

Babel

Version 3.41
2020/02/28

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

Localization and
internationalization

TeX

pdfTeX

LuaTeX

LuaHBTeX

XeTeX

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	7
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	8
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	9
1.9	More on selection	10
1.10	Shorthands	12
1.11	Package options	15
1.12	The base option	17
1.13	ini files	18
1.14	Selecting fonts	25
1.15	Modifying a language	27
1.16	Creating a language	28
1.17	Digits and counters	31
1.18	Accessing language info	32
1.19	Hyphenation and line breaking	33
1.20	Selecting scripts	35
1.21	Selecting directions	36
1.22	Language attributes	40
1.23	Hooks	40
1.24	Languages supported by babel with ldf files	41
1.25	Unicode character properties in luatex	43
1.26	Tweaking some features	43
1.27	Tips, workarounds, known issues and notes	43
1.28	Current and future work	45
1.29	Tentative and experimental code	45
2	Loading languages with language.dat	45
2.1	Format	46
3	The interface between the core of babel and the language definition files	46
3.1	Guidelines for contributed languages	48
3.2	Basic macros	48
3.3	Skeleton	49
3.4	Support for active characters	50
3.5	Support for saving macro definitions	51
3.6	Support for extending macros	51
3.7	Macros common to a number of languages	51
3.8	Encoding-dependent strings	52
4	Changes	55
4.1	Changes in babel version 3.9	55
II	Source code	56
5	Identification and loading of required files	56

6	locale directory	56
7	Tools	57
7.1	Multiple languages	61
8	The Package File (\LaTeX, babel.sty)	62
8.1	base	62
8.2	key=value options and other general option	64
8.3	Conditional loading of shorthands	66
8.4	Language options	67
9	The kernel of Babel (babel.def, common)	70
9.1	Tools	70
9.2	Hooks	72
9.3	Setting up language files	74
9.4	Shorthands	76
9.5	Language attributes	86
9.6	Support for saving macro definitions	88
9.7	Short tags	89
9.8	Hyphens	89
9.9	Multiencoding strings	91
9.10	Macros common to a number of languages	97
9.11	Making glyphs available	97
9.11.1	Quotation marks	97
9.11.2	Letters	98
9.11.3	Shorthands for quotation marks	99
9.11.4	Umlauts and tremas	100
9.12	Layout	101
9.13	Load engine specific macros	102
9.14	Creating languages	102
10	Adjusting the Babel behavior	116
11	The kernel of Babel (babel.def for \LaTeXonly)	117
11.1	The redefinition of the style commands	117
11.2	Cross referencing macros	117
11.3	Marks	120
11.4	Preventing clashes with other packages	122
11.4.1	ifthen	122
11.4.2	varioref	122
11.4.3	hhline	123
11.4.4	hyperref	123
11.4.5	fancyhdr	124
11.5	Encoding and fonts	124
11.6	Basic bidi support	126
11.7	Local Language Configuration	129
12	Multiple languages (switch.def)	130
12.1	Selecting the language	131
12.2	Errors	140
13	Loading hyphenation patterns	141
14	Font handling with fontspec	146

15	Hooks for XeTeX and LuaTeX	151
15.1	XeTeX	151
15.2	Layout	153
15.3	LuaTeX	155
15.4	Southeast Asian scripts	160
15.5	CJK line breaking	164
15.6	Automatic fonts and ids switching	164
15.7	Layout	172
15.8	Auto bidi with basic and basic-r	174
16	Data for CJK	185
17	The ‘nil’ language	185
18	Support for Plain T_EX (plain.def)	186
18.1	Not renaming hyphen.tex	186
18.2	Emulating some L _A T _E X features	187
18.3	General tools	187
18.4	Encoding related macros	191
19	Acknowledgements	194

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	8
Unknown language ‘LANG’	8
Argument of \language@active@arg” has an extra }	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	27
Package babel Info: The following fonts are not babel standard families	27

Part I

User guide

- This user guide focuses on internationalization and localization with \LaTeX . There are also some notes on its use with Plain \TeX .
- 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 could be still unstable. Please, report any issues you find in GitHub, which is better than just complaining on an e-mail list or a web forum.
- If you are interested in the \TeX multilingual support, please join the kadingira mail list. You can follow the development of babel in GitHub (which provides many sample files, too).
- See section 3.1 for contributing a language.
- The first sections describe the traditional way of loading a language (with ldf files). The alternative way based on ini files, which complements the previous one (it does *not* replace it), is described below.

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

EXAMPLE Here is a simple full example for “traditional” \TeX engines (see below for xetex and luatex). The packages fontenc and inputenc do not belong to babel, but they are included in the example because typically you will need them (however, the package inputenc may be omitted with $\LaTeX \geq 2018-04-01$ if the encoding is UTF-8):

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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{article}

\usepackage[russian]{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 could 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.

Another approach is making the language (french in the example) a global option in order to let other packages detect and use it:

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

In this last example, the package `varioref` will also see the option and will be able to use it.

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.

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 could be overridden with something like that before `\documentclass`:

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

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

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

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

To switch the language there are two basic macros, described below in detail: `\selectlanguage` is used for blocks of text, while `\foreignlanguage` is for chunks of text inside paragraphs.

EXAMPLE A full bilingual document follows. The main language is french, which is activated when the document begins. The package inputenc may be omitted with \LaTeX \geq 2018-04-01 if the encoding is UTF-8.

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

And an English paragraph, with a short text in
\foreignlanguage{french}{français}.

\end{document}
```

EXAMPLE With xetex and luatex, the following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and \today in Danish and Vietnamese. No additional packages are required.

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename{} -- \alsoname{} -- \today

\selectlanguage{vietnamese}

\prefacename{} -- \alsoname{} -- \today

\end{document}
```

1.3 Mostly monolingual documents

New 3.39 Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of \babelfont, if used.)

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

EXAMPLE A trivial document is:

LUATEX/XETEX

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



```

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

```

! 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):

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

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

NOTE For “historical reasons”, a macro name is converted to a language name without the leading `\`; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a “real” name is deprecated.

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

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 `bidirules` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

1.8 Auxiliary language selectors

`\begin{otherlanguage}` $\langle\textit{language}\rangle$... `\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*}` $\langle\textit{language}\rangle$... `\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}` $\langle\textit{language}\rangle$... `\end{hyphenrules}`

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

1.9 More on selection

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

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

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

EXAMPLE With

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

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with `fontenc`.⁵ A couple of examples:

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

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

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary TeX code. Shorthands can be used for different kinds of things, for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionaries and breaks can be inserted easily with "-", "=", etc. 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 \knbccode, and luatex can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

NOTE Note the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE Let’s assume you want a unified set of shorthand for dictionaries (languages do not define shorthands consistently, and “-”, “\”, “=” have different meanings). You could 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 is repeated at the beginning of the next line. You could then set:

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

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

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

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

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

⁶Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.

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

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

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

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

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

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

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

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

For your records, here is a list of shorthands, but you must double check them, as they may change:⁷

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

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

Basque " ' ~

Breton : ; ? !

Catalan " ' `

Czech " -

Esperanto ^

Estonian " ~

French (all varieties) : ; ? !

Galician " . ' ~ < >

Greek ~

Hungarian `

Kurmanji ^

Latin " ^ =

Slovak " ^ ' -

Spanish " . < > ' ~

Turkish : ! =

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

⁷Thanks to Enrico Gregorio

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

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

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

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

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

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

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

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

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

1.11 Package options

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

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

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

activegrave Same for ```.

shorthands= $\langle char \rangle \langle char \rangle \dots \mid \text{off}$

The only language shorthands activated are those given, like, eg:

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

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

safe=	none ref bib
	Some \LaTeX macros are redefined so that using shorthands is safe. With <code>safe=bib</code> only <code>\nocite</code> , <code>\bibcite</code> and <code>\bibitem</code> are redefined. With <code>safe=ref</code> only <code>\newlabel</code> , <code>\ref</code> and <code>\pageref</code> are redefined (as well as a few macros from <code>varioref</code> and <code>ifthen</code>). With <code>safe=none</code> no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of New 3.34 , in $\epsilon\TeX$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).
math=	active normal
	Shorthands are mainly intended for text, not for math. By setting this option with the value <code>normal</code> they are deactivated in math mode (default is <code>active</code>) and things like <code>#{a'}</code> (a closing brace after a shorthand) are not a source of trouble anymore.
config=	$\langle file \rangle$
	Load $\langle file \rangle$.cfg instead of the default config file <code>bblopts.cfg</code> (the file is loaded even with <code>noconfigs</code>).
main=	$\langle language \rangle$
	Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.
headfoot=	$\langle language \rangle$
	By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
noconfigs	Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected .cfg file. However, if the key config is set, this file is loaded.
showlanguages	Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
nocase	New 3.9l Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code>) are ignored. Use only if there are incompatibilities with other packages.
silent	New 3.9l No warnings and no <i>infos</i> are written to the log file. ⁹
strings=	generic unicode encoded $\langle label \rangle$ $\langle font encoding \rangle$
	Selects the encoding of strings in languages supporting this feature. Predefined labels are <code>generic</code> (for traditional \TeX , LICR and ASCII strings), <code>unicode</code> (for engines like <code>xetex</code> and <code>luatex</code>) and <code>encoded</code> (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in <code>\MakeUpper</code> case and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
hyphenmap=	off main select other other*

⁹You can use alternatively the package `silence`.

