 \def\bbl@footnotetext#1#2#3{%
4572 \ifnextchar[%
4573 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4574 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4575 \long\def\bbl@footnotetext@x#1#2#3#4{%
4576 \bgroup
4577 \select@language@x{\bbl@main@language}%
4578 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}{#3}%
4579 \egroup}
4580 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4581 \bgroup
4582 \select@language@x{\bbl@main@language}%
4583 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}{#3}%
4584 \egroup}
4585 \def\BabelFootnote#1#2#3#4{%
4586 \ifx\bbl@fn@footnote\undefined
4587 \let\bbl@fn@footnote\footnote
4588 \fi
4589 \ifx\bbl@fn@footnotetext\undefined
4590 \let\bbl@fn@footnotetext\footnotetext
4591 \fi
4592 \bbl@ifblank{#2}%
4593 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4594 \@namedef{\bbl@stripslash#1text}%
4595 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4596 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4597 \@namedef{\bbl@stripslash#1text}%
4598 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4599 \fi
4600 <</Footnote changes>>

```

Now, the code.

```

4601 (*xetex)
4602 \def\BabelStringsDefault{unicode}
4603 \let\xebbl@stop\relax
4604 \AddBabelHook{xetex}{encodedcommands}{%
4605 \def\bbl@tempa{#1}%
4606 \ifx\bbl@tempa\empty
4607 \XeTeXinputencoding"bytes"%
4608 \else
4609 \XeTeXinputencoding"#1"%
4610 \fi
4611 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4612 \AddBabelHook{xetex}{stopcommands}{%
4613 \xebbl@stop
4614 \let\xebbl@stop\relax}
4615 \def\bbl@intraspace#1 #2 #3\@@{%
4616 \bbl@csarg\gdef{xeisp@\languagename}%
4617 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4618 \def\bbl@intrapenalty#1\@@{%
4619 \bbl@csarg\gdef{xeipn@\languagename}%
4620 {\XeTeXlinebreakpenalty #1\relax}}
4621 \def\bbl@provide@intraspace{%
4622 \bbl@xin@{\bbl@cl{\lnbrk}}{s}%
4623 \ifin@else\bbl@xin@{\bbl@cl{\lnbrk}}{c}\fi

```

```

4624 \ifin@
4625 \bbl@ifunset{bbl@intsp@\languagename}{}%
4626 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4627 \ifx\bbl@KVP@intraspace\@nil
4628 \bbl@exp{%
4629 \\\bbl@intraspace\bbl@cl{intsp}\@{}%
4630 \fi
4631 \ifx\bbl@KVP@intrapenalty\@nil
4632 \bbl@intrapenalty0\@{}
4633 \fi
4634 \fi
4635 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4636 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@{}
4637 \fi
4638 \ifx\bbl@KVP@intrapenalty\@nil\else
4639 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@{}
4640 \fi
4641 \bbl@exp{%
4642 \\\bbl@add<extras\languagename>{%
4643 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4644 \<bbl@xeisp@\languagename>%
4645 \<bbl@xeipn@\languagename>%
4646 \\\bbl@toglobal<extras\languagename>%
4647 \\\bbl@add<noextras\languagename>%
4648 \XeTeXlinebreaklocale "en"%
4649 \\\bbl@toglobal<noextras\languagename>}%
4650 \ifx\bbl@ispacesize\undefined
4651 \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4652 \ifx\AtBeginDocument\@notprerr
4653 \expandafter\@secondoftwo % to execute right now
4654 \fi
4655 \AtBeginDocument{%
4656 \expandafter\bbl@add
4657 \csname selectfont \endcsname{\bbl@ispacesize}%
4658 \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4659 \fi}%
4660 \fi}
4661 \ifx\DisableBabelHook\undefined\endinput\fi
4662 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4663 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4664 \DisableBabelHook{babel-fontspec}
4665 <<Font selection>>
4666 \input txtbabel.def
4667 </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.

```

4668 <texxet>
4669 \providecommand\bbl@provide@intraspace{}
4670 \bbl@trace{Redefinitions for bidi layout}
4671 \def\bbl@sspre@caption{%

```

```

4672 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4673 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4674 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4675 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4676 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4677 \def\@hangfrom#1{%
4678 \setbox\@tempboxa\hbox{#1}%
4679 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4680 \noindent\box\@tempboxa}
4681 \def\raggedright{%
4682 \let\@centercr
4683 \bbl@startskip\z@skip
4684 \@rightskip\@flushglue
4685 \bbl@endskip\@rightskip
4686 \parindent\z@
4687 \parfillskip\bbl@startskip}
4688 \def\raggedleft{%
4689 \let\@centercr
4690 \bbl@startskip\@flushglue
4691 \bbl@endskip\z@skip
4692 \parindent\z@
4693 \parfillskip\bbl@endskip}
4694 \fi
4695 \IfBabelLayout{lists}
4696 {\bbl@sreplace\list
4697 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4698 \def\bbl@listleftmargin{%
4699 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4700 \ifcase\bbl@engine
4701 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4702 \def\p@enumiii{\p@enumii}\theenumii{}\fi
4703 \fi
4704 \bbl@sreplace\@verbatim
4705 {\leftskip\@totalleftmargin}%
4706 {\bbl@startskip\textwidth
4707 \advance\bbl@startskip-\linewidth}%
4708 \bbl@sreplace\@verbatim
4709 {\rightskip\z@skip}%
4710 {\bbl@endskip\z@skip}}%
4711 {}
4712 \IfBabelLayout{contents}
4713 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4714 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4715 {}
4716 \IfBabelLayout{columns}
4717 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4718 \def\bbl@outputbox#1{%
4719 \hb@xt@\textwidth{%
4720 \hskip\columnwidth
4721 \hfil
4722 {\normalcolor\vrule \@width\columnseprule}%
4723 \hfil
4724 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4725 \hskip-\textwidth
4726 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4727 \hskip\columnsep
4728 \hskip\columnwidth}}}%
4729 {}
4730 <<Footnote changes>>

```



```

4731 \IfBabelLayout{footnotes}%
4732   {\BabelFootnote\footnote\language\language\language}%
4733   \BabelFootnote\localfootnote\language\language}%
4734   \BabelFootnote\mainfootnote\language\language}%
4735   {}

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.

4736 \IfBabelLayout{counters}%
4737   {\let\bbl@latinarabic=\@arabic
4738    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4739   \let\bbl@asciroman=\@roman
4740   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4741   \let\bbl@asciiRoman=\@Roman
4742   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
4743 \end{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 file is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg. \babelpatterns).

```

4744 \begin{luatex}
4745 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4746 \bbl@trace{Read language.dat}
4747 \ifx\bbl@readstream\undefined
4748   \csname newread\endcsname\bbl@readstream
4749 \fi
4750 \begin{group

```

```

4751 \toks@{}
4752 \count@ \z@ % 0=start, 1=0th, 2=normal
4753 \def\bbl@process@line#1#2 #3 #4 {%
4754   \ifx=#1%
4755     \bbl@process@synonym{#2}%
4756   \else
4757     \bbl@process@language{#1#2}{#3}{#4}%
4758   \fi
4759   \ignorespaces}
4760 \def\bbl@manylang{%
4761   \ifnum\bbl@last>\@ne
4762     \bbl@info{Non-standard hyphenation setup}%
4763   \fi
4764   \let\bbl@manylang\relax}
4765 \def\bbl@process@language#1#2#3{%
4766   \ifcase\count@
4767     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4768   \or
4769     \count@\tw@
4770   \fi
4771   \ifnum\count@=\tw@
4772     \expandafter\addlanguage\csname l@#1\endcsname
4773     \language\allocationnumber
4774     \chardef\bbl@last\allocationnumber
4775     \bbl@manylang
4776     \let\bbl@elt\relax
4777     \xdef\bbl@languages{%
4778       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4779   \fi
4780   \the\toks@
4781   \toks@{}}
4782 \def\bbl@process@synonym@aux#1#2{%
4783   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4784   \let\bbl@elt\relax
4785   \xdef\bbl@languages{%
4786     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4787 \def\bbl@process@synonym#1{%
4788   \ifcase\count@
4789     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4790   \or
4791     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4792   \else
4793     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4794   \fi}
4795 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4796   \chardef\l@english\z@
4797   \chardef\l@USenglish\z@
4798   \chardef\bbl@last\z@
4799   \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
4800   \gdef\bbl@languages{%
4801     \bbl@elt{english}{0}{\hyphen.tex}{}%
4802     \bbl@elt{USenglish}{0}{}}
4803 \else
4804   \global\let\bbl@languages@format\bbl@languages
4805   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4806     \ifnum#2>\z@\else
4807       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4808     \fi}%
4809   \xdef\bbl@languages{\bbl@languages}%

```

```

4810 \fi
4811 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4812 \bbl@languages
4813 \openin\bbl@readstream=language.dat
4814 \ifeof\bbl@readstream
4815 \bbl@warning{I couldn't find language.dat. No additional\\%
4816             patterns loaded. Reported}%
4817 \else
4818 \loop
4819 \endlinechar\m@ne
4820 \read\bbl@readstream to \bbl@line
4821 \endlinechar\^^M
4822 \if T\ifeof\bbl@readstream F\fi T\relax
4823 \ifx\bbl@line\@empty\else
4824 \edef\bbl@line{\bbl@line\space\space\space}%
4825 \expandafter\bbl@process@line\bbl@line\relax
4826 \fi
4827 \repeat
4828 \fi
4829 \endgroup
4830 \bbl@trace{Macros for reading patterns files}
4831 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
4832 \ifx\babelcatcodetablenum\@undefined
4833 \ifx\newcatcodetable\@undefined
4834 \def\babelcatcodetablenum{5211}
4835 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4836 \else
4837 \newcatcodetable\babelcatcodetablenum
4838 \newcatcodetable\bbl@pattcodes
4839 \fi
4840 \else
4841 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4842 \fi
4843 \def\bbl@luapatterns#1#2{%
4844 \bbl@get@enc#1::@@@
4845 \setbox\z@\hbox\bgroup
4846 \begingroup
4847 \savecatcodetable\babelcatcodetablenum\relax
4848 \initcatcodetable\bbl@pattcodes\relax
4849 \catcodetable\bbl@pattcodes\relax
4850 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4851 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4852 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4853 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4854 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4855 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4856 \input #1\relax
4857 \catcodetable\babelcatcodetablenum\relax
4858 \endgroup
4859 \def\bbl@tempa{#2}%
4860 \ifx\bbl@tempa\@empty\else
4861 \input #2\relax
4862 \fi
4863 \egroup}%
4864 \def\bbl@patterns@lua#1{%
4865 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4866 \csname l@#1\endcsname
4867 \edef\bbl@tempa{#1}%
4868 \else

```