New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.¹⁰ It can take the following values:

off deactivates this feature and no case mapping is applied;
first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at `\begin{document}`), but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated;¹¹
select sets it only at `\selectlanguage`;
other also sets it at `otherlanguage`;
other* also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.¹²

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

New 3.14 Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.21.

layout=

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.21.

1.12 The base option

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

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

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

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

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

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

¹⁰Turned off in plain.

¹¹Duplicated options count as several ones.

¹²Providing `foreign` is pointless, because the case mapping applied is that at the end of the paragraph, but if either `xetex` or `luatex` change this behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

```

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

```

1.13 ini files

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

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

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them currently (by means of \babelprovide), but a higher interface, based on package options, is under study. In other words, \babelprovide is mainly meant for auxiliary tasks.

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}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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 follows:

Arabic Monolingual documents mostly work in luatex, but it must be fine tuned, and a recent version of fontspec/loaotfloat is required. In xetex babel resorts to the bidi package, which seems to work.

Hebrew Niqud marks seem to work in both engines, but cantillation marks are misplaced (xetex seems better, but still problematic).

Devanagari In luatex many fonts work, but some others do not, the main issue being the ‘ra’. It is advisable to set explicitly the script to either deva or dev2, eg:

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

Other Indic scripts are still under development in luatex. On the other hand, xetex is better. The upcoming luatex will be based on luahtex, so Indic scripts will be rendered correctly with the option `Renderer=Harfbuzz` in `FONTSPEC`.

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

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

East Asia scripts Settings for either Simplified or Traditional should work out of the box. luatex does basic line breaking, but currently xetex does not (you may load `zhspacing`). 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{ltjbook}
\usepackage[japanese]{babel}
```

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

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

af	Afrikaans ^{ul}	bem	Bemba
agq	Aghem	bez	Bena
ak	Akan	bg	Bulgarian ^{ul}
am	Amharic ^{ul}	bm	Bambara
ar	Arabic ^{ul}	bn	Bangla ^{ul}
ar-DZ	Arabic ^{ul}	bo	Tibetan ^u
ar-MA	Arabic ^{ul}	brx	Bodo
ar-SY	Arabic ^{ul}	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian ^{ul}
asa	Asu	bs	Bosnian ^{ul}
ast	Asturian ^{ul}	ca	Catalan ^{ul}
az-Cyrl	Azerbaijani	ce	Chechen
az-Latn	Azerbaijani	cgg	Chiga
az	Azerbaijani ^{ul}	chr	Cherokee
bas	Basaa	ckb	Central Kurdish
be	Belarusian ^{ul}	cs	Czech ^{ul}

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

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

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

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

filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda

konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole

nynorsk	serbian-latin-bosniaherzegovina
occitan	serbian-latin-kosovo
oriya	serbian-latin-montenegro
oromo	serbian-latin
ossetic	serbian-latn-ba
pashto	serbian-latn-me
persian	serbian-latn-xk
piedmontese	serbian-latn
polish	serbian
portuguese-br	shambala
portuguese-brazil	shona
portuguese-portugal	sichuanyi
portuguese-pt	sinhala
portuguese	slovak
punjabi-arab	slovene
punjabi-arabic	slovenian
punjabi-gurmukhi	soga
punjabi-guru	somali
punjabi	spanish-mexico
quechua	spanish-mx
romanian	spanish
romansh	standardmoroccantamazight
rombo	swahili
rundi	swedish
russian	swissgerman
rwa	tachelhit-latin
sakha	tachelhit-latn
samburu	tachelhit-tfng
samin	tachelhit-tifinagh
sango	tachelhit
sangu	taita
sanskrit-beng	tamil
sanskrit-bengali	tasawaq
sanskrit-deva	telugu
sanskrit-devanagari	teso
sanskrit-gujarati	thai
sanskrit-gujr	tibetan
sanskrit-kannada	tigrinya
sanskrit-knda	tongan
sanskrit-malayalam	turkish
sanskrit-mlym	turkmen
sanskrit-telu	ukenglish
sanskrit-telugu	ukrainian
sanskrit	upporsorbian
scottishgaelic	urdu
sena	usenglish
serbian-cyrillic-bosniaherzegovina	usorbian
serbian-cyrillic-kosovo	uyghur
serbian-cyrillic-montenegro	uzbek-arab
serbian-cyrillic	uzbek-arabic
serbian-cyrl-ba	uzbek-cyrillic
serbian-cyrl-me	uzbek-cyrl
serbian-cyrl-xk	uzbek-latin
serbian-cyrl	uzbek-latn

uzbek	walser
vai-latin	welsh
vai-latn	westernfrisian
vai-vai	yangben
vai-vaii	yiddish
vai	yoruba
vietnam	zarma
vietnamese	zulu afrikaans
vunjo	

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=abcdefghijklj`. Keys may be added, too. Without `import` you may modify the identification keys. This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

1.14 Selecting fonts

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

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

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}
```

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

```

\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 could replace the red line above with, say:

LUATEX/XETEX

```

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

```

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

```

\babelfont{kai}{FandolKai}

```

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

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

LUATEX/XETEX

```

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

```

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

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

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

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

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

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

This is *not* and error. This warning is shown by fontspec, not by babel. It could 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.

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

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

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do so.

- 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}
```

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

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

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

NOTE 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.

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

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

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

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

1.16 Creating a language

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

`\babelprovide` [$\langle options \rangle$] { $\langle language-name \rangle$ }

If the language $\langle language-name \rangle$ has not been loaded as class or package option and there are no $\langle options \rangle$, 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`, $\langle language-name \rangle$ is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the `ini` file corresponding to that name; the same applies to OpenType language and script.

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

```
Package babel Warning: \mylangchaptername not set. Please, define  
(babel)                it in the preamble with something like:  
(babel)                \renewcommand\maylangchaptername{..}  
(babel)                Reported on input line 18.
```

In most cases, you will only need to define a few macros.

EXAMPLE If you need a language named `arhinish`:

```
\usepackage[danish]{babel}  
\babelprovide{arhinish}  
\renewcommand\arhinishchaptername{Chapitula}  
\renewcommand\arhinishrefname{Refirenke}  
\renewcommand\arhinishhyphenmins{22}
```

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

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

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

The main language is not changed (danish in this example). So, you must add `\selectlanguage{arhinish}` or other selectors where necessary. If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

import= *<language-tag>*

New 3.13 Imports data from an ini file, including captions, date, and hyphenmins. 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 could be written:

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

There are about 200 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 will show a warning about the current lack of suitability of the date format (french, breton, and occitan).

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}`.

captions= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

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

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

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

main This valueless option makes the language the main one. Only in newly defined languages.

script= \langle *script-name* \rangle

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

language= \langle *language-name* \rangle

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

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 with a character belonging to the script of this locale is found. 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 the this locale (as set with `\babelfont`). This option is not compatible with `mapfont`. Characters can be added with `\babelcharproperty`.

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.

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

Sets the interword space for the writing system of the language, in em units (so, `0 .1 0` is `0em` plus `.1em`). Like `\spaceskip`, the em unit applied is that of the current text (more

precisely, the previous glyph). Currently used only in Southeast Asian scripts, like Thai, and CJK.

`intrapenalty=` $\langle penalty \rangle$

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

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).

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 `\edef`). Currently, they are limited to numbers below 10000.

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

- `\localenumeral{<style>}{<number>}`, like `\localenumeral{abjad}{15}`

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

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

The styles are:

Ancient Greek lower.ancient, upper.ancient
Arabic abjad, maghrebi.abjad
Belarusan, Bulgarian, Macedonian, Serbian lower, upper
Hebrew letters (neither geresh nor gershayim yet)
Hindi alphabetic
Armenian lower, upper
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
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

1.18 Accessing language info

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

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

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

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

`\localeinfo` `{<field>}`

New 3.38 If an ini file has been loaded for the current language, you may access the information stored in it. This macros 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 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 language 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).

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`.

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

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

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

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

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

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

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

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

```

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

```

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

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

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

EXAMPLE Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account). For example, you can use the `string` replacement to replace a character (or series of them) by another character (or series of them). Thus, to enter \check{z} as `zh` and \check{s} as `sh` in a newly created locale for transliterated Russian:

```

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

```

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

1.20 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.¹⁵

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.¹⁶

`\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.

¹⁵The 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.

¹⁶But still defined for backwards compatibility.

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.

1.21 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 could 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 could be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including the picture environment (with `pict2e`) and `pfg/tikz`. Also, indexes and the like are under study, as well as math (there is progress in the latter, too, but for example cases may fail).

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg. `\subsection`..`\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 could 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.¹⁷

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

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

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

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

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

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

tabular required in `luatex` for R tabular (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** .

¹⁷Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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. It attempts to do the same for `pgf/tikz`. Somewhat experimental. **New 3.32** .

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

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 `{\section-name}`

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

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

New 3.17 Something like:

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

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


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

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

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

(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}{\footnote}
```

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.

1.22 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`).

1.23 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 loads `switch.def`. It can be used to load a different version of this file or to load nothing.

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).

1.24 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 bahasa, indonesian, indon, bahasai
Interlingua interlingua
Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay bahasam, malay, melayu
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuges, portuguese, brazilian, brazil
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

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

```

\documentclass{article}
\usepackage[hindi]{babel}
\begin{document}

```

```
{\dn devaanaa.m priya.h}
\end{document}
```

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

1.25 Unicode character properties in luatex

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

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

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

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

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

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

1.26 Tweaking some features

`\babeladjust` $\{ \langle \textit{key-value-list} \rangle \}$

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

1.27 Tips, workarounds, known issues and notes

- If you use the document class `book` and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), \LaTeX will keep complaining about an undefined label. To prevent such problems, you could 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}}
```

(A recent version of `inputenc` is required.)

- 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.¹⁸ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- `Babel` does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in `xetex`.

¹⁸This explains why \LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

1.28 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.¹⁹ But that is the easy part, because they don't require modifying the L^AT_EX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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).

1.29 Tentative and experimental code

See the code section for \foreignlanguage* (a new starred version of \foreignlanguage).

Old and deprecated stuff

A couple of tentative macros were provided by babel ($\geq 3.9g$) with a partial solution for “Unicode” fonts. These macros are now deprecated — use \babelfont. A short description follows, for reference:

- \babelFSstore{\(babel-language\)} sets the current three basic families (rm, sf, tt) as the default for the language given.
- \babelFSdefault{\(babel-language\)}{\(fontspec-features\)} patches \fontspec so that the given features are always passed as the optional argument or added to it (not an ideal solution).

So, for example:

```
\setmainfont[Language=Turkish]{Minion Pro}
\babelFSstore{turkish}
\setmainfont{Minion Pro}
\babelFSfeatures{turkish}{Language=Turkish}
```

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).²⁰ Until

¹⁹See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to T_EX because their aim is just to display information and not fine typesetting.

²⁰This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

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).²¹

2.1 Format

In that file the person who maintains a \TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored²². 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 \TeX 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.²³ For example:

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

With the previous settings, if the encoding when the language is selected is T1 then the patterns in `hyphenT1.ger` are used, but otherwise use those in `hyphen.ger` (note the encoding could 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

²¹The 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`.

²²This is because different operating systems sometimes use *very* different file-naming conventions.

²³This is not a new feature, but in former versions it didn't work correctly.

the macros that produce texts. Also the language-switching possibility which has been built into the babel system has its implications.

The following assumptions are made:

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

Some recommendations:

- The preferred shorthand is `"`, which is not used in \LaTeX (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras\langle lang \rangle` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁴
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

²⁴But not removed, for backward compatibility.

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

Now language files are “outsourced” and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN). Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

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

The following page provides a starting point: <http://www.texnia.com/incubator.html>. If you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

3.2 Basic macros

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

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

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

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

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

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

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters

	were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do <i>not</i> set them).
<code>\captions<lang></code>	The macro <code>\captions<lang></code> defines the macros that hold the texts to replace the original hard-wired texts.
<code>\date<lang></code>	The macro <code>\date<lang></code> defines <code>\today</code> .
<code>\extras<lang></code>	The macro <code>\extras<lang></code> 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.
<code>\noextras<lang></code>	Because we want to let the user switch between languages, but we do not know what state \TeX might be in after the execution of <code>\extras<lang></code> , a macro that brings \TeX into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras<lang></code> .
<code>\bbl@declareattribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \TeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \TeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions<lang></code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

```

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

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

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

```

```

\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`. \LaTeX 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 this²⁵.

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

`\babel@savevariable` A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the `<variable>`.
 The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6 Support for extending macros

`\addto` The macro `\addto{<control sequence>}{<TeX code>}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`. Be careful when using this macro, because depending on the case the assignment could 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

²⁵This mechanism was introduced by Bernd Raichle.

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

`\bbl@nonfrenchspacing`

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

The $\langle category \rangle$ is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.²⁶ It may be empty, too, but in such a case using $\backslash SetString$ is an error (but not $\backslash SetCase$).

```
\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}

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

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

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

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

\EndBabelCommands
```

When used in ldf files, previous values of $\backslash \langle category \rangle \langle language \rangle$ 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).

²⁶In future releases further categories may be added.

\StartBabelCommands *{\language-list}{\category}{[\selector]}

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.²⁷

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands {\code}

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

\SetString {\macro-name}{\string}

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

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

\SetStringLoop {\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 could 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=`I\relax
 \uccode`ı=`I\relax}
{\lccode`İ=`i\relax
 \lccode`I=`ı\relax}

\StartBabelCommands{turkish}{}[
```

²⁷This replaces in 3.9g a short-lived \UseStrings which has been removed because it did not work.


```

\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 through the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{<uccode-from>}{<uccode-to>}{<step>}{<lccode>}` loops through the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for *many-to-one*).

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

```

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

```

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

4 Changes

4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`name. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.

- The :ENC mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- ' (with activeacute) had the original value when writing to an auxiliary file, and things like an infinite loop could 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 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. By default it also loads switch.def.

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

6 locale directory

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

Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and

polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). Multi-letter qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (all lowercase).

7 Tools

```
1 <<version=3.41>>
2 <<date=2020/02/28>>
```

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@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
14 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
15 \def\bbl@loop#1#2#3,{%
16   \ifx\@nnil#3\relax\else
17     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
18   \fi}
19 \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.

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

`\bbl@afterelse` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement²⁸. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

`\bbl@afterfi`

```
26 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
27 \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.

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

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

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

```
47 \begingroup
48 \gdef\bbl@ifunset#1{%
49   \expandafter\ifx\csname#1\endcsname\relax
50     \expandafter\@firstoftwo
51   \else
52     \expandafter\@secondoftwo
```

²⁸This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

```

53   \fi}
54   \bbl@ifunset{ifcstype}%
55   {}%
56   {\gdef\bbl@ifunset#1{%
57     \ifcstype#1\endcstype
58     \expandafter\ifx\cstype#1\endcstype\relax
59     \bbl@afterelse\expandafter\@firstoftwo
60     \else
61     \bbl@afterfi\expandafter\@secondoftwo
62     \fi
63     \else
64     \expandafter\@firstoftwo
65     \fi}}
66 \endgroup

```

`\bbl@ifblank` A tool from url, by Donald Arseneau, which tests if a string is empty or space.

```

67 \def\bbl@ifblank#1{%
68   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
69 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}

```

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).

```

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

```

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

```

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

```

`\bbl@replace`

```

90 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
91   \toks@{}%
92   \def\bbl@replace@aux##1#2##2#2{%
93     \ifx\bbl@nil##2%
94       \toks@\expandafter{\the\toks@##1}%
95     \else
96       \toks@\expandafter{\the\toks@##1#3}%
97       \bbl@afterfi
98       \bbl@replace@aux##2#2%
99     \fi}%

```

```

100 \expandafter\bb1@replace@aux#1#2\bb1@nil#2%
101 \edef#1{\the\toks@}

```

An extensions 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 \bb1@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bb1@replace; I'm not sure ckecking the replacement is really necessary or just paranoia).

```

102 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
103 \bb1@exp{\def\\bb1@parsedef##1\detokenize{macro:}}#2->#3\relax{%
104   \def\bb1@tempa{#1}%
105   \def\bb1@tempb{#2}%
106   \def\bb1@tempe{#3}}
107 \def\bb1@sreplace#1#2#3{%
108   \begingroup
109     \expandafter\bb1@parsedef\meaning#1\relax
110     \def\bb1@tempc{#2}%
111     \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
112     \def\bb1@tempd{#3}%
113     \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
114     \bb1@xin@\bb1@tempc{\bb1@tempe}% If not in macro, do nothing
115     \ifin@
116       \bb1@exp{\\bb1@replace\\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
117       \def\bb1@tempc{%      Expanded an executed below as 'uplevel'
118         \\makeatletter % "internal" macros with @ are assumed
119         \\scantokens{%
120           \bb1@tempa\\@namedef{\bb1@stripslash#1}\bb1@tempb{\bb1@tempe}}%
121         \catcode64=\the\catcode64\relax}% Restore @
122       \else
123         \let\bb1@tempc\@empty % Not \relax
124       \fi
125       \bb1@exp{%      For the 'uplevel' assignments
126     \endgroup
127     \bb1@tempc}} % empty or expand to set #1 with changes
128 \fi

```

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

```

129 \def\bb1@ifsamestring#1#2{%
130   \begingroup
131     \protected@edef\bb1@tempb{#1}%
132     \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
133     \protected@edef\bb1@tempc{#2}%
134     \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
135     \ifx\bb1@tempb\bb1@tempc
136       \aftergroup\@firstoftwo
137     \else
138       \aftergroup\@secondoftwo
139     \fi
140   \endgroup}
141 \chardef\bb1@engine=%
142 \ifx\directlua\undefined
143   \ifx\XeTeXinputencoding\undefined
144     \z@
145   \else

```

```

146     \tw@
147     \fi
148   \else
149     \@ne
150   \fi
151 <</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 .

```

152 <<*Make sure ProvidesFile is defined>> ≡
153 \ifx\ProvidesFile\@undefined
154   \def\ProvidesFile#1[#2 #3 #4]{%
155     \wlog{File: #1 #4 #3 <#2>}%
156     \let\ProvidesFile\@undefined}
157 \fi
158 <</Make sure ProvidesFile is defined>>

```

The following code is used in `babel.sty` and `babel.def`, and loads (only once) the data in `language.dat`.

```

159 <<*Load patterns in luatex>> ≡
160 \ifx\directlua\@undefined\else
161   \ifx\bbl@luapatterns\@undefined
162     \input luababel.def
163   \fi
164 \fi
165 <</Load patterns in luatex>>

```

The following code is used in `babel.def` and `switch.def`.

```

166 <<*Load macros for plain if not LaTeX>> ≡
167 \ifx\AtBeginDocument\@undefined
168   \input plain.def\relax
169 \fi
170 <</Load macros for plain if not LaTeX>>

```

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.