```

4869 \csname l@#1:\f@encoding\endcsname
4870 \edef\bbl@tempa{#1:\f@encoding}%
4871 \fi\relax
4872 \@namedef{lu@texhyphen@loaded@the\language{}}% Temp
4873 \@ifundefined{bbl@hyphendata@the\language}%
4874 {\def\bbl@elt##1##2##3##4{%
4875 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4876 \def\bbl@tempb{##3}%
4877 \ifx\bbl@tempb@empty\else % if not a synonymous
4878 \def\bbl@tempc{##3}{##4}}%
4879 \fi
4880 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4881 \fi}%
4882 \bbl@languages
4883 \@ifundefined{bbl@hyphendata@the\language}%
4884 {\bbl@info{No hyphenation patterns were set for\%
4885 language '\bbl@tempa'. Reported}}%
4886 {\expandafter\expandafter\expandafter\bbl@luapatterns
4887 \csname bbl@hyphendata@the\language\endcsname}}}%
4888 \endinput\fi
4889 % Here ends \ifx\AddBabelHook\@undefined
4890 % A few lines are only read by hyphen.cfg
4891 \ifx\DisableBabelHook\@undefined
4892 \AddBabelHook{luatex}{everylanguage}{%
4893 \def\process@language##1##2##3{%
4894 \def\process@line####1####2 ####3 ####4 {}}}
4895 \AddBabelHook{luatex}{loadpatterns}{%
4896 \input #1\relax
4897 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4898 {{#1}}}%
4899 \AddBabelHook{luatex}{loadexceptions}{%
4900 \input #1\relax
4901 \def\bbl@tempb##1##2{{#1}{#1}}%
4902 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4903 {\expandafter\expandafter\expandafter\bbl@tempb
4904 \csname bbl@hyphendata@the\language\endcsname}}
4905 \endinput\fi
4906 % Here stops reading code for hyphen.cfg
4907 % The following is read the 2nd time it's loaded
4908 \begingroup % TODO - to a lua file
4909 \catcode`\%=12
4910 \catcode`\'=12
4911 \catcode`\%=12
4912 \catcode`\:=12
4913 \directlua{
4914 Babel = Babel or {}
4915 function Babel.bytes(line)
4916 return line:gsub("(.)",
4917 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4918 end
4919 function Babel.begin_process_input()
4920 if luatexbase and luatexbase.add_to_callback then
4921 luatexbase.add_to_callback('process_input_buffer',
4922 Babel.bytes, 'Babel.bytes')
4923 else
4924 Babel.callback = callback.find('process_input_buffer')
4925 callback.register('process_input_buffer', Babel.bytes)
4926 end
4927 end

```

```

4928 function Babel.end_process_input ()
4929   if luatexbase and luatexbase.remove_from_callback then
4930     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4931   else
4932     callback.register('process_input_buffer', Babel.callback)
4933   end
4934 end
4935 function Babel.addpatterns(pp, lg)
4936   local lg = lang.new(lg)
4937   local pats = lang.patterns(lg) or ''
4938   lang.clear_patterns(lg)
4939   for p in pp:gmatch('[^%s]+') do
4940     ss = ''
4941     for i in string.utfcharacters(p:gsub('%d', '')) do
4942       ss = ss .. '%d?' .. i
4943     end
4944     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4945     ss = ss:gsub('%.%%d%?$', '%%.')
4946     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4947     if n == 0 then
4948       tex.sprint(
4949         [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4950         .. p .. [[]]])
4951       pats = pats .. ' ' .. p
4952     else
4953       tex.sprint(
4954         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4955         .. p .. [[]]])
4956     end
4957   end
4958   lang.patterns(lg, pats)
4959 end
4960 }
4961 \endgroup
4962 \ifx\newattribute\@undefined\else
4963   \newattribute\bbl@attr@locale
4964   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4965   \AddBabelHook{luatex}{beforeextras}{%
4966     \setattribute\bbl@attr@locale\localeid}
4967 \fi
4968 \def\BabelStringsDefault{unicode}
4969 \let\luabbl@stop\relax
4970 \AddBabelHook{luatex}{encodedcommands}{%
4971   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4972   \ifx\bbl@tempa\bbl@tempb\else
4973     \directlua{Babel.begin_process_input()}%
4974     \def\luabbl@stop{%
4975       \directlua{Babel.end_process_input()}}%
4976   \fi}%
4977 \AddBabelHook{luatex}{stopcommands}{%
4978   \luabbl@stop
4979   \let\luabbl@stop\relax}
4980 \AddBabelHook{luatex}{patterns}{%
4981   \ifundefined\bbl@hyphendata@the\language}%
4982   {\def\bbl@elt##1##2##3##4{%
4983     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
4984     \def\bbl@tempb{##3}%
4985     \ifx\bbl@tempb@empty\else % if not a synonymous
4986       \def\bbl@tempc{##3}{##4}%

```

```

4987         \fi
4988         \bbl@csarg\undef{hyphendata@##2}{\bbl@tempc}%
4989     \fi}%
4990 \bbl@languages
4991 \@ifundefined{bbl@hyphendata@the\language}%
4992     {\bbl@info{No hyphenation patterns were set for\%
4993         language '#2'. Reported}}%
4994     {\expandafter\expandafter\expandafter\bbl@luapatterns
4995         \csname bbl@hyphendata@the\language\endcsname}}}%
4996 \@ifundefined{bbl@patterns@}{}%
4997     \begingroup
4998     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4999     \ifin\else
5000         \ifx\bbl@patterns@\empty\else
5001             \directlua{ Babel.addpatterns(
5002                 [[\bbl@patterns@]], \number\language) }%
5003         \fi
5004         \@ifundefined{bbl@patterns@#1}%
5005             \empty
5006             {\directlua{ Babel.addpatterns(
5007                 [[\space\csname bbl@patterns@#1\endcsname]],
5008                 \number\language) }}%
5009         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5010     \fi
5011 \endgroup}%
5012 \bbl@exp{%
5013     \bbl@ifunset{bbl@prehc@\language\name}{}%
5014     {\bbl@ifblank{\bbl@cs{prehc@\language\name}}{}}%
5015     {\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.

```

5016 \@onlypreamble\babelpatterns
5017 \AtEndOfPackage{%
5018     \newcommand\babelpatterns[2][\empty]{%
5019         \ifx\bbl@patterns@\relax
5020             \let\bbl@patterns@\empty
5021         \fi
5022         \ifx\bbl@pttnlist\empty\else
5023             \bbl@warning{%
5024                 You must not intermingle \string\selectlanguage\space and\%
5025                 \string\babelpatterns\space or some patterns will not\%
5026                 be taken into account. Reported}%
5027             \fi
5028             \ifx\empty#1%
5029                 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5030             \else
5031                 \edef\bbl@tempb{\zap@space#1 \empty}%
5032                 \bbl@for\bbl@tempa\bbl@tempb{%
5033                     \bbl@fixname\bbl@tempa
5034                     \bbl@iflanguage\bbl@tempa{%
5035                         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5036                             \@ifundefined{bbl@patterns@\bbl@tempa}%
5037                                 \empty
5038                                 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5039                             #2}}}%
5040             \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.

```
5041% TODO - to a lua file
5042 \directlua{
5043   Babel = Babel or {}
5044   Babel.linebreaking = Babel.linebreaking or {}
5045   Babel.linebreaking.before = {}
5046   Babel.linebreaking.after = {}
5047   Babel.locale = {} % Free to use, indexed with \localeid
5048   function Babel.linebreaking.add_before(func)
5049     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5050     table.insert(Babel.linebreaking.before, func)
5051   end
5052   function Babel.linebreaking.add_after(func)
5053     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5054     table.insert(Babel.linebreaking.after, func)
5055   end
5056 }
5057 \def\bbl@intraspace#1 #2 #3\@{
5058   \directlua{
5059     Babel = Babel or {}
5060     Babel.intraspaces = Babel.intraspaces or {}
5061     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5062       {b = #1, p = #2, m = #3}
5063     Babel.locale_props[\the\localeid].intraspace = %
5064       {b = #1, p = #2, m = #3}
5065   }}
5066 \def\bbl@intrapenalty#1\@{
5067   \directlua{
5068     Babel = Babel or {}
5069     Babel.intrapenalties = Babel.intrapenalties or {}
5070     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5071     Babel.locale_props[\the\localeid].intrapenalty = #1
5072   }}
5073 \begingroup
5074 \catcode`\%=12
5075 \catcode`\^=14
5076 \catcode`\'=12
5077 \catcode`\~=12
5078 \gdef\bbl@seaintraspace{^
5079   \let\bbl@seaintraspace\relax
5080   \directlua{
5081     Babel = Babel or {}
5082     Babel.sea_enabled = true
5083     Babel.sea_ranges = Babel.sea_ranges or {}
5084     function Babel.set_chranges (script, chrng)
5085       local c = 0
5086       for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5087         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5088         c = c + 1
5089       end
5090     end
5091     function Babel.sea_disc_to_space (head)
5092       local sea_ranges = Babel.sea_ranges
5093       local last_char = nil
```

```

5094     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5095     for item in node.traverse(head) do
5096         local i = item.id
5097         if i == node.id'glyph' then
5098             last_char = item
5099         elseif i == 7 and item.subtype == 3 and last_char
5100             and last_char.char > 0x0C99 then
5101             quad = font.getfont(last_char.font).size
5102             for lg, rg in pairs(sea_ranges) do
5103                 if last_char.char > rg[1] and last_char.char < rg[2] then
5104                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5105                     local intraspace = Babel.intraspaces[lg]
5106                     local intrapenalty = Babel.intrapenalties[lg]
5107                     local n
5108                     if intrapenalty ~= 0 then
5109                         n = node.new(14, 0)      ^% penalty
5110                         n.penalty = intrapenalty
5111                         node.insert_before(head, item, n)
5112                     end
5113                     n = node.new(12, 13)      ^% (glue, spaceskip)
5114                     node.setglue(n, intraspace.b * quad,
5115                                 intraspace.p * quad,
5116                                 intraspace.m * quad)
5117                     node.insert_before(head, item, n)
5118                     node.remove(head, item)
5119                 end
5120             end
5121         end
5122     end
5123 end
5124 }^^
5125 \bbl@luahyphenate}
5126 \catcode`\%=14
5127 \gdef\bbl@cjkintraspaces{%
5128 \let\bbl@cjkintraspaces\relax
5129 \directlua{
5130     Babel = Babel or {}
5131     require('babel-data-cjk.lua')
5132     Babel.cjk_enabled = true
5133     function Babel.cjk_linebreak(head)
5134         local GLYPH = node.id'glyph'
5135         local last_char = nil
5136         local quad = 655360      % 10 pt = 655360 = 10 * 65536
5137         local last_class = nil
5138         local last_lang = nil
5139
5140         for item in node.traverse(head) do
5141             if item.id == GLYPH then
5142
5143                 local lang = item.lang
5144
5145                 local LOCALE = node.get_attribute(item,
5146                     luatexbase.registernumber'bbl@attr@locale')
5147                 local props = Babel.locale_props[LOCALE]
5148
5149                 local class = Babel.cjk_class[item.char].c
5150
5151                 if class == 'cp' then class = 'cl' end % )] as CL
5152                 if class == 'id' then class = 'I' end

```



```

5153
5154     local br = 0
5155     if class and last_class and Babel.cjk_breaks[last_class][class] then
5156         br = Babel.cjk_breaks[last_class][class]
5157     end
5158
5159     if br == 1 and props.linebreak == 'c' and
5160         lang ~= \the\l@nohyphenation\space and
5161         last_lang ~= \the\l@nohyphenation then
5162         local intrapenalty = props.intrapenalty
5163         if intrapenalty ~= 0 then
5164             local n = node.new(14, 0)    % penalty
5165             n.penalty = intrapenalty
5166             node.insert_before(head, item, n)
5167         end
5168         local intraspace = props.intraspace
5169         local n = node.new(12, 13)    % (glue, spaceskip)
5170         node.setglue(n, intraspace.b * quad,
5171             intraspace.p * quad,
5172             intraspace.m * quad)
5173         node.insert_before(head, item, n)
5174     end
5175
5176     if font.getfont(item.font) then
5177         quad = font.getfont(item.font).size
5178     end
5179     last_class = class
5180     last_lang = lang
5181     else % if penalty, glue or anything else
5182         last_class = nil
5183     end
5184 end
5185 lang.hyphenate(head)
5186 end
5187 }%
5188 \bbl@luahyphenate}
5189 \gdef\bbl@luahyphenate{%
5190 \let\bbl@luahyphenate\relax
5191 \directlua{
5192     luatexbase.add_to_callback('hyphenate',
5193     function (head, tail)
5194         if Babel.linebreaking.before then
5195             for k, func in ipairs(Babel.linebreaking.before) do
5196                 func(head)
5197             end
5198         end
5199         if Babel.cjk_enabled then
5200             Babel.cjk_linebreak(head)
5201         end
5202         lang.hyphenate(head)
5203         if Babel.linebreaking.after then
5204             for k, func in ipairs(Babel.linebreaking.after) do
5205                 func(head)
5206             end
5207         end
5208         if Babel.sea_enabled then
5209             Babel.sea_disc_to_space(head)
5210         end
5211     end,

```

```

5212 'Babel.hyphenate')
5213 }
5214 }
5215 \endgroup
5216 \def\bbl@provide@intraspace{%
5217   \bbl@ifunset{\bbl@intsp@{language}}{%
5218     {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5219       \bbl@xin@{\bbl@cl{lbrk}}{c}%
5220       \ifin@           % cjk
5221       \bbl@cjk@intraspace
5222       \directlua{
5223         Babel = Babel or {}
5224         Babel.locale_props = Babel.locale_props or {}
5225         Babel.locale_props[\the\localeid].linebreak = 'c'
5226       }%
5227       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5228       \ifx\bbl@KVP@intrapenalty\@nil
5229         \bbl@intrapenalty0\@@
5230       \fi
5231     \else           % sea
5232       \bbl@sea@intraspace
5233       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5234       \directlua{
5235         Babel = Babel or {}
5236         Babel.sea_ranges = Babel.sea_ranges or {}
5237         Babel.set_chranges('\bbl@cl{sbc}',
5238                           '\bbl@cl{chrng}')
5239       }%
5240       \ifx\bbl@KVP@intrapenalty\@nil
5241         \bbl@intrapenalty0\@@
5242       \fi
5243     \fi
5244   \fi
5245   \ifx\bbl@KVP@intrapenalty\@nil\else
5246     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5247   \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.

```

5248 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5249 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5250 \DisableBabelHook{babel-fontspec}
5251 <<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.

```

5252 % TODO - to a lua file
5253 \directlua{
5254 Babel.script_blocks = {
5255   ['dflt'] = {},
5256   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5257             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5258   ['Armn'] = {{0x0530, 0x058F}},
5259   ['Beng'] = {{0x0980, 0x09FF}},
5260   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5261   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5262   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5263             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5264   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5265   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5266             {0xAB00, 0xAB2F}},
5267   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5268   % Don't follow strictly Unicode, which places some Coptic letters in
5269   % the 'Greek and Coptic' block
5270   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5271   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5272             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5273             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5274             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5275             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5276             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5277   ['Hebr'] = {{0x0590, 0x05FF}},
5278   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5279             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5280   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5281   ['Knda'] = {{0x0C80, 0x0CFF}},
5282   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5283             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5284             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5285   ['Lao0'] = {{0x0E80, 0x0EFF}},
5286   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5287             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5288             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5289   ['Mahj'] = {{0x11150, 0x1117F}},
5290   ['Mlym'] = {{0x0D00, 0x0D7F}},
5291   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5292   ['Orya'] = {{0x0B00, 0x0B7F}},
5293   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5294   ['Syr1'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5295   ['Taml'] = {{0x0B80, 0x0BFF}},
5296   ['Telu'] = {{0x0C00, 0x0C7F}},
5297   ['Tfng'] = {{0x2D30, 0x2D7F}},
5298   ['Thai'] = {{0x0E00, 0x0E7F}},
5299   ['Tibt'] = {{0x0F00, 0x0FFF}},
5300   ['Vaii'] = {{0xA500, 0xA63F}},
5301   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5302 }
5303
5304 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyr1
5305 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5306 Babel.script_blocks.Kana = Babel.script_blocks.Jpan

```

```

5307
5308 function Babel.locale_map(head)
5309   if not Babel.locale_mapped then return head end
5310
5311   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5312   local GLYPH = node.id('glyph')
5313   local inmath = false
5314   local toloc_save
5315   for item in node.traverse(head) do
5316     local toloc
5317     if not inmath and item.id == GLYPH then
5318       % Optimization: build a table with the chars found
5319       if Babel.chr_to_loc[item.char] then
5320         toloc = Babel.chr_to_loc[item.char]
5321       else
5322         for lc, maps in pairs(Babel.loc_to_scr) do
5323           for _, rg in pairs(maps) do
5324             if item.char >= rg[1] and item.char <= rg[2] then
5325               Babel.chr_to_loc[item.char] = lc
5326               toloc = lc
5327               break
5328             end
5329           end
5330         end
5331       end
5332       % Now, take action, but treat composite chars in a different
5333       % fashion, because they 'inherit' the previous locale. Not yet
5334       % optimized.
5335       if not toloc and
5336         (item.char >= 0x0300 and item.char <= 0x036F) or
5337         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5338         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5339         toloc = toloc_save
5340       end
5341       if toloc and toloc > -1 then
5342         if Babel.locale_props[toloc].lg then
5343           item.lang = Babel.locale_props[toloc].lg
5344           node.set_attribute(item, LOCALE, toloc)
5345         end
5346         if Babel.locale_props[toloc]['/'..item.font] then
5347           item.font = Babel.locale_props[toloc]['/'..item.font]
5348         end
5349         toloc_save = toloc
5350       end
5351     elseif not inmath and item.id == 7 then
5352       item.replace = item.replace and Babel.locale_map(item.replace)
5353       item.pre = item.pre and Babel.locale_map(item.pre)
5354       item.post = item.post and Babel.locale_map(item.post)
5355     elseif item.id == node.id'math' then
5356       inmath = (item.subtype == 0)
5357     end
5358   end
5359   return head
5360 end
5361 }

```

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

```

5362 \newcommand\babelcharproperty[1]{%

```

```

5363 \count@=#1\relax
5364 \ifvmode
5365 \expandafter\bb1@chprop
5366 \else
5367 \bb1@error{\string\babelcharproperty\space can be used only in\%
5368             vertical mode (preamble or between paragraphs)}%
5369             {See the manual for futher info}%
5370 \fi}
5371 \newcommand\bb1@chprop[3][\the\count@]{%
5372 \@tempcnta=#1\relax
5373 \bb1@ifunset{\bb1@chprop@#2}%
5374 {\bb1@error{No property named '#2'. Allowed values are\%
5375             direction (bc), mirror (bmg), and linebreak (lb)}%
5376             {See the manual for futher info}}%
5377 }%
5378 \loop
5379 \bb1@cs{chprop@#2}{#3}%
5380 \ifnum\count@<\@tempcnta
5381 \advance\count@\@ne
5382 \repeat}
5383 \def\bb1@chprop@direction#1{%
5384 \directlua{
5385     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5386     Babel.characters[\the\count@]['d'] = '#1'
5387 }}
5388 \let\bb1@chprop@bc\bb1@chprop@direction
5389 \def\bb1@chprop@mirror#1{%
5390 \directlua{
5391     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5392     Babel.characters[\the\count@]['m'] = '\number#1'
5393 }}
5394 \let\bb1@chprop@bmg\bb1@chprop@mirror
5395 \def\bb1@chprop@linebreak#1{%
5396 \directlua{
5397     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5398     Babel.cjk_characters[\the\count@]['c'] = '#1'
5399 }}
5400 \let\bb1@chprop@lb\bb1@chprop@linebreak
5401 \def\bb1@chprop@locale#1{%
5402 \directlua{
5403     Babel.chr_to_loc = Babel.chr_to_loc or {}
5404     Babel.chr_to_loc[\the\count@] =
5405         \bb1@ifblank{#1}{-1000}{\the\bb1@cs{id@#1}}\space
5406 }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

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

```

5407 \begingroup % TODO - to a lua file

```

```

5408 \catcode`\~ = 12
5409 \catcode`\# = 12
5410 \catcode`\% = 12
5411 \catcode`\& = 14
5412 \directlua{
5413   Babel.linebreaking.replacements = {}
5414   Babel.linebreaking.replacements[0] = {}  %% pre
5415   Babel.linebreaking.replacements[1] = {}  %% post
5416
5417   %% Discretionaries contain strings as nodes
5418   function Babel.str_to_nodes(fn, matches, base)
5419     local n, head, last
5420     if fn == nil then return nil end
5421     for s in string.utfvalues(fn(matches)) do
5422       if base.id == 7 then
5423         base = base.replace
5424       end
5425       n = node.copy(base)
5426       n.char = s
5427       if not head then
5428         head = n
5429       else
5430         last.next = n
5431       end
5432       last = n
5433     end
5434     return head
5435   end
5436
5437   Babel.fetch_subtext = {}
5438
5439   %% Merging both functions doesn't seem feasible, because there are too
5440   %% many differences.
5441   Babel.fetch_subtext[0] = function(head)
5442     local word_string = ''
5443     local word_nodes = {}
5444     local lang
5445     local item = head
5446     local inmath = false
5447
5448     while item do
5449
5450       if item.id == 11 then
5451         inmath = (item.subtype == 0)
5452       end
5453
5454       if inmath then
5455         %% pass
5456
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
5462             lang = lang or locale
5463             word_string = word_string .. unicode.utf8.char(item.char)
5464             word_nodes[#word_nodes+1] = item
5465           end
5466         else