```

171 <<*Define core switching macros>> ≡
172 \ifx\language\@undefined
173   \csname newcount\endcsname\language
174 \fi
175 <</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` To add languages to \TeX 's memory plain \TeX version 3.0 supplies `\newlanguage`, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`. For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefore `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain \TeX version 3.0.

For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`. Plain \TeX version 3.0 uses `\count 19` for this purpose.

```

176 <<*Define core switching macros>> ≡
177 \ifx\newlanguage\undefined
178   \csname newcount\endcsname\last@language
179   \def\addlanguage#1{%
180     \global\advance\last@language\@ne
181     \ifnum\last@language<\@ccclvi
182       \else
183         \errmessage{No room for a new \string\language!}%
184       \fi
185     \global\chardef#1\last@language
186     \wlog{\string#1 = \string\language\the\last@language}}
187   \else
188     \countdef\last@language=19
189     \def\addlanguage{\alloc@9\language\chardef\@ccclvi}
190   \fi
191 <</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 \TeX 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.

8 The Package File (\TeX , `babel.sty`)

In order to make use of the features of \TeX 2 ϵ , the `babel` system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options whose name is different from that of the `.ldf` file (like variant spellings). It 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.

8.1 base

The first 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 `switch.def` has been loaded (above) and `\AfterBabelLanguage` defined, exits.

```

192 <*package>
193 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
194 \ProvidesPackage{babel}[\<<date>> \<<version>> The Babel package]
195 \@ifpackagewith{babel}{debug}
196   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
197    \let\bbl@debug\@firstofone}
198   {\providecommand\bbl@trace[1]{}%
199    \let\bbl@debug\@gobble}

```

```

200 \ifx\bbl@switchflag\undefined % Prevent double input
201   \let\bbl@switchflag\relax
202   \input switch.def\relax
203 \fi
204 <<Load patterns in luatex>>
205 <<Basic macros>>
206 \def\AfterBabelLanguage#1{%
207   \global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}%

```

If the format created a list of loaded languages (in \bbl@languages), get the name of the 0-th to show the actual language used.

```

208 \ifx\bbl@languages\undefined\else
209   \begingroup
210     \catcode'\^^I=12
211     \@ifpackagewith{babel}{showlanguages}{%
212       \begingroup
213         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
214         \wlog{<*languages>}%
215         \bbl@languages
216         \wlog{</languages>}%
217       \endgroup}{%
218     \endgroup
219     \def\bbl@elt#1#2#3#4{%
220       \ifnum#2=\z@
221         \gdef\bbl@nulllanguage{#1}%
222         \def\bbl@elt##1##2##3##4{}%
223       \fi}%
224     \bbl@languages
225 \fi
226 \ifodd\bbl@engine
227   \def\bbl@activate@preotf{%
228     \let\bbl@activate@preotf\relax % only once
229     \directlua{
230       Babel = Babel or {}
231       %
232       function Babel.pre_otfload_v(head)
233         if Babel.numbers and Babel.digits_mapped then
234           head = Babel.numbers(head)
235         end
236         if Babel.bidi_enabled then
237           head = Babel.bidi(head, false, dir)
238         end
239         return head
240       end
241       %
242       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
243         if Babel.numbers and Babel.digits_mapped then
244           head = Babel.numbers(head)
245         end
246         if Babel.bidi_enabled then
247           head = Babel.bidi(head, false, dir)
248         end
249         return head
250       end
251       %
252       luatexbase.add_to_callback('pre_linebreak_filter',
253         Babel.pre_otfload_v,
254         'Babel.pre_otfload_v',
255       luatexbase.priority_in_callback('pre_linebreak_filter',

```



```

256         'luaotfload.node_processor') or nil)
257     %
258     luatexbase.add_to_callback('hpack_filter',
259         Babel.pre_otfload_h,
260         'Babel.pre_otfload_h',
261         luatexbase.priority_in_callback('hpack_filter',
262             'luaotfload.node_processor') or nil)
263     }}
264     \let\bbl@tempa\relax
265     \@ifpackagewith{babel}{bidi=basic}%
266     {\def\bbl@tempa{basic}}%
267     {\@ifpackagewith{babel}{bidi=basic-r}%
268         {\def\bbl@tempa{basic-r}}}%
269     {}}
270     \ifx\bbl@tempa\relax\else
271         \let\bbl@beforeforeign\leavevmode
272         \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
273         \RequirePackage{luatexbase}%
274         \directlua{
275             require('babel-data-bidi.lua')
276             require('babel-bidi-\bbl@tempa.lua')
277         }
278         \bbl@activate@preotf
279     \fi
280 \fi

```

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. Useful for old versions of polyglossia, too.

```

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

```

8.2 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no modifiers have been given, the former is `\relax`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\@for` or `load keyval`, for example.

```

296 \bbl@trace{key=value and another general options}
297 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
298 \def\bbl@tempb#1.#2{%
299     #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
300 \def\bbl@tempd#1.#2\@nnil{%
301     \ifx\@empty#2%
302         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%

```

```

303 \else
304   \in@{=}{#1}\ifin@
305   \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
306 \else
307   \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
308   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
309 \fi
310 \fi}
311 \let\bbl@tempc\@empty
312 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
313 \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.

```

314 \DeclareOption{KeepShorthandsActive}{}
315 \DeclareOption{activeacute}{}
316 \DeclareOption{activegrave}{}
317 \DeclareOption{debug}{}
318 \DeclareOption{noconfigs}{}
319 \DeclareOption{showlanguages}{}
320 \DeclareOption{silent}{}
321 \DeclareOption{mono}{}
322 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
323 % Don't use. Experimental:
324 \newif\ifbbl@single
325 \DeclareOption{selectors=off}{\bbl@singletrue}
326 <<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.

```

327 \let\bbl@opt@shorthands\@nnil
328 \let\bbl@opt@config\@nnil
329 \let\bbl@opt@main\@nnil
330 \let\bbl@opt@headfoot\@nnil
331 \let\bbl@opt@layout\@nnil

```

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

```

332 \def\bbl@tempa#1=#2\bbl@tempa{%
333   \bbl@csarg\ifx{opt@#1}\@nnil
334   \bbl@csarg\edef{opt@#1}{#2}%
335 \else
336   \bbl@error{%
337     Bad option `#1=#2'. Either you have misspelled the\\%
338     key or there is a previous setting of `#1'-%
339     Valid keys are `shorthands', `config', `strings', `main',\\%
340     `headfoot', `safe', `math', among others.}
341 \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.

```

342 \let\bbl@language@opts\@empty

```

```

343 \DeclareOption*{%
344   \bbl@xin@{\string=}{\CurrentOption}%
345   \ifin@
346     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
347   \else
348     \bbl@add@list\bbl@language@opts{\CurrentOption}%
349   \fi}

```

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

```

350 \ProcessOptions*

```

8.3 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=...`

```

351 \bbl@trace{Conditional loading of shorthands}
352 \def\bbl@sh@string#1{%
353   \ifx#1\@empty\else
354     \ifx#1t\string-%
355     \else\ifx#1c\string,%
356     \else\string#1%
357   \fi\fi
358   \expandafter\bbl@sh@string
359 \fi}
360 \ifx\bbl@opt@shorthands\@nnil
361   \def\bbl@ifshorthand#1#2#3{#2}%
362 \else\ifx\bbl@opt@shorthands\@empty
363   \def\bbl@ifshorthand#1#2#3{#3}%
364 \else

```

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

```

365   \def\bbl@ifshorthand#1{%
366     \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
367     \ifin@
368       \expandafter\@firstoftwo
369     \else
370       \expandafter\@secondoftwo
371   \fi}

```

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

```

372   \edef\bbl@opt@shorthands{%
373     \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.

```

374   \bbl@ifshorthand{'}%
375   {\PassOptionsToPackage{activeacute}{babel}}{}
376   \bbl@ifshorthand{`}%
377   {\PassOptionsToPackage{activegrave}{babel}}{}
378 \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.

```

379 \ifx\bbl@opt@headfoot\@nnil\else
380   \g@addto@macro\@resetactivechars{%
381     \set@typeset@protect
382     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
383     \let\protect\noexpand}
384 \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.

```

385 \ifx\bbl@opt@safe\undefined
386   \def\bbl@opt@safe{BR}
387 \fi
388 \ifx\bbl@opt@main\@nnil\else
389   \edef\bbl@language@opts{%
390     \ifx\bbl@language@opts\empty\else\bbl@language@opts,\fi
391     \bbl@opt@main}
392 \fi

```

For layout an auxiliary macro is provided, available for packages and language styles.

```

393 \bbl@trace{Defining IfBabelLayout}
394 \ifx\bbl@opt@layout\@nnil
395   \newcommand\IfBabelLayout[3]{#3}%
396 \else
397   \newcommand\IfBabelLayout[1]{%
398     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
399     \ifin@
400       \expandafter\@firstoftwo
401     \else
402       \expandafter\@secondoftwo
403     \fi}
404 \fi

```

8.4 Language options

Languages are loaded when processing the corresponding option *except* if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs the `ldf` file and does some additional checks (`\input` works, too, but possible errors are not caught).