```

```

5467         break
5468     end
5469
5470     elseif item.id == 12 and item.subtype == 13 then
5471         word_string = word_string .. '|'
5472         word_nodes[#word_nodes+1] = item
5473
5474         %% Ignore leading unrecognized nodes, too.
5475         elseif word_string ~= '' then
5476             word_string = word_string .. Babel.us_char
5477             word_nodes[#word_nodes+1] = item %% Will be ignored
5478         end
5479
5480         item = item.next
5481     end
5482
5483     %% Here and above we remove some trailing chars but not the
5484     %% corresponding nodes. But they aren't accessed.
5485     if word_string:sub(-1) == '|' then
5486         word_string = word_string:sub(1,-2)
5487     end
5488     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5489     return word_string, word_nodes, item, lang
5490 end
5491
5492 Babel.fetch_subtext[1] = function(head)
5493     local word_string = ''
5494     local word_nodes = {}
5495     local lang
5496     local item = head
5497     local inmath = false
5498
5499     while item do
5500
5501         if item.id == 11 then
5502             inmath = (item.subtype == 0)
5503         end
5504
5505         if inmath then
5506             %% pass
5507         end
5508
5509         elseif item.id == 29 then
5510             if item.lang == lang or lang == nil then
5511                 if (item.char ~= 124) and (item.char ~= 61) then %% not =, not |
5512                     lang = lang or item.lang
5513                     word_string = word_string .. unicode.utf8.char(item.char)
5514                     word_nodes[#word_nodes+1] = item
5515                 end
5516             else
5517                 break
5518             end
5519
5520             elseif item.id == 7 and item.subtype == 2 then
5521                 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

```

```

5526
5527     %% (1) Go to next word if nothing was found, and (2) implicitly
5528     %% remove leading USs.
5529     elseif word_string == '' then
5530         %% pass
5531
5532     %% This is the responsible for splitting by words.
5533     elseif (item.id == 12 and item.subtype == 13) then
5534         break
5535
5536     else
5537         word_string = word_string .. Babel.us_char
5538         word_nodes[#word_nodes+1] = item %% Will be ignored
5539     end
5540
5541     item = item.next
5542 end
5543
5544 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5545 return word_string, word_nodes, item, lang
5546 end
5547
5548 function Babel.pre_hyphenate_replace(head)
5549     Babel.hyphenate_replace(head, 0)
5550 end
5551
5552 function Babel.post_hyphenate_replace(head)
5553     Babel.hyphenate_replace(head, 1)
5554 end
5555
5556 Babel.us_char = string.char(31)
5557
5558 function Babel.hyphenate_replace(head, mode)
5559     local u = unicode.utf8
5560     local lbkr = Babel.linebreaking.replacements[mode]
5561
5562     local word_head = head
5563
5564     while true do %% for each subtext block
5565
5566         local w, wn, nw, lang = Babel.fetch_subtext[mode](word_head)
5567
5568         if Babel.debug then
5569             print()
5570             print('@@@@', w, nw)
5571         end
5572
5573         if nw == nil and w == '' then break end
5574
5575         if not lang then goto next end
5576         if not lbkr[lang] then goto next end
5577
5578         %% For each saved (pre|post)hyphenation. TODO. Reconsider how
5579         %% loops are nested.
5580         for k=1, #lbkr[lang] do
5581             local p = lbkr[lang][k].pattern
5582             local r = lbkr[lang][k].replace
5583
5584             if Babel.debug then

```



```

5585         print('====', p, mode)
5586     end
5587
5588     %% This variable is set in some cases below to the first *byte*
5589     %% after the match, either as found by u.match (faster) or the
5590     %% computed position based on sc if w has changed.
5591     local last_match = 0
5592
5593     %% For every match.
5594     while true do
5595         if Babel.debug then
5596             print('-----')
5597         end
5598         local new    %% used when inserting and removing nodes
5599         local refetch = false
5600
5601         local matches = { u.match(w, p, last_match) }
5602         if #matches < 2 then break end
5603
5604         %% Get and remove empty captures (with ())'s, which return a
5605         %% number with the position), and keep actual captures
5606         %% (from (...)), if any, in matches.
5607         local first = table.remove(matches, 1)
5608         local last  = table.remove(matches, #matches)
5609         %% Non re-fetched substrings may contain \31, which separates
5610         %% subsubstrings.
5611         if string.find(w:sub(first, last-1), Babel.us_char) then break end
5612
5613         local save_last = last %% with A()BC()D, points to D
5614
5615         %% Fix offsets, from bytes to unicode. Explained above.
5616         first = u.len(w:sub(1, first-1)) + 1
5617         last  = u.len(w:sub(1, last-1)) %% now last points to C
5618
5619         if Babel.debug then
5620             print(p)
5621             print('', 'sc', 'first', 'last', 'last_m', 'w')
5622         end
5623
5624         %% This loop traverses the matched substring and takes the
5625         %% corresponding action stored in the replacement list.
5626         %% sc = the position in substr nodes / string
5627         %% rc = the replacement table index
5628         local sc = first-1
5629         local rc = 0
5630         while rc < last-first+1 do %% for each replacement
5631             if Babel.debug then
5632                 print('.....')
5633             end
5634             sc = sc + 1
5635             rc = rc + 1
5636             local crep = r[rc]
5637             local char_node = wn[sc]
5638             local char_base = char_node
5639             local end_replacement = false
5640
5641             if crep and crep.data then
5642                 char_base = wn[crep.data+first-1]
5643             end

```

```

5644
5645     if Babel.debug then
5646         print('*', sc, first, last, last_match, w)
5647     end
5648
5649     if crep and next(crep) == nil then &% {}
5650         last_match = save_last
5651
5652     elseif crep == nil then &% remove
5653         node.remove(head, char_node)
5654         table.remove(wn, sc)
5655         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5656         last_match = utf8.offset(w, sc)
5657         sc = sc - 1 &% Nothing has been inserted
5658
5659     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
5660         local d = node.new(7, 0) &% (disc, discretionary)
5661         d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5662         d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5663         d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5664         d.attr = char_base.attr
5665         if crep.pre == nil then &% TeXbook p96
5666             d.penalty = crep.penalty or tex.hyphenpenalty
5667         else
5668             d.penalty = crep.penalty or tex.exhyphenpenalty
5669         end
5670         head, new = node.insert_before(head, char_node, d)
5671         end_replacement = true
5672
5673     elseif crep and crep.penalty then
5674         local d = node.new(14, 0) &% (penalty, userpenalty)
5675         d.attr = char_base.attr
5676         d.penalty = crep.penalty
5677         head, new = node.insert_before(head, char_node, d)
5678         end_replacement = true
5679
5680     elseif crep and crep.string then
5681         local str = crep.string(matches)
5682         if str == '' then &% Gather with nil
5683             refetch = true
5684             if sc == 1 then
5685                 word_head = char_node.next
5686             end
5687             head, new = node.remove(head, char_node)
5688         elseif char_node.id == 29 and u.len(str) == 1 then
5689             char_node.char = string.utfvalue(str)
5690             w = u.sub(w, 1, sc-1) .. str .. u.sub(w, sc+1)
5691             last_match = utf8.offset(w, sc+1)
5692         else
5693             refetch = true
5694             local n
5695             for s in string.utfvalues(str) do
5696                 if char_node.id == 7 then
5697                     &% TODO. Remove this limitation.
5698                     texio.write_nl('Automatic hyphens cannot be replaced, just removed.')
5699                 else
5700                     n = node.copy(char_base)
5701                     end
5702                     n.char = s

```

```

5703         if sc == 1 then
5704             head, new = node.insert_before(head, char_node, n)
5705             word_head = new
5706         else
5707             node.insert_before(head, char_node, n)
5708         end
5709     end
5710     node.remove(head, char_node)
5711 end    %% string length
5712 end    %% if char and char.string (ie replacement cases)
5713
5714 %% Shared by disc and penalty.
5715 if end_replacement then
5716     if sc == 1 then
5717         word_head = new
5718     end
5719     if crep.insert then
5720         last_match = save_last
5721     else
5722         node.remove(head, char_node)
5723         w = u.sub(w, 1, sc-1) .. Babel.us_char .. u.sub(w, sc+1)
5724         last_match = utf8.offset(w, sc)
5725     end
5726 end
5727 end    %% for each replacement
5728
5729 if Babel.debug then
5730     print('/', sc, first, last, last_match, w)
5731 end
5732
5733 %% TODO. refetch will be eventually unnecessary.
5734 if refetch then
5735     w, wn, nw, lang = Babel.fetch_subtext[mode](word_head)
5736 end
5737
5738 end    %% for match
5739 end    %% for patterns
5740
5741 ::next::
5742     word_head = nw
5743 end    %% for substring
5744     return head
5745 end
5746
5747 %% This table stores capture maps, numbered consecutively
5748 Babel.capture_maps = {}
5749
5750 %% The following functions belong to the next macro
5751 function Babel.capture_func(key, cap)
5752     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[(") .. "]"
5753     ret = ret:gsub('{{[0-9]}|([^\]|+)|(-)}', Babel.capture_func_map)
5754     ret = ret:gsub("%[%[%]%.%.%", '')
5755     ret = ret:gsub("%%.%[%[%]%.%.%", '')
5756     return key .. "[=function(m) return ]] .. ret .. [[ end]]"
5757 end
5758
5759 function Babel.capt_map(from, mapno)
5760     return Babel.capture_maps[mapno][from] or from
5761 end

```

```

5762
5763 &% Handle the {n|abc|ABC} syntax in captures
5764 function Babel.capture_func_map(capno, from, to)
5765     local froms = {}
5766     for s in string.utfcharacters(from) do
5767         table.insert(froms, s)
5768     end
5769     local cnt = 1
5770     table.insert(Babel.capture_maps, {})
5771     local mlen = table.getn(Babel.capture_maps)
5772     for s in string.utfcharacters(to) do
5773         Babel.capture_maps[mlen][froms[cnt]] = s
5774         cnt = cnt + 1
5775     end
5776     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5777         (mlen) .. ").. " .. "[["
5778 end
5779 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the {*n*} syntax. For example, `pre={1}{1}` becomes `function(m) return m[1]..m[1]..'-' end`, where *m* are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to *m*[1]. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