```

405 \bbl@trace{Language options}
406 \let\bbl@afterlang\relax
407 \let\BabelModifiers\relax
408 \let\bbl@loaded\@empty
409 \def\bbl@load@language#1{%
410   \InputIfFileExists{#1.ldf}%
411   {\edef\bbl@loaded{\CurrentOption
412     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
413     \expandafter\let\expandafter\bbl@afterlang
414     \csname\CurrentOption.ldf-h@@k\endcsname
415     \expandafter\let\expandafter\BabelModifiers
416     \csname\bbl@mod@\CurrentOption\endcsname}%
417   {\bbl@error{%
418     Unknown option '\CurrentOption'. Either you misspelled it\\%
419     or the language definition file \CurrentOption.ldf was not found}}{%
420     Valid options are: shorthands=, KeepShorthandsActive,\\%
421     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
422     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

Now, we set language options whose names are different from `ldf` files.

```

423 \def\bbl@try@load@lang#1#2#3{%
424   \IfFileExists{\CurrentOption.ldf}%
425     {\bbl@load@language{\CurrentOption}}%
426     {#1\bbl@load@language{#2}#3}}
427 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}}{}
428 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}}{}
429 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}}{}
430 \DeclareOption{hebrew}{%
431   \input{rlbabel.def}%
432   \bbl@load@language{hebrew}}
433 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}{}
434 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}{}
435 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}}{}
436 \DeclareOption{polutonikogreek}{%
437   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
438 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}}{}
439 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}{}
440 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}}{}
441 \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.

```

442 \ifx\bbl@opt@config\@nnil
443   \@ifpackagewith{babel}{noconfigs}}{}%
444   {\InputIfFileExists{bblopts.cfg}%
445     {\typeout{*****^J%
446               * Local config file bblopts.cfg used^^J%
447               *}}%
448     {}}%
449 \else
450   \InputIfFileExists{\bbl@opt@config.cfg}%
451   {\typeout{*****^J%
452             * Local config file \bbl@opt@config.cfg used^^J%
453             *}}%
454   {\bbl@error{%
455     Local config file '\bbl@opt@config.cfg' not found}%
456     Perhaps you misspelled it.}}%
457 \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.

```

458 \bbl@for\bbl@tempa\bbl@language@opts{%
459   \bbl@ifunset{ds@\bbl@tempa}%
460   {\edef\bbl@tempb{%
461     \noexpand\DeclareOption
462     {\bbl@tempa}%
463     {\noexpand\bbl@load@language{\bbl@tempa}}}%
464     \bbl@tempb}%
465   \@empty}

```

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.

```

466 \bbl@foreach\@classoptionslist{%
467   \bbl@ifunset{ds@#1}%
468   {\IfFileExists{#1.ldf}%
469     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
470     {}}%
471   {}}

```

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

```

472 \ifx\bbl@opt@main\@nnil\else
473   \expandafter
474   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
475   \DeclareOption{\bbl@opt@main}{}
476 \fi

```

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

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

```

477 \def\AfterBabelLanguage#1{%
478   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}
479   \DeclareOption*{}
480   \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`.

```

481 \ifx\bbl@opt@main\@nnil
482   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
483   \let\bbl@tempc\empty
484   \bbl@for\bbl@tempb\bbl@tempa{%
485     \bbl@xin@{\bbl@tempb},{\bbl@loaded},%
486     \ifin@{\edef\bbl@tempc{\bbl@tempb}}\fi}
487   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
488   \expandafter\bbl@tempa\bbl@loaded,\@nnil
489   \ifx\bbl@tempb\bbl@tempc\else
490     \bbl@warning{%
491       Last declared language option is '\bbl@tempc',\%
492       but the last processed one was '\bbl@tempb'.\%
493       The main language cannot be set as both a global\%
494       and a package option. Use 'main=\bbl@tempc' as\%
495       option. Reported}%
496   \fi
497 \else
498   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
499   \ExecuteOptions{\bbl@opt@main}
500   \DeclareOption*{}
501   \ProcessOptions*
502 \fi
503 \def\AfterBabelLanguage{%
504   \bbl@error
505   {Too late for \string\AfterBabelLanguage}%
506   {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.

```

507 \ifx\bbl@main@language\@undefined
508   \bbl@info{%
509     You haven't specified a language. I'll use 'nil'\\%
510     as the main language. Reported}
511   \bbl@load@language{nil}
512 \fi
513 </package>
514 <*core>

```

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

The kernel of the babel system is stored in either hyphen.cfg or switch.def and babel.def. The file babel.def contains most of the code, while switch.def defines the language-switching commands; both can be read at run time. 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 (by default, it also inputs switch.def, for “historical reasons”, but it is not necessary). When babel.def is loaded it checks if the current version of switch.def is in the format; if not, it is loaded. A further file, babel.sty, contains \LaTeX -specific stuff. Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only. Plain formats based on etex (etex, xetex, luatex) don't load hyphen.cfg but etex.src, which follows a different naming convention, so we need to define the babel names. It presumes language.def exists and it is the same file used when formats were created.

9.1 Tools

```

515 \ifx\ldf@quit\@undefined
516 \else
517   \expandafter\endinput
518 \fi
519 <<Make sure ProvidesFile is defined>>
520 \ProvidesFile{babel.def}[[<<date>>] [<<version>>] Babel common definitions]
521 <<Load macros for plain if not LaTeX>>

```

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

```

522 \ifx\bbl@ifshorthand\@undefined
523   \let\bbl@opt@shorthands\@nnil
524   \def\bbl@ifshorthand#1#2#3{#2}%
525   \let\bbl@language@opts\@empty
526   \ifx\babeloptionstrings\@undefined
527     \let\bbl@opt@strings\@nnil
528   \else
529     \let\bbl@opt@strings\babeloptionstrings
530   \fi
531   \def\BabelStringsDefault{generic}
532   \def\bbl@tempa{normal}
533   \ifx\babeloptionmath\bbl@tempa
534     \def\bbl@mathnormal{\noexpand\textormath}
535   \fi

```

```

536 \def\AfterBabelLanguage#1#2{}
537 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
538 \let\bbl@afterlang\relax
539 \def\bbl@opt@safe{BR}
540 \ifx\uclclist\undefined\let\uclclist\empty\fi
541 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
542 \expandafter\newif\csname ifbbl@single\endcsname
543 \fi

And continue.
544 \ifx\bbl@switchflag\undefined % Prevent double input
545 \let\bbl@switchflag\relax
546 \input switch.def\relax
547 \fi
548 \bbl@trace{Compatibility with language.def}
549 \ifx\bbl@languages\undefined
550 \ifx\directlua\undefined
551 \openin1 = language.def
552 \ifeof1
553 \closein1
554 \message{I couldn't find the file language.def}
555 \else
556 \closein1
557 \begingroup
558 \def\addLanguage#1#2#3#4#5{%
559 \expandafter\ifx\csname lang@#1\endcsname\relax\else
560 \global\expandafter\let\csname l@#1\expandafter\endcsname
561 \csname lang@#1\endcsname
562 \fi}%
563 \def\uselanguage#1{}\%
564 \input language.def
565 \endgroup
566 \fi
567 \fi
568 \chardef\l@english\z@
569 \fi
570 <<Load patterns in luatex>>
571 <<Basic macros>>

```

\addto For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro \addto is introduced. It takes two arguments, a *<control sequence>* and T_EX-code to be added to the *<control sequence>*.

If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to \relax, in which case a circular definition results. The net result is a stack overflow. Otherwise the replacement text for the *<control sequence>* is expanded and stored in a token register, together with the T_EX-code to be added. Finally the *<control sequence>* is redefined, using the contents of the token register.

```

572 \def\addto#1#2{%
573 \ifx#1\undefined
574 \def#1{#2}%
575 \else
576 \ifx#1\relax
577 \def#1{#2}%
578 \else
579 {\toks@\expandafter{#1#2}%
580 \xdef#1{\the\toks@}}%
581 \fi
582 \fi}

```


The macro `\initiate@active@char` takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character.

```
583 \def\bbl@withactive#1#2{%
584   \begingroup
585   \lccode`~=`#2\relax
586   \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).

Because we need to redefine a number of commands we define the command `\bbl@redefine` which takes care of this. It creates a new control sequence, `\org@...`

```
587 \def\bbl@redefine#1{%
588   \edef\bbl@tempa{\bbl@stripslash#1}%
589   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
590   \expandafter\def\csname\bbl@tempa\endcsname}
```

This command should only be used in the preamble of the document.

```
591 \@onlypreamble\bbl@redefine
```

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

```
592 \def\bbl@redefine@long#1{%
593   \edef\bbl@tempa{\bbl@stripslash#1}%
594   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
595   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
596 \@onlypreamble\bbl@redefine@long
```

`\bbl@redefineroobust` For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo_`. So it is necessary to check whether `\foo_` exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define `\foo_`.

```
597 \def\bbl@redefineroobust#1{%
598   \edef\bbl@tempa{\bbl@stripslash#1}%
599   \bbl@ifunset{\bbl@tempa\space}%
600   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
601     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
602   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
603   \@namedef{\bbl@tempa\space}}
```

This command should only be used in the preamble of the document.

```
604 \@onlypreamble\bbl@redefineroobust
```

9.2 Hooks

Note they are loaded in `babel.def`. `switch.def` only provides a “hook” for hooks (with a default value which is a no-op, below). Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is intended for developers, after all. `\bbl@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```
605 \bbl@trace{Hooks}
606 \newcommand\AddBabelHook[3][{}]{%
607   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}}%
608   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
```

```

609 \expandafter\bb1@tempa\bb1@evargs,#3=,\@empty
610 \bb1@ifunset{bb1@ev@#2@#3@#1}%
611 {\bb1@csarg\bb1@add{ev@#3@#1}{\bb1@elt{#2}}}%
612 {\bb1@csarg\let{ev@#2@#3@#1}\relax}%
613 \bb1@csarg\newcommand{ev@#2@#3@#1}{\bb1@tempb}}
614 \newcommand\EnableBabelHook[1]{\bb1@csarg\let{hk@#1}\@firstofone}
615 \newcommand\DisableBabelHook[1]{\bb1@csarg\let{hk@#1}\@gobble}
616 \def\bb1@usehooks#1#2{%
617 \def\bb1@elt#1{%
618 \@nameuse{bb1@hk@#1}{\@nameuse{bb1@ev@#1@#1@#2}}}%
619 \@nameuse{bb1@ev@#1@}%
620 \ifx\language\undefined\else % Test required for Plain (?)
621 \def\bb1@elt#1{%
622 \@nameuse{bb1@hk@#1}{\@nameuse{bb1@ev@#1@#1@\language}#2}}%
623 \@nameuse{bb1@ev@#1@\language}%
624 \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).