```

5780 \catcode`\#=6
5781 \gdef\babelposthyphenation#1#2#3{&%
5782     \bbl@activateposthyphen
5783     \beginngroup
5784     \def\babeltempa{\bbl@add@list\babeltempb}&%
5785     \let\babeltempb\@empty
5786     \bbl@foreach{#3}{&%
5787         \bbl@ifsamestring{##1}{remove}&%
5788         {\bbl@add@list\babeltempb{nil}}&%
5789         {\directlua{
5790             local rep = [[##1]]
5791             rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5792             rep = rep:gsub('^(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5793             rep = rep:gsub('^(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5794             rep = rep:gsub('^(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5795             rep = rep:gsub('^(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5796             tex.print([[string\babeltempa{}} .. rep .. [{}]])
5797         }}}&%
5798     \directlua{
5799         local lbkr = Babel.linebreaking.replacements[1]
5800         local u = unicode.utf8
5801         &% Convert pattern:
5802         local patt = string.gsub([=[#2]=], '%s', '')
5803         if not u.find(patt, '()', nil, true) then
5804             patt = '()' .. patt .. '()'
5805         end
5806         patt = string.gsub(patt, '%(%)^', '^()')
5807         patt = string.gsub(patt, '%$(%)', '()$')
5808         patt = u.gsub(patt, '{(.)}',
5809             function (n)
5810                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)

```

```

5811         end)
5812     lbr[\the\csname l@#1\endcsname] = lbr[\the\csname l@#1\endcsname] or {}
5813     table.insert(lbr[\the\csname l@#1\endcsname],
5814         { pattern = patt, replace = { \babeltempb } })
5815 }&%
5816 \endgroup}
5817 % TODO. Copypaste pattern.
5818 \gdef\babelprehyphenation#1#2#3{&%
5819     \bbl@activateprehyphen
5820     \begin{group}
5821     \def\babeltempa{\bbl@add@list\babeltempb}&%
5822     \let\babeltempb\empty
5823     \bbl@foreach{#3}{&%
5824         \bbl@ifsamestring{##1}{remove}&%
5825         {\bbl@add@list\babeltempb{nil}}&%
5826         {\directlua{
5827             local rep = {[##1]}
5828             rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5829             rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
5830             tex.print([[\\string\babeltempa{}}] .. rep .. [[]]{}]}
5831         }}&%
5832     \directlua{
5833         local lbr = Babel.linebreaking.replacements[0]
5834         local u = unicode.utf8
5835         &% Convert pattern:
5836         local patt = string.gsub([=[#2]=], '%s', '')
5837         if not u.find(patt, '()', nil, true) then
5838             patt = '()' .. patt .. '()'
5839         end
5840         &% patt = string.gsub(patt, '%(%)%', '^()')
5841         &% patt = string.gsub(patt, '([%^%])%$%$', '%1()$')
5842         patt = u.gsub(patt, '{(.)}',
5843             function (n)
5844                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5845             end)
5846         lbr[\the\csname bbl@id@@#1\endcsname] = lbr[\the\csname bbl@id@@#1\endcsname] or {}
5847         table.insert(lbr[\the\csname bbl@id@@#1\endcsname],
5848             { pattern = patt, replace = { \babeltempb } })
5849     }&%
5850     \endgroup}
5851 \endgroup
5852 \def\bbl@activateposthyphen{%
5853     \let\bbl@activateposthyphen\relax
5854     \directlua{
5855         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5856     }}
5857 \def\bbl@activateprehyphen{%
5858     \let\bbl@activateprehyphen\relax
5859     \directlua{
5860         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5861     }}

```

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.

```

5862 \bbl@trace{Redefinitions for bidi layout}
5863 \ifx\@eqnnum\undefined\else
5864   \ifx\bbl@attr@dir\undefined\else
5865     \edef\@eqnnum{%
5866       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5867       \unexpanded\expandafter{\@eqnnum}}%
5868   \fi
5869 \fi
5870 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5871 \ifnum\bbl@bidimode>\z@
5872   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5873     \bbl@exp{%
5874       \mathdir\the\bodydir
5875       #1% Once entered in math, set boxes to restore values
5876       \<ifmode>%
5877       \everyvbox{%
5878         \the\everyvbox
5879         \bodydir\the\bodydir
5880         \mathdir\the\mathdir
5881         \everyhbox{\the\everyhbox}%
5882         \everyvbox{\the\everyvbox}}%
5883       \everyhbox{%
5884         \the\everyhbox
5885         \bodydir\the\bodydir
5886         \mathdir\the\mathdir
5887         \everyhbox{\the\everyhbox}%
5888         \everyvbox{\the\everyvbox}}%
5889       \<fi>}}%
5890   \def\@hangfrom#1{%
5891     \setbox\@tempboxa\hbox{#1}%
5892     \hangindent\wd\@tempboxa
5893     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5894       \shapemode\@ne
5895     \fi
5896     \noindent\box\@tempboxa}
5897 \fi
5898 \IfBabelLayout{tabular}
5899   {\let\bbl@OL@tabular\@tabular
5900    \bbl@replace\@tabular{$}\bbl@nextfake$}%
5901   \let\bbl@NL@tabular\@tabular
5902   \AtBeginDocument{%
5903     \ifx\bbl@NL@tabular\@tabular\else
5904       \bbl@replace\@tabular{$}\bbl@nextfake$}%
5905     \let\bbl@NL@tabular\@tabular
5906   \fi}}
5907 {}
5908 \IfBabelLayout{lists}
5909   {\let\bbl@OL@list\list
5910    \bbl@sreplace\list{\parshape}\bbl@listparshape}%
5911   \let\bbl@NL@list\list
5912   \def\bbl@listparshape#1#2#3{%

```

```

5913     \parshape #1 #2 #3 %
5914     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5915         \shapemode\tw@
5916     \fi}}
5917 {}
5918 \IfBabelLayout{graphics}
5919 {\let\bbl@pictresetdir\relax
5920  \def\bbl@pictsetdir#1{%
5921      \ifcase\bbl@thetextdir
5922          \let\bbl@pictresetdir\relax
5923      \else
5924          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5925              \or\textdir TLT
5926              \else\bodydir TLT \textdir TLT
5927          \fi
5928          % \(\text|par)dir required in pgf:
5929          \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5930      \fi}%
5931  \ifx\AddToHook\@undefined\else
5932      \AddToHook{env/picture/begin}{\bbl@pictsetdir\z@}%
5933      \directlua{
5934          Babel.get_picture_dir = true
5935          Babel.picture_has_bidi = 0
5936          function Babel.picture_dir (head)
5937              if not Babel.get_picture_dir then return head end
5938              for item in node.traverse(head) do
5939                  if item.id == node.id'glyph' then
5940                      local itemchar = item.char
5941                      % TODO. Copypaste pattern from Babel.bidi (-r)
5942                      local chardata = Babel.characters[itemchar]
5943                      local dir = chardata and chardata.d or nil
5944                      if not dir then
5945                          for nn, et in ipairs(Babel.ranges) do
5946                              if itemchar < et[1] then
5947                                  break
5948                              elseif itemchar <= et[2] then
5949                                  dir = et[3]
5950                                  break
5951                              end
5952                          end
5953                      end
5954                      if dir and (dir == 'al' or dir == 'r') then
5955                          Babel.picture_has_bidi = 1
5956                      end
5957                  end
5958              end
5959              return head
5960          end
5961          luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5962              "Babel.picture_dir")
5963      }%
5964  \AtBeginDocument{%
5965      \long\def\put(#1,#2)#3{%
5966          \@killglue
5967          % Try:
5968          \ifx\bbl@pictresetdir\relax
5969              \def\bbl@tempc{0}%
5970          \else
5971              \directlua{

```

```

5972         Babel.get_picture_dir = true
5973         Babel.picture_has_bidi = 0
5974     }%
5975     \setbox\z@\hb@xt@\z@{%
5976         \@defaultunitsset\@tempdimc{#1}\unitlength
5977         \kern\@tempdimc
5978         #3\hss}%
5979     \edef\bb1@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
5980     \fi
5981     % Do:
5982     \@defaultunitsset\@tempdimc{#2}\unitlength
5983     \raise\@tempdimc\hb@xt@\z@{%
5984         \@defaultunitsset\@tempdimc{#1}\unitlength
5985         \kern\@tempdimc
5986         {\ifnum\bb1@tempc>\z@\bb1@pictresetdir\fi#3}\hss}%
5987     \ignorespaces}%
5988     \MakeRobust\put}%
5989 \fi
5990 \AtBeginDocument
5991 {\ifx\tikz@atbegin@node\undefined\else
5992     \let\bb1@OL@pgfpicture\pgfpicture
5993     \bb1@sreplace\pgfpicture{\pgfpicturetrue}%
5994     {\bb1@pictsetdir\z@\pgfpicturetrue}%
5995     \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir\z@}%
5996     \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5997     \bb1@sreplace\tikz{\beginpgroup}%
5998     {\beginpgroup\bb1@pictsetdir\tw@}%
5999     \fi
6000     \ifx\AddToHook\undefined\else
6001         \AddToHook{env/tcolorbox/begin}{\bb1@pictsetdir\@ne}%
6002     \fi
6003     }}
6004 {}

```

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.