```

625 \def\bb1@evargs{% <- don't delete this comma
626 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
627 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
628 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
629 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
630 beforestart=0}

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bb1@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 `\bb1@e@<language>` contains `\bb1@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bb1@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.

```

631 \bb1@trace{Defining babelensure}
632 \newcommand\babelensure[2][{}]{% TODO - revise test files
633 \AddBabelHook{babel-ensure}{afterextras}{%
634 \ifcase\bb1@select@type
635 \@nameuse{bb1@e@\language}%
636 \fi}%
637 \begin{group}
638 \let\bb1@ens@include\@empty
639 \let\bb1@ens@exclude\@empty
640 \def\bb1@ens@fontenc{\relax}%
641 \def\bb1@tempb##1{%
642 \ifx\@empty##1\else\noexpand##1\expandafter\bb1@tempb\fi}%
643 \edef\bb1@tempa{\bb1@tempb##1\@empty}%
644 \def\bb1@tempb##1=##2\@{ \@namedef{bb1@ens@##1}{##2}}%
645 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
646 \def\bb1@tempc{\bb1@ensure}%
647 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
648 \expandafter{\bb1@ens@include}}%
649 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
650 \expandafter{\bb1@ens@exclude}}%

```

```

651 \toks@\expandafter{\bbl@tempc}%
652 \bbl@exp{%
653 \endgroup
654 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
655 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
656 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
657 \ifx##1\undefined % 3.32 - Don't assume the macros exists
658 \edef##1{\noexpand\bbl@nocaption
659 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}}%
660 \fi
661 \ifx##1\@empty\else
662 \in@{##1}{#2}%
663 \ifin\else
664 \bbl@ifunset{\bbl@ensure@\language\language}%
665 {\bbl@exp{%
666 \\\DeclareRobustCommand\<bbl@ensure@\language\language>[1]{%
667 \\\foreignlanguage{\language\language}%
668 {\ifx\relax#3\else
669 \\\fontencoding{#3}\selectfont
670 \fi
671 #####1}}}}}%
672 {}%
673 \toks@\expandafter{##1}%
674 \edef##1{%
675 \bbl@csarg\noexpand{\ensure@\language\language}%
676 {\the\toks@}}}%
677 \fi
678 \expandafter\bbl@tempb
679 \fi}%
680 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
681 \def\bbl@tempa##1{% elt for include list
682 \ifx##1\@empty\else
683 \bbl@csarg\in@{\ensure@\language\language\expandafter}\expandafter{##1}%
684 \ifin\else
685 \bbl@tempb##1\@empty
686 \fi
687 \expandafter\bbl@tempa
688 \fi}%
689 \bbl@tempa#1\@empty}
690 \def\bbl@captionslist{%
691 \prefacename\refname\abstractname\bibname\chaptername\appendixname
692 \contentsname\listfigurename\listtablename\indexname\figurename
693 \tablename\partname\enclname\ccname\headtoname\pagename\seename
694 \alsoname\proofname\glossaryname}

```

9.3 Setting up language files

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

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

Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions

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

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

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

Finally we check `\originalTeX`.

```

695 \bbl@trace{Macros for setting language files up}
696 \def\bbl@ldfinit{%
697   \let\bbl@screset\@empty
698   \let\BabelStrings\bbl@opt@string
699   \let\BabelOptions\@empty
700   \let\BabelLanguages\relax
701   \ifx\originalTeX\@undefined
702     \let\originalTeX\@empty
703   \else
704     \originalTeX
705   \fi}
706 \def\LdfInit#1#2{%
707   \chardef\atcatcode=\catcode`\@
708   \catcode`\@=11\relax
709   \chardef\eqcatcode=\catcode`\=
710   \catcode`\==12\relax
711   \expandafter\if\expandafter\@backslashchar
712     \expandafter\@car\string#2\@nil
713   \ifx#2\@undefined\else
714     \ldf@quit{#1}%
715   \fi
716 \else
717   \expandafter\ifx\csname#2\endcsname\relax\else
718     \ldf@quit{#1}%
719   \fi
720 \fi
721 \bbl@ldfinit}

```

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

```

722 \def\ldf@quit#1{%
723   \expandafter\main@language\expandafter{#1}%
724   \catcode`\@=\atcatcode \let\atcatcode\relax
725   \catcode`\==\eqcatcode \let\eqcatcode\relax
726   \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.

```

727 \def\bbl@afterldf#1{%
728   \bbl@afterlang
729   \let\bbl@afterlang\relax
730   \let\BabelModifiers\relax
731   \let\bbl@screset\relax}%
732 \def\ldf@finish#1{%
733   \loadlocalcfg{#1}%

```

```

734 \bbl@afterldf{#1}%
735 \expandafter\main@language\expandafter{#1}%
736 \catcode`\@=\atcatcode \let\atcatcode\relax
737 \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

After the preamble of the document the commands `\LdfInit`, `\ldf@quit` and `\ldf@finish` are no longer needed. Therefore they are turned into warning messages in \LaTeX .

```

738 \@onlypreamble\LdfInit
739 \@onlypreamble\ldf@quit
740 \@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.

```

741 \def\main@language#1{%
742   \def\bbl@main@language{#1}%
743   \let\language\name\bbl@main@language
744   \bbl@id@assign
745   \bbl@patterns{\language}}

```

We also have to make sure that some code gets executed at the beginning of the document. Languages do not set `\pagedir`, so we set here for the whole document to the main `\bodydir`.

```

746 \def\bbl@beforestart{%
747   \bbl@usehooks{beforestart}{}%
748   \global\let\bbl@beforestart\relax}
749 \AtBeginDocument{%
750   \@nameuse{bbl@beforestart}%
751   \if@filesw
752     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
753   \fi
754   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
755   \ifbbl@single % must go after the line above
756     \renewcommand\selectlanguage[1]{}%
757     \renewcommand\foreignlanguage[2]{#2}%
758     \global\let\babel@aux\@gobbletwo % Also as flag
759   \fi
760   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

761 \def\select@language@x#1{%
762   \ifcase\bbl@select@type
763     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
764   \else
765     \select@language{#1}%
766   \fi}

```

9.4 Shorthands

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

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

```

767 \bbl@trace{Shorhands}
768 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
769 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
770 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
771 \ifx\nfss@catcodes\undefined\else % TODO - same for above
772 \begingroup
773 \catcode`#1\active
774 \nfss@catcodes
775 \ifnum\catcode`#1=\active
776 \endgroup
777 \bbl@add\nfss@catcodes{\@makeother#1}%
778 \else
779 \endgroup
780 \fi
781 \fi}

```

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

```

782 \def\bbl@remove@special#1{%
783 \begingroup
784 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
785 \else\noexpand##1\noexpand##2\fi}%
786 \def\do{\x\do}%
787 \def\@makeother{\x\@makeother}%
788 \edef\x{\endgroup
789 \def\noexpand\dospecials{\dospecials}%
790 \expandafter\ifx\csname @sanitize\endcsname\relax\else
791 \def\noexpand\@sanitize{\@sanitize}%
792 \fi}%
793 \x}

```

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char⟨char⟩` to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char⟨char⟩` by default (`⟨char⟩` being the character to be made active). Later its definition can be changed to expand to `\active@char⟨char⟩` by calling `\bbl@activate{⟨char⟩}`.

For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines `"` as `\active@prefix "\active@char"` (where the first `"` is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original `"`); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char`).

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, `<level>@group`, `<level>@active` and `<next-level>@active` (except in system).

```

794 \def\bbl@active@def#1#2#3#4{%
795 \namedef{#3#1}{%
796 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
797 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
798 \else

```

```

799     \bbl@afterfi\csname#2@sh@#1@\endcsname
800     \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.

```

801 \long\@namedef{#3@arg#1}##1{%
802   \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
803     \bbl@afterelse\csname#4#1\endcsname##1%
804   \else
805     \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
806   \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.

```

807 \def\initiate@active@char#1{%
808   \bbl@ifunset{active@char\string#1}%
809   {\bbl@withactive
810    {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
811   {}}

```

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`).

```

812 \def\@initiate@active@char#1#2#3{%
813   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
814   \ifx#1\@undefined
815     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
816   \else
817     \bbl@csarg\let{oridef@#2}#1%
818     \bbl@csarg\edef{oridef@#2}{%
819       \let\noexpand#1%
820       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
821   \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*").

```

822 \ifx#1#3\relax
823   \expandafter\let\csname normal@char#2\endcsname#3%
824 \else
825   \bbl@info{Making #2 an active character}%
826   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
827   \@namedef{normal@char#2}{%
828     \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
829   \else
830     \@namedef{normal@char#2}{#3}%
831   \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).

```

832 \bbl@restoreactive{#2}%
833 \AtBeginDocument{%
834   \catcode`#2\active
835   \if@filesw
836     \immediate\write\@mainaux{\catcode`\string#2\active}%
837   \fi}%
838 \expandafter\bbl@add@special\csname#2\endcsname
839 \catcode`#2\active
840 \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⟩`).

```

841 \let\bbl@tempa\@firstoftwo
842 \if\string^#2%
843   \def\bbl@tempa{\noexpand\textormath}%
844 \else
845   \ifx\bbl@mathnormal\@undefined\else
846     \let\bbl@tempa\bbl@mathnormal
847   \fi
848 \fi
849 \expandafter\edef\csname active@char#2\endcsname{%
850   \bbl@tempa
851     {\noexpand\if@safe@actives
852       \noexpand\expandafter
853       \expandafter\noexpand\csname normal@char#2\endcsname
854     \noexpand\else
855       \noexpand\expandafter
856       \expandafter\noexpand\csname bbl@doactive#2\endcsname
857     \noexpand\fi}%
858   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
859 \bbl@csarg\edef{doactive#2}{%
860   \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!).

```

861 \bbl@csarg\edef{active@#2}{%
862   \noexpand\active@prefix\noexpand#1%
863   \expandafter\noexpand\csname active@char#2\endcsname}%
864 \bbl@csarg\edef{normal@#2}{%
865   \noexpand\active@prefix\noexpand#1%
866   \expandafter\noexpand\csname normal@char#2\endcsname}%
867 \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.

```

868 \bbl@active@def#2\user@group{user@active}{language@active}%
869 \bbl@active@def#2\language@group{language@active}{system@active}%
870 \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 T_EX would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```
871 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
872   {\expandafter\noexpand\csname normal@char#2\endcsname}%
873 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
874   {\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.

```
875 \if\string'#2%
876   \let\prim@s\bbl@prim@s
877   \let\active@math@prime#1%
878 \fi
879 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
880 <<{*More package options}>> ≡
881 \DeclareOption{math=active}{}
882 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
883 <</More package options>>
```

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

```
884 \@ifpackagewith{babel}{KeepShorthandsActive}%
885   {\let\bbl@restoreactive\@gobble}%
886   {\def\bbl@restoreactive#1{%
887     \bbl@exp{%
888       \\\AfterBabelLanguage\\CurrentOption
889       {\catcode`#1=\the\catcode`#1\relax}%
890       \\\AtEndOfPackage
891       {\catcode`#1=\the\catcode`#1\relax}}}%
892   \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.

```
893 \def\bbl@sh@select#1#2{%
894   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
895     \bbl@afterelse\bbl@scndcs
896   \else
897     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
898   \fi}
```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protects` the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar`: (when the double colon was the active character to be dealt with). There are

two definitions, depending of \ifincsname is available. If there is, the expansion will be more robust.

```

899 \begingroup
900 \bbl@ifunset{ifincsname}%
901   {\gdef\active@prefix#1{%
902     \ifx\protect\@typeset@protect
903     \else
904       \ifx\protect\@unexpandable@protect
905         \noexpand#1%
906       \else
907         \protect#1%
908       \fi
909     \expandafter\@gobble
910   \fi}}
911   {\gdef\active@prefix#1{%
912     \ifincsname
913       \string#1%
914     \expandafter\@gobble
915   \else
916     \ifx\protect\@typeset@protect
917     \else
918       \ifx\protect\@unexpandable@protect
919         \noexpand#1%
920       \else
921         \protect#1%
922       \fi
923     \expandafter\expandafter\expandafter\@gobble
924     \fi
925   \fi}}
926 \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⟩.

```

927 \newif\if@safe@actives
928 \@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.

```

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

```

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

```

930 \def\bbl@activate#1{%
931   \bbl@withactive{\expandafter\let\expandafter}#1%
932   \csname bbl@active@\string#1\endcsname}
933 \def\bbl@deactivate#1{%
934   \bbl@withactive{\expandafter\let\expandafter}#1%
935   \csname bbl@normal@\string#1\endcsname}

```

\bbl@firstcs These macros have two arguments. They use one of their arguments to build a control
\bbl@scndcs sequence from.

```

936 \def\bbl@firstcs#1#2{\csname#1\endcsname}
937 \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.

```

938 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
939 \def\@decl@short#1#2#3\@nil#4{%
940   \def\bbl@tempa{#3}%
941   \ifx\bbl@tempa\@empty
942     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
943     \bbl@ifunset{#1@sh@\string#2@}\{}%
944     {\def\bbl@tempa{#4}%
945       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
946       \else
947         \bbl@info
948         {Redefining #1 shorthand \string#2\\
949          in language \CurrentOption}%
950       \fi}%
951     \@namedef{#1@sh@\string#2@}{#4}%
952   \else
953     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
954     \bbl@ifunset{#1@sh@\string#2@\string#3@}\{}%
955     {\def\bbl@tempa{#4}%
956       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
957       \else
958         \bbl@info
959         {Redefining #1 shorthand \string#2\string#3\\
960          in language \CurrentOption}%
961       \fi}%
962     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
963   \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.

```

964 \def\textormath{%
965   \ifmmode
966     \expandafter\@secondoftwo
967   \else
968     \expandafter\@firstoftwo
969   \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’.

```

970 \def\user@group{user}
971 \def\language@group{english}
972 \def\system@group{system}

```

`\useshorthands` This is the user level command to tell \TeX that user level shorthands will be used in the document. It takes one argument, the character that starts a shorthand. First note that this is user level, and then initialize and activate 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.

```

973 \def\useshorthands{%
974   \ifstar\bbl@usesh@s{\bbl@usesh@x{}}
975 \def\bbl@usesh@s#1{%
976   \bbl@usesh@x
977   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
978   {#1}}
979 \def\bbl@usesh@x#1#2{%
980   \bbl@ifshorthand{#2}%
981   {\def\user@group{user}%
982     \initiate@active@char{#2}%
983     #1%
984     \bbl@activate{#2}}%
985   {\bbl@error
986     {Cannot declare a shorthand turned off (\string#2)}
987     {Sorry, but you cannot use shorthands which have been\%
988       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.

```

989 \def\user@language@group{user@\language@group}
990 \def\bbl@set@user@generic#1#2{%
991   \bbl@ifunset{user@generic@active#1}%
992   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
993     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
994     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
995       \expandafter\noexpand\csname normal@char#1\endcsname}%
996     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
997       \expandafter\noexpand\csname user@active#1\endcsname}}%
998   \@empty}
999 \newcommand\defineshorthand[3][user]{%
1000   \edef\bbl@tempa{\zap@space#1 \@empty}%
1001   \bbl@for\bbl@tempb\bbl@tempa{%
1002     \if*\expandafter\@car\bbl@tempb\@nil
1003       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1004       \@expandtwoargs
1005       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1006     \fi
1007     \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.

```

1008 \def\languageshorthands#1{\def\language@group{#1}}

```

`\aliasshorthand` First the new shorthand needs to be initialized,

```

1009 \def\aliasshorthand#1#2{%
1010   \bbl@ifshorthand{#2}%
1011   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1012     \ifx\document\@notprerr
1013       \@notshorthand{#2}%
1014     \else
1015       \initiate@active@char{#2}%

```

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

littest to \active@char".

```

1016      \expandafter\let\csname active@char\string#2\expandafter\endcsname
1017      \csname active@char\string#1\endcsname
1018      \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1019      \csname normal@char\string#1\endcsname
1020      \bbl@activate{#2}%
1021      \fi
1022      \fi}%
1023      {\bbl@error
1024      {Cannot declare a shorthand turned off (\string#2)}
1025      {Sorry, but you cannot use shorthands which have been\\%
1026      turned off in the package options}}}
```

\@notshorthand

```

1027 \def\@notshorthand#1{%
1028   \bbl@error{%
1029     The character '\string #1' should be made a shorthand character;\\%
1030     add the command \string\useshorthands\string{#1\string} to
1031     the preamble.\\%
1032     I will ignore your instruction}%
1033   {You may proceed, but expect unexpected results}}}
```

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

```

1034 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1035 \DeclareRobustCommand*\shorthandoff{%
1036   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1037 \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.

```

1038 \def\bbl@switch@sh#1#2{%
1039   \ifx#2\@nnil\else
1040     \bbl@ifunset{\bbl@active@\string#2}%
1041     {\bbl@error
1042      {I cannot switch '\string#2' on or off--not a shorthand}%
1043      {This character is not a shorthand. Maybe you made\\%
1044      a typing mistake? I will ignore your instruction}}}%
1045     {\ifcase#1%
1046       \catcode'#212\relax
1047       \or
1048       \catcode'#2\active
1049       \or
1050       \csname bbl@oricat@\string#2\endcsname
1051       \csname bbl@oridef@\string#2\endcsname
1052       \fi}%
1053     \bbl@afterfi\bbl@switch@sh#1%
1054   \fi}
```

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

```

1055 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1056 \def\bbl@putsh#1{%
1057   \bbl@ifunset{\bbl@active@\string#1}%
1058   {\bbl@putsh@i#1\@empty\@nnil}%
1059   {\csname bbl@active@\string#1\endcsname}}
1060 \def\bbl@putsh@i#1#2\@nnil{%
1061   \csname\language\sh@\string#1@%
1062     \ifx\@empty#2\else\string#2\fi\endcsname}
1063 \ifx\bbl@opt@shorthands\@nnil\else
1064   \let\bbl@s@initiate@active@char\initiate@active@char
1065   \def\initiate@active@char#1{%
1066     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1067   \let\bbl@s@switch@sh\bbl@switch@sh
1068   \def\bbl@switch@sh#1#2{%
1069     \ifx#2\@nnil\else
1070       \bbl@afterfi
1071       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1072       \fi}
1073   \let\bbl@s@activate\bbl@activate
1074   \def\bbl@activate#1{%
1075     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1076   \let\bbl@s@deactivate\bbl@deactivate
1077   \def\bbl@deactivate#1{%
1078     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1079 \fi