```

6005 \IfBabelLayout{counters}%
6006 {\let\bb1@OL@@textsuperscript\textsuperscript
6007  \bb1@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6008  \let\bb1@latinarabic=\@arabic
6009  \let\bb1@OL@@arabic\@arabic
6010  \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
6011  \@ifpackagewith{babel}{bidi=default}%
6012  {\let\bb1@asciroman=\@roman
6013   \let\bb1@OL@@roman\@roman
6014   \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
6015   \let\bb1@asciiRoman=\@Roman
6016   \let\bb1@OL@@roman\@Roman
6017   \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}%
6018   \let\bb1@OL@labelenumii\labelenumii
6019   \def\labelenumii{}\theenumii}%
6020   \let\bb1@OL@p@enumiii\p@enumiii
6021   \def\p@enumiii{\p@enumii}\theenumii{}\{}\}%
6022  <<Footnote changes>>
6023 \IfBabelLayout{footnotes}%
6024 {\let\bb1@OL@footnote\footnote
6025  \BabelFootnote\footnote\languagename{}\}%
6026  \BabelFootnote\localfootnote\languagename{}\}%

```



```

6027 \BabelFootnote\mainfootnote{}{}{}
6028 {}

```

Some \TeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

6029 \IfBabelLayout{extras}%
6030 {\let\bbl@OL@underline\underline
6031 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6032 \let\bbl@OL@LaTeX2e\LaTeX2e
6033 \DeclareRobustCommand{\LaTeXe}{\mbox{\math
6034 \if b\expandafter\@car\f@series\@nil\boldmath\fi
6035 \babelsublr}%
6036 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}}}}
6037 {}
6038 \luatex

```

13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

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

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

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

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

```

6039 (*basic-r)
6040 Babel = Babel or {}
6041
6042 Babel.bidi_enabled = true
6043

```

```

6044 require('babel-data-bidi.lua')
6045
6046 local characters = Babel.characters
6047 local ranges = Babel.ranges
6048
6049 local DIR = node.id("dir")
6050
6051 local function dir_mark(head, from, to, outer)
6052   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6053   local d = node.new(DIR)
6054   d.dir = '+' .. dir
6055   node.insert_before(head, from, d)
6056   d = node.new(DIR)
6057   d.dir = '-' .. dir
6058   node.insert_after(head, to, d)
6059 end
6060
6061 function Babel.bidi(head, ispar)
6062   local first_n, last_n          -- first and last char with nums
6063   local last_es                  -- an auxiliary 'last' used with nums
6064   local first_d, last_d          -- first and last char in L/R block
6065   local dir, dir_real

```

Next also depends on script/lang (<al>/<r>). To be set by babel. tex.pardir is dangerous, could be (re)set but it should be changed only in vmode. There are two strong's – strong = l/al/r and strong_lr = l/r (there must be a better way):

```

6066   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6067   local strong_lr = (strong == 'l') and 'l' or 'r'
6068   local outer = strong
6069
6070   local new_dir = false
6071   local first_dir = false
6072   local inmath = false
6073
6074   local last_lr
6075
6076   local type_n = ''
6077
6078   for item in node.traverse(head) do
6079
6080     -- three cases: glyph, dir, otherwise
6081     if item.id == node.id'glyph'
6082       or (item.id == 7 and item.subtype == 2) then
6083
6084       local itemchar
6085       if item.id == 7 and item.subtype == 2 then
6086         itemchar = item.replace.char
6087       else
6088         itemchar = item.char
6089       end
6090       local chardata = characters[itemchar]
6091       dir = chardata and chardata.d or nil
6092       if not dir then
6093         for nn, et in ipairs(ranges) do
6094           if itemchar < et[1] then
6095             break
6096           elseif itemchar <= et[2] then
6097             dir = et[3]
6098             break

```

```

6099         end
6100     end
6101 end
6102 dir = dir or 'l'
6103 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).

```

6104     if new_dir then
6105         attr_dir = 0
6106         for at in node.traverse(item.attr) do
6107             if at.number == luatexbase.registernumber'bbl@attr@dir' then
6108                 attr_dir = at.value % 3
6109             end
6110         end
6111         if attr_dir == 1 then
6112             strong = 'r'
6113         elseif attr_dir == 2 then
6114             strong = 'al'
6115         else
6116             strong = 'l'
6117         end
6118         strong_lr = (strong == 'l') and 'l' or 'r'
6119         outer = strong_lr
6120         new_dir = false
6121     end
6122
6123     if dir == 'nsm' then dir = strong end -- W1

```

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

```

6124     dir_real = dir -- We need dir_real to set strong below
6125     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:

```

6126     if strong == 'al' then
6127         if dir == 'en' then dir = 'an' end -- W2
6128         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6129         strong_lr = 'r' -- W3
6130     end

```

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

```

6131     elseif item.id == node.id'dir' and not inmath then
6132         new_dir = true
6133         dir = nil
6134     elseif item.id == node.id'math' then
6135         inmath = (item.subtype == 0)
6136     else
6137         dir = nil -- Not a char
6138     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>.

```

6139   if dir == 'en' or dir == 'an' or dir == 'et' then
6140       if dir ~= 'et' then
6141           type_n = dir
6142       end
6143       first_n = first_n or item
6144       last_n = last_es or item
6145       last_es = nil
6146   elseif dir == 'es' and last_n then -- W3+W6
6147       last_es = item
6148   elseif dir == 'cs' then -- it's right - do nothing
6149   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6150       if strong_lr == 'r' and type_n ~= '' then
6151           dir_mark(head, first_n, last_n, 'r')
6152       elseif strong_lr == 'l' and first_d and type_n == 'an' then
6153           dir_mark(head, first_n, last_n, 'r')
6154           dir_mark(head, first_d, last_d, outer)
6155           first_d, last_d = nil, nil
6156       elseif strong_lr == 'l' and type_n ~= '' then
6157           last_d = last_n
6158       end
6159       type_n = ''
6160       first_n, last_n = nil, nil
6161   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:

```

6162   if dir == 'l' or dir == 'r' then
6163       if dir ~= outer then
6164           first_d = first_d or item
6165           last_d = item
6166       elseif first_d and dir ~= strong_lr then
6167           dir_mark(head, first_d, last_d, outer)
6168           first_d, last_d = nil, nil
6169       end
6170   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.

```

6171   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6172       item.char = characters[item.char] and
6173           characters[item.char].m or item.char
6174   elseif (dir or new_dir) and last_lr ~= item then
6175       local mir = outer .. strong_lr .. (dir or outer)
6176       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6177           for ch in node.traverse(node.next(last_lr)) do
6178               if ch == item then break end
6179               if ch.id == node.id'glyph' and characters[ch.char] then
6180                   ch.char = characters[ch.char].m or ch.char
6181               end
6182           end
6183       end
6184   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).

```

6185     if dir == 'l' or dir == 'r' then
6186         last_lr = item
6187         strong = dir_real          -- Don't search back - best save now
6188         strong_lr = (strong == 'l') and 'l' or 'r'
6189     elseif new_dir then
6190         last_lr = nil
6191     end
6192 end

```

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

```

6193 if last_lr and outer == 'r' then
6194     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6195         if characters[ch.char] then
6196             ch.char = characters[ch.char].m or ch.char
6197         end
6198     end
6199 end
6200 if first_n then
6201     dir_mark(head, first_n, last_n, outer)
6202 end
6203 if first_d then
6204     dir_mark(head, first_d, last_d, outer)
6205 end

```

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

```

6206 return node.prev(head) or head
6207 end
6208 </basic-r>

```

And here the Lua code for bidi=basic:

```

6209 <(*basic)
6210 Babel = Babel or {}
6211
6212 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6213
6214 Babel.fontmap = Babel.fontmap or {}
6215 Babel.fontmap[0] = {}          -- l
6216 Babel.fontmap[1] = {}          -- r
6217 Babel.fontmap[2] = {}          -- al/an
6218
6219 Babel.bidi_enabled = true
6220 Babel.mirroring_enabled = true
6221
6222 require('babel-data-bidi.lua')
6223
6224 local characters = Babel.characters
6225 local ranges = Babel.ranges
6226
6227 local DIR = node.id('dir')
6228 local GLYPH = node.id('glyph')
6229
6230 local function insert_implicit(head, state, outer)
6231     local new_state = state
6232     if state.sim and state.eim and state.sim ~= state.eim then
6233         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6234         local d = node.new(DIR)
6235         d.dir = '+' .. dir
6236         node.insert_before(head, state.sim, d)

```

```

6237     local d = node.new(DIR)
6238     d.dir = '-' .. dir
6239     node.insert_after(head, state.eim, d)
6240 end
6241 new_state.sim, new_state.eim = nil, nil
6242 return head, new_state
6243 end
6244
6245 local function insert_numeric(head, state)
6246     local new
6247     local new_state = state
6248     if state.san and state.ean and state.san ~= state.ean then
6249         local d = node.new(DIR)
6250         d.dir = '+TLT'
6251         _, new = node.insert_before(head, state.san, d)
6252         if state.san == state.sim then state.sim = new end
6253         local d = node.new(DIR)
6254         d.dir = '-TLT'
6255         _, new = node.insert_after(head, state.ean, d)
6256         if state.ean == state.eim then state.eim = new end
6257     end
6258     new_state.san, new_state.ean = nil, nil
6259     return head, new_state
6260 end
6261
6262 -- TODO - \hbox with an explicit dir can lead to wrong results
6263 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6264 -- was s made to improve the situation, but the problem is the 3-dir
6265 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6266 -- well.
6267
6268 function Babel.bidi(head, ispar, hdir)
6269     local d -- d is used mainly for computations in a loop
6270     local prev_d = ''
6271     local new_d = false
6272
6273     local nodes = {}
6274     local outer_first = nil
6275     local inmath = false
6276
6277     local glue_d = nil
6278     local glue_i = nil
6279
6280     local has_en = false
6281     local first_et = nil
6282
6283     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6284
6285     local save_outer
6286     local temp = node.get_attribute(head, ATDIR)
6287     if temp then
6288         temp = temp % 3
6289         save_outer = (temp == 0 and 'l') or
6290                     (temp == 1 and 'r') or
6291                     (temp == 2 and 'al')
6292     elseif ispar then -- Or error? Shouldn't happen
6293         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6294     else -- Or error? Shouldn't happen
6295         save_outer = ('TRT' == hdir) and 'r' or 'l'

```

```

6296 end
6297 -- when the callback is called, we are just _after_ the box,
6298 -- and the textdir is that of the surrounding text
6299 -- if not ispar and hdir ~= tex.textdir then
6300 --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6301 -- end
6302 local outer = save_outer
6303 local last = outer
6304 -- 'al' is only taken into account in the first, current loop
6305 if save_outer == 'al' then save_outer = 'r' end
6306
6307 local fontmap = Babel.fontmap
6308
6309 for item in node.traverse(head) do
6310
6311   -- In what follows, #node is the last (previous) node, because the
6312   -- current one is not added until we start processing the neutrals.
6313
6314   -- three cases: glyph, dir, otherwise
6315   if item.id == GLYPH
6316     or (item.id == 7 and item.subtype == 2) then
6317
6318     local d_font = nil
6319     local item_r
6320     if item.id == 7 and item.subtype == 2 then
6321       item_r = item.replace -- automatic discs have just 1 glyph
6322     else
6323       item_r = item
6324     end
6325     local chardata = characters[item_r.char]
6326     d = chardata and chardata.d or nil
6327     if not d or d == 'nsm' then
6328       for nn, et in ipairs(ranges) do
6329         if item_r.char < et[1] then
6330           break
6331         elseif item_r.char <= et[2] then
6332           if not d then d = et[3]
6333             elseif d == 'nsm' then d_font = et[3]
6334           end
6335           break
6336         end
6337       end
6338     end
6339     d = d or 'l'
6340
6341     -- A short 'pause' in bidi for mapfont
6342     d_font = d_font or d
6343     d_font = (d_font == 'l' and 0) or
6344       (d_font == 'nsm' and 0) or
6345       (d_font == 'r' and 1) or
6346       (d_font == 'al' and 2) or
6347       (d_font == 'an' and 2) or nil
6348     if d_font and fontmap and fontmap[d_font][item_r.font] then
6349       item_r.font = fontmap[d_font][item_r.font]
6350     end
6351
6352     if new_d then
6353       table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6354       if inmath then