```

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

```

1080 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

\bbl@prim@s One of the internal macros that are involved in substituting \prime for each right quote in mathmode is \prim@s. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.

\bbl@pr@m@s

```

1081 \def\bbl@prim@s{%
1082   \prime\futurelet\@let@token\bbl@pr@m@s}
1083 \def\bbl@if@primes#1#2{%
1084   \ifx#1\@let@token
1085     \expandafter\@firstoftwo
1086   \else\ifx#2\@let@token
1087     \bbl@afterelse\expandafter\@firstoftwo
1088   \else
1089     \bbl@afterfi\expandafter\@secondoftwo
1090   \fi\fi}
1091 \begingroup
1092 \catcode`\^=7 \catcode`\*= \active \lccode`\^=\^
1093 \catcode`\'=12 \catcode`\`= \active \lccode`\`= `
1094 \lowercase{%
1095   \gdef\bbl@pr@m@s{%
1096     \bbl@if@primes" "%
1097     \pr@@@s
1098     {\bbl@if@primes*\^ \pr@@@t\egroup}}
1099 \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).

```
1100 \initiate@active@char{~}
1101 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1102 \bbl@activate{~}
```

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

```
1103 \expandafter\def\csname OT1dqpos\endcsname{127}
1104 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1105 \ifx\f@encoding\@undefined
1106   \def\f@encoding{OT1}
1107 \fi
```

9.5 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.

```
1108 \bbl@trace{Language attributes}
1109 \newcommand\languageattribute[2]{%
1110   \def\bbl@tempc{#1}%
1111   \bbl@fixname\bbl@tempc
1112   \bbl@iflanguage\bbl@tempc{%
1113     \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.

```
1114     \ifx\bbl@known@attrs\@undefined
1115       \in@false
1116     \else
1117       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1118     \fi
1119     \ifin@
1120       \bbl@warning{%
1121         You have more than once selected the attribute '##1'\%
1122         for language #1. Reported}%
1123     \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.

```
1124       \bbl@exp{%
1125         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1126       \edef\bbl@tempa{\bbl@tempc-##1}%
1127       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1128       {\csname\bbl@tempc @attr##1\endcsname}%
1129       {\@attrerr{\bbl@tempc}{##1}}%
1130     \fi}}
```

This command should only be used in the preamble of a document.

```
1131 \@onlypreamble\languageattribute
```

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

```
1132 \newcommand*{\@attrerr}[2]{%
1133   \bbl@error
1134   {The attribute #2 is unknown for language #1.}%
1135   {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}`.

```
1136 \def\bbl@declare@ttribute#1#2#3{%
1137   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1138   \ifin@
1139     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1140   \fi
1141   \bbl@add@list\bbl@attributes{#1-#2}%
1142   \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.

```
1143 \def\bbl@ifattributeset#1#2#3#4{%
    First we need to find out if any attributes were set; if not we're done.
```

```
1144   \ifx\bbl@known@attribs\undefined
1145     \in@false
1146   \else
```

The we need to check the list of known attributes.

```
1147   \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1148   \fi
```

When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```
1149   \ifin@
1150     \bbl@afterelse#3%
1151   \else
1152     \bbl@afterfi#4%
1153   \fi
1154 }
```

`\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.

```
1155 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1156   \let\bbl@tempa\@secondoftwo
```


Then we loop over the list of known attributes, trying to find a match.

```
1157 \bbl@loopx\bbl@tempb{#2}{%
1158   \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1159   \ifin@
```

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

```
1160   \let\bbl@tempa\@firstoftwo
1161   \else
1162   \fi}%
```

Finally we execute `\bbl@tempa`.

```
1163 \bbl@tempa
1164 }
```

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

```
1165 \def\bbl@clear@ttribs{%
1166   \ifx\bbl@attributes\@undefined\else
1167     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1168       \expandafter\bbl@clear@ttrib\bbl@tempa.
1169     }%
1170     \let\bbl@attributes\@undefined
1171   \fi}
1172 \def\bbl@clear@ttrib#1-#2.{%
1173   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1174 \AtBeginDocument{\bbl@clear@ttribs}
```

9.6 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 1175 \bbl@trace{Macros for saving definitions}
1176 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1177 \newcount\babel@savecnt
1178 \babel@beginsave
```

`\babel@save` The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX`²⁹. 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.

```
1179 \def\babel@save#1{%
1180   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1181   \toks@\expandafter{\originalTeX\let#1=}
1182   \bbl@exp{%
1183     \def\\originalTeX{\the\toks@<\babel@\number\babel@savecnt>\relax}}%
```

²⁹`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```

1184 \advance\babel@savecnt\@ne}
1185 \def\babel@savevariable#1{%
1186 \toks@\expandafter\originalTeX #1=%}
1187 \bbl@exp{\def\\originalTeX{\the\toks@\the#1\relax}}

```

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The
`\bbl@nonfrenchspacing` command `\bbl@frenchspacing` switches it on when it isn't already in effect and
`\bbl@nonfrenchspacing` switches it off if necessary.

```

1188 \def\bbl@frenchspacing{%
1189 \ifnum\the\sfcodes\@m
1190 \let\bbl@nonfrenchspacing\relax
1191 \else
1192 \frenchspacing
1193 \let\bbl@nonfrenchspacing\nonfrenchspacing
1194 \fi}
1195 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

9.7 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.

```

1196 \bbl@trace{Short tags}
1197 \def\babeltags#1{%
1198 \edef\bbl@tempa{\zap@space#1 \@empty}%
1199 \def\bbl@tempb##1=##2\@{ }%
1200 \edef\bbl@tempc{%
1201 \noexpand\newcommand
1202 \expandafter\noexpand\csname ##1\endcsname{%
1203 \noexpand\protect
1204 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1205 \noexpand\newcommand
1206 \expandafter\noexpand\csname text##1\endcsname{%
1207 \noexpand\foreignlanguage{##2}}
1208 \bbl@tempc}%
1209 \bbl@for\bbl@tempa\bbl@tempa{%
1210 \expandafter\bbl@tempb\bbl@tempa\@{ }

```

9.8 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.

```

1211 \bbl@trace{Hyphens}
1212 \@onlypreamble\babelhyphenation
1213 \AtEndOfPackage{%
1214 \newcommand\babelhyphenation[2][\@empty]{%
1215 \ifx\bbl@hyphenation@\relax
1216 \let\bbl@hyphenation@\@empty
1217 \fi
1218 \ifx\bbl@hyphlist\@empty\else
1219 \bbl@warning{%
1220 You must not intermingle \string\selectlanguage\space and\\%
1221 \string\babelhyphenation\space or some exceptions will not\\%
1222 be taken into account. Reported}%

```

```

1223 \fi
1224 \ifx\@empty#1%
1225 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@space#2}%
1226 \else
1227 \bbl@vforeach{#1}{%
1228 \def\bbl@tempa{##1}%
1229 \bbl@fixname\bbl@tempa
1230 \bbl@iflanguage\bbl@tempa{%
1231 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1232 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1233 \@empty
1234 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1235 #2}}}%
1236 \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip Opt plus Opt`³⁰.

```

1237 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1238 \def\bbl@t@one{T1}
1239 \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`.

```

1240 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1241 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1242 \def\bbl@hyphen{%
1243 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1244 \def\bbl@hyphen@i#1#2{%
1245 \bbl@ifunset{bbl@hy@#1#2\@empty}%
1246 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{-}{#2}}}%
1247 {\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.

```

1248 \def\bbl@usehyphen#1{%
1249 \leavevmode
1250 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1251 \nobreak\hskip\z@skip}
1252 \def\bbl@@usehyphen#1{%
1253 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1254 \def\bbl@hyphenchar{%
1255 \ifnum\hyphenchar\font=\m@ne
1256 \babellnullhyphen
1257 \else
1258 \char\hyphenchar\font
1259 \fi}

```

³⁰ \TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

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.

```
1260 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1261 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1262 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1263 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1264 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1265 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1266 \def\bbl@hy@repeat{%
1267   \bbl@usehyphen{%
1268     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1269 \def\bbl@hy@@repeat{%
1270   \bbl@usehyphen{%
1271     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1272 \def\bbl@hy@empty{\hskip\z@skip}
1273 \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.

```
1274 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}
```

9.9 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.

```
1275 \bbl@trace{Multiencoding strings}
1276 \def\bbl@tglobal#1{\global\let#1#1}
1277 \def\bbl@recatcode#1{%
1278   \@tempcnta="7F
1279   \def\bbl@tempa{%
1280     \ifnum\@tempcnta>"FF\else
1281       \catcode\@tempcnta=#1\relax
1282       \advance\@tempcnta\@ne
1283       \expandafter\bbl@tempa
1284     \fi}%
1285   \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).

```
1286 \@ifpackagewith{babel}{nocase}%
1287   {\let\bbl@patchuclc\relax}%
1288   {\def\bbl@patchuclc%
```

```

1289 \global\let\bbl@patchuclc\relax
1290 \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1291 \gdef\bbl@uclc##1{%
1292   \let\bbl@encoded\bbl@encoded