```

```

6355         attr_d = 0
6356     else
6357         attr_d = node.get_attribute(item, ATDIR)
6358         attr_d = attr_d % 3
6359     end
6360     if attr_d == 1 then
6361         outer_first = 'r'
6362         last = 'r'
6363     elseif attr_d == 2 then
6364         outer_first = 'r'
6365         last = 'al'
6366     else
6367         outer_first = 'l'
6368         last = 'l'
6369     end
6370     outer = last
6371     has_en = false
6372     first_et = nil
6373     new_d = false
6374 end
6375
6376 if glue_d then
6377     if (d == 'l' and 'l' or 'r') ~= glue_d then
6378         table.insert(nodes, {glue_i, 'on', nil})
6379     end
6380     glue_d = nil
6381     glue_i = nil
6382 end
6383
6384 elseif item.id == DIR then
6385     d = nil
6386     new_d = true
6387
6388 elseif item.id == node.id'glue' and item.subtype == 13 then
6389     glue_d = d
6390     glue_i = item
6391     d = nil
6392
6393 elseif item.id == node.id'math' then
6394     inmath = (item.subtype == 0)
6395
6396 else
6397     d = nil
6398 end
6399
6400 -- AL <= EN/ET/ES      -- W2 + W3 + W6
6401 if last == 'al' and d == 'en' then
6402     d = 'an'           -- W3
6403 elseif last == 'al' and (d == 'et' or d == 'es') then
6404     d = 'on'           -- W6
6405 end
6406
6407 -- EN + CS/ES + EN      -- W4
6408 if d == 'en' and #nodes >= 2 then
6409     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6410         and nodes[#nodes-1][2] == 'en' then
6411         nodes[#nodes][2] = 'en'
6412     end
6413 end

```



```

6414
6415 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6416 if d == 'an' and #nodes >= 2 then
6417   if (nodes[#nodes][2] == 'cs')
6418     and nodes[#nodes-1][2] == 'an' then
6419     nodes[#nodes][2] = 'an'
6420   end
6421 end
6422
6423 -- ET/EN                  -- W5 + W7->l / W6->on
6424 if d == 'et' then
6425   first_et = first_et or (#nodes + 1)
6426 elseif d == 'en' then
6427   has_en = true
6428   first_et = first_et or (#nodes + 1)
6429 elseif first_et then      -- d may be nil here !
6430   if has_en then
6431     if last == 'l' then
6432       temp = 'l'        -- W7
6433     else
6434       temp = 'en'       -- W5
6435     end
6436   else
6437     temp = 'on'         -- W6
6438   end
6439   for e = first_et, #nodes do
6440     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6441   end
6442   first_et = nil
6443   has_en = false
6444 end
6445
6446 -- Force mathdir in math if ON (currently works as expected only
6447 -- with 'l')
6448 if inmath and d == 'on' then
6449   d = ('TRT' == tex.mathdir) and 'r' or 'l'
6450 end
6451
6452 if d then
6453   if d == 'al' then
6454     d = 'r'
6455     last = 'al'
6456   elseif d == 'l' or d == 'r' then
6457     last = d
6458   end
6459   prev_d = d
6460   table.insert(nodes, {item, d, outer_first})
6461 end
6462
6463 outer_first = nil
6464
6465 end
6466
6467 -- TODO -- repeated here in case EN/ET is the last node. Find a
6468 -- better way of doing things:
6469 if first_et then      -- dir may be nil here !
6470   if has_en then
6471     if last == 'l' then
6472       temp = 'l'      -- W7

```

```

6473     else
6474         temp = 'en'    -- W5
6475     end
6476     else
6477         temp = 'on'    -- W6
6478     end
6479     for e = first_et, #nodes do
6480         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6481     end
6482 end
6483
6484 -- dummy node, to close things
6485 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6486
6487 ----- NEUTRAL -----
6488
6489 outer = save_outer
6490 last = outer
6491
6492 local first_on = nil
6493
6494 for q = 1, #nodes do
6495     local item
6496
6497     local outer_first = nodes[q][3]
6498     outer = outer_first or outer
6499     last = outer_first or last
6500
6501     local d = nodes[q][2]
6502     if d == 'an' or d == 'en' then d = 'r' end
6503     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6504
6505     if d == 'on' then
6506         first_on = first_on or q
6507     elseif first_on then
6508         if last == d then
6509             temp = d
6510         else
6511             temp = outer
6512         end
6513         for r = first_on, q - 1 do
6514             nodes[r][2] = temp
6515             item = nodes[r][1]    -- MIRRORING
6516             if Babel.mirroring_enabled and item.id == GLYPH
6517                 and temp == 'r' and characters[item.char] then
6518                 local font_mode = font.fonts[item.font].properties.mode
6519                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6520                     item.char = characters[item.char].m or item.char
6521                 end
6522             end
6523         end
6524         first_on = nil
6525     end
6526
6527     if d == 'r' or d == 'l' then last = d end
6528 end
6529
6530 ----- IMPLICIT, REORDER -----
6531

```

```

6532 outer = save_outer
6533 last = outer
6534
6535 local state = {}
6536 state.has_r = false
6537
6538 for q = 1, #nodes do
6539
6540     local item = nodes[q][1]
6541
6542     outer = nodes[q][3] or outer
6543
6544     local d = nodes[q][2]
6545
6546     if d == 'nsm' then d = last end          -- W1
6547     if d == 'en' then d = 'an' end
6548     local isdir = (d == 'r' or d == 'l')
6549
6550     if outer == 'l' and d == 'an' then
6551         state.san = state.san or item
6552         state.ean = item
6553     elseif state.san then
6554         head, state = insert_numeric(head, state)
6555     end
6556
6557     if outer == 'l' then
6558         if d == 'an' or d == 'r' then      -- im -> implicit
6559             if d == 'r' then state.has_r = true end
6560             state.sim = state.sim or item
6561             state.eim = item
6562         elseif d == 'l' and state.sim and state.has_r then
6563             head, state = insert_implicit(head, state, outer)
6564         elseif d == 'l' then
6565             state.sim, state.eim, state.has_r = nil, nil, false
6566         end
6567     else
6568         if d == 'an' or d == 'l' then
6569             if nodes[q][3] then -- nil except after an explicit dir
6570                 state.sim = item -- so we move sim 'inside' the group
6571             else
6572                 state.sim = state.sim or item
6573             end
6574             state.eim = item
6575         elseif d == 'r' and state.sim then
6576             head, state = insert_implicit(head, state, outer)
6577         elseif d == 'r' then
6578             state.sim, state.eim = nil, nil
6579         end
6580     end
6581
6582     if isdir then
6583         last = d          -- Don't search back - best save now
6584     elseif d == 'on' and state.san then
6585         state.san = state.san or item
6586         state.ean = item
6587     end
6588
6589 end
6590

```

```

6591 return node.prev(head) or head
6592 end
6593 </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.

```

6594 <{*nil}
6595 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6596 \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.

```

6597 \ifx\l@nil\@undefined
6598   \newlanguage\l@nil
6599   \@namedef{bbl@hyphendata@the\l@nil}{\relax}% Remove warning
6600   \let\bbl@elt\relax
6601   \edef\bbl@languages{% Add it to the list of languages
6602     \bbl@languages\bbl@elt{nil}{the\l@nil}}
6603 \fi

```

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

```

6604 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
6605 \let\captionnil\@empty
6606 \let\datenil\@empty

```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```

6607 \ldf@finish{nil}
6608 </nil>

```

16 Support for Plain T_EX (plain.def)

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```
6609 <(*bplain | blplain)
6610 \catcode`\{=1 % left brace is begin-group character
6611 \catcode`\}=2 % right brace is end-group character
6612 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
6613 \openin 0 hyphen.cfg
6614 \ifeof0
6615 \else
6616   \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that’s done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
6617   \def\input #1 {%
6618     \let\input\input\input\input\input
6619     \a hyphen.cfg
6620     \let\input\input\input\input\input
6621   }
6622 \fi
6623 >(/bplain | blplain)
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
6624 <(*bplain)\a plain.tex
6625 <(*blplain)\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
6626 <(*bplain)\def\fmtname{babel-plain}
6627 <(*blplain)\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some L^AT_EX features

The following code duplicates or emulates parts of L^AT_EX 2_ε that are needed for `babel`.

```
6628 <<(*Emulate LaTeX)>> ≡
```

```

6629 % == Code for plain ==
6630 \def\@empty{}
6631 \def\loadlocalcfg#1{%
6632   \openin0#1.cfg
6633   \ifeof0
6634     \closein0
6635   \else
6636     \closein0
6637     {\immediate\write16{*****}%
6638      \immediate\write16{* Local config file #1.cfg used}%
6639      \immediate\write16{*}%
6640     }
6641     \input #1.cfg\relax
6642   \fi
6643 \endofldf}

```

16.3 General tools

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

```

6644 \long\def\@firstofone#1{#1}
6645 \long\def\@firstoftwo#1#2{#1}
6646 \long\def\@secondoftwo#1#2{#2}
6647 \def\@nnil{\@nil}
6648 \def\@gobbletwo#1#2{}
6649 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6650 \def\@star@or@long#1{%
6651   \@ifstar
6652   {\let\l@ngrel@x\relax#1}%
6653   {\let\l@ngrel@x\long#1}}
6654 \let\l@ngrel@x\relax
6655 \def\@car#1#2\@nil{#1}
6656 \def\@cdr#1#2\@nil{#2}
6657 \let\@typeset@protect\relax
6658 \let\protected@edef\edef
6659 \long\def\@gobble#1{}
6660 \edef\@backslashchar{\expandafter\@gobble\string\}
6661 \def\strip@prefix#1>{}
6662 \def\g@addto@macro#1#2{%
6663   \toks@\expandafter{#1#2}%
6664   \xdef#1{\the\toks@}}
6665 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6666 \def\@nameuse#1{\csname #1\endcsname}
6667 \def\@ifundefined#1{%
6668   \expandafter\ifx\csname#1\endcsname\relax
6669     \expandafter\@firstoftwo
6670   \else
6671     \expandafter\@secondoftwo
6672   \fi}
6673 \def\@expandtwoargs#1#2#3{%
6674   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6675 \def\zap@space#1 #2{%
6676   #1%
6677   \ifx#2\@empty\else\expandafter\zap@space\fi
6678   #2}
6679 \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}`.

```

6680 \ifx\@preamblecmds\undefined
6681   \def\@preamblecmds{}
6682 \fi
6683 \def\@onlypreamble#1{%
6684   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6685     \@preamblecmds\do#1}}
6686 \@onlypreamble\@onlypreamble

```

Mimick L^AT_EX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```

6687 \def\begindocument{%
6688   \@begindocumenthook
6689   \global\let\@begindocumenthook\@undefined
6690   \def\do##1{\global\let##1\@undefined}%
6691   \@preamblecmds
6692   \global\let\do\noexpand}

6693 \ifx\@begindocumenthook\@undefined
6694   \def\@begindocumenthook{}
6695 \fi
6696 \@onlypreamble\@begindocumenthook
6697 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick L^AT_EX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endoflfd.

```

6698 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
6699 \@onlypreamble\AtEndOfPackage
6700 \def\@endoflfd{}
6701 \@onlypreamble\@endoflfd
6702 \let\bbl@afterlang\@empty
6703 \chardef\bbl@opt@hyphenmap\z@

```

L^AT_EX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

```

6704 \catcode`\&=\z@
6705 \ifx&\if@files\@undefined
6706   \expandafter\let\csname if@files\expandafter\endcsname
6707     \csname iffalse\endcsname
6708 \fi
6709 \catcode`\&=4

```

Mimick L^AT_EX's commands to define control sequences.

```

6710 \def\newcommand{\@star@or@long\new@command}
6711 \def\new@command#1{%
6712   \@testopt{\@newcommand#1}0}
6713 \def\@newcommand#1[#2]{%
6714   \@ifnextchar [{\@xargdef#1[#2]}%
6715     {\@argdef#1[#2]}}
6716 \long\def\@argdef#1[#2]#3{%
6717   \@yargdef#1\@ne{#2}{#3}}
6718 \long\def\@xargdef#1[#2][#3]#4{%
6719   \expandafter\def\expandafter#1\expandafter{%
6720     \expandafter\@protected@testopt\expandafter #1%
6721     \csname\string#1\expandafter\endcsname{#3}}}%
6722 \expandafter\@yargdef \csname\string#1\endcsname
6723   \tw@{#2}{#4}}
6724 \long\def\@yargdef#1#2#3{%
6725   \@tempcnta#3\relax
6726   \advance \@tempcnta \@ne
6727   \let\@hash@\relax

```

```

6728 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6729 \@tempcntb #2%
6730 \@whilenum\@tempcntb <\@tempcnta
6731 \do{%
6732   \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6733   \advance\@tempcntb \@ne}%
6734 \let\@hash@###
6735 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6736 \def\providecommand{\@star@or@long\provide@command}
6737 \def\provide@command#1{%
6738   \begingroup
6739   \escapechar\m@ne\xdef\@gtempa{\string#1}%
6740   \endgroup
6741   \expandafter\ifundefined\@gtempa
6742     {\def\reserved@a{\new@command#1}}%
6743     {\let\reserved@a\relax
6744      \def\reserved@a{\new@command\reserved@a}}%
6745   \reserved@a}%
6746 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6747 \def\declare@robustcommand#1{%
6748   \edef\reserved@a{\string#1}%
6749   \def\reserved@b{#1}%
6750   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6751   \edef#1{%
6752     \ifx\reserved@a\reserved@b
6753       \noexpand\x@protect
6754       \noexpand#1%
6755     \fi
6756     \noexpand\protect
6757     \expandafter\noexpand\csname
6758       \expandafter\@gobble\string#1 \endcsname
6759   }%
6760   \expandafter\new@command\csname
6761     \expandafter\@gobble\string#1 \endcsname
6762 }
6763 \def\x@protect#1{%
6764   \ifx\protect\@typeset@protect\else
6765     \x@protect#1%
6766   \fi
6767 }
6768 \catcode`\&=\z@ % Trick to hide conditionals
6769 \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`.

```

6770 \def\bbl@tempa{\csname newif\endcsname&fin@}
6771 \catcode`\&=4
6772 \ifx\in@\@undefined
6773   \def\in@#1#2{%
6774     \def\in@##1#1##2##3\in@{%
6775       \ifx\in@##2\in@false\else\in@true\fi}%
6776     \in@#2#1\in@\in@@}
6777 \else
6778   \let\bbl@tempa\@empty
6779 \fi
6780 \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).

```
6781 \def\ifpackagewith#1#2#3#4{#3}
```

The \TeX macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```
6782 \def\ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\TeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```
6783 \ifx\@tempcnta\@undefined
6784   \csname newcount\endcsname\@tempcnta\relax
6785 \fi
6786 \ifx\@tempcntb\@undefined
6787   \csname newcount\endcsname\@tempcntb\relax
6788 \fi
```

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

```
6789 \ifx\bye\@undefined
6790   \advance\count10 by -2\relax
6791 \fi
6792 \ifx\@ifnextchar\@undefined
6793   \def\@ifnextchar#1#2#3{%
6794     \let\reserved@d=#1%
6795     \def\reserved@a{#2}\def\reserved@b{#3}%
6796     \futurelet\@let@token\@ifnch}
6797   \def\@ifnch{%
6798     \ifx\@let@token\@sptoken
6799       \let\reserved@c\@xifnch
6800     \else
6801       \ifx\@let@token\reserved@d
6802         \let\reserved@c\reserved@a
6803       \else
6804         \let\reserved@c\reserved@b
6805       \fi
6806     \fi
6807     \reserved@c}
6808   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6809   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6810 \fi
6811 \def\@testopt#1#2{%
6812   \@ifnextchar[#{#1}{#1[#2]}}
6813 \def\@protected@testopt#1{%
6814   \ifx\protect\@typeset@protect
6815     \expandafter\@testopt
6816   \else
6817     \@x@protect#1%
6818   \fi}
6819 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6820   #2\relax}\fi}
6821 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6822   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```

6823 \def\DeclareTextCommand{%
6824   \@dec@text@cmd\providecommand
6825 }
6826 \def\ProvideTextCommand{%
6827   \@dec@text@cmd\providecommand
6828 }
6829 \def\DeclareTextSymbol#1#2#3{%
6830   \@dec@text@cmd\chardef#1{#2}#3\relax
6831 }
6832 \def\@dec@text@cmd#1#2#3{%
6833   \expandafter\def\expandafter#2%
6834     \expandafter{%
6835       \csname#3-cmd\expandafter\endcsname
6836       \expandafter#2%
6837       \csname#3\string#2\endcsname
6838     }%
6839 %   \let\@ifdefinable\rc@ifdefinable
6840   \expandafter#1\csname#3\string#2\endcsname
6841 }
6842 \def\@current@cmd#1{%
6843   \ifx\protect\@typeset@protect\else
6844     \noexpand#1\expandafter\@gobble
6845   \fi
6846 }
6847 \def\@changed@cmd#1#2{%
6848   \ifx\protect\@typeset@protect
6849     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6850       \expandafter\ifx\csname ?\string#1\endcsname\relax
6851         \expandafter\def\csname ?\string#1\endcsname{%
6852           \@changed@x@err{#1}%
6853         }%
6854       \fi
6855       \global\expandafter\let
6856         \csname\cf@encoding \string#1\expandafter\endcsname
6857         \csname ?\string#1\endcsname
6858     \fi
6859     \csname\cf@encoding\string#1%
6860     \expandafter\endcsname
6861   \else
6862     \noexpand#1%
6863   \fi
6864 }
6865 \def\@changed@x@err#1{%
6866   \errhelp{Your command will be ignored, type <return> to proceed}%
6867   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6868 \def\DeclareTextCommandDefault#1{%
6869   \DeclareTextCommand#1?%
6870 }
6871 \def\ProvideTextCommandDefault#1{%
6872   \ProvideTextCommand#1?%
6873 }
6874 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6875 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6876 \def\DeclareTextAccent#1#2#3{%
6877   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6878 }
6879 \def\DeclareTextCompositeCommand#1#2#3#4{%
6880   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6881   \edef\reserved@b{\string#1}%

```

```

6882 \edef\reserved@c{%
6883   \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6884 \ifx\reserved@b\reserved@c
6885   \expandafter\expandafter\expandafter\ifx
6886     \expandafter\@car\reserved@a\relax\relax\@nil
6887     \@text@composite
6888   \else
6889     \edef\reserved@b##1{%
6890       \def\expandafter\noexpand
6891         \csname#2\string#1\endcsname####1{%
6892         \noexpand\@text@composite
6893         \expandafter\noexpand\csname#2\string#1\endcsname
6894         #####1\noexpand\@empty\noexpand\@text@composite
6895         {##1}%
6896       }%
6897     }%
6898     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6899   \fi
6900   \expandafter\def\csname\expandafter\string\csname
6901     #2\endcsname\string#1-\string#3\endcsname{#4}
6902   \else
6903     \errhelp{Your command will be ignored, type <return> to proceed}%
6904     \errmessage{\string\DeclareTextCompositeCommand\space used on
6905       inappropriate command \protect#1}
6906   \fi
6907 }
6908 \def\@text@composite#1#2#3\@text@composite{%
6909   \expandafter\@text@composite@x
6910     \csname\string#1-\string#2\endcsname
6911 }
6912 \def\@text@composite@x#1#2{%
6913   \ifx#1\relax
6914     #2%
6915   \else
6916     #1%
6917   \fi
6918 }
6919 %
6920 \def\@strip@args#1:#2-#3\@strip@args{#2}
6921 \def\DeclareTextComposite#1#2#3#4{%
6922   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6923   \bgroup
6924     \lccode`\@=#4%
6925     \lowercase{%
6926   \egroup
6927     \reserved@a @%
6928   }%
6929 }
6930 %
6931 \def\UseTextSymbol#1#2{#2}
6932 \def\UseTextAccent#1#2#3{#3}
6933 \def\@use@text@encoding#1{#1}
6934 \def\DeclareTextSymbolDefault#1#2{%
6935   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6936 }
6937 \def\DeclareTextAccentDefault#1#2{%
6938   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6939 }
6940 \def\cf@encoding{OT1}

```

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

```
6941 \DeclareTextAccent{"}{OT1}{127}
6942 \DeclareTextAccent{'}{OT1}{19}
6943 \DeclareTextAccent{^}{OT1}{94}
6944 \DeclareTextAccent`}{OT1}{18}
6945 \DeclareTextAccent{~}{OT1}{126}
```

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

```
6946 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6947 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
6948 \DeclareTextSymbol{\textquoteleft}{OT1}{`'}
6949 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
6950 \DeclareTextSymbol{\i}{OT1}{16}
6951 \DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```
6952 \ifx\scriptsize@undefined
6953   \let\scriptsize\sevenrm
6954 \fi
6955 % End of code for plain
6956 <</Emulate LaTeX>>
```

A proxy file:

```
6957 <*plain>
6958 \input babel.def
6959 </plain>
```

17 Acknowledgements

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

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

References

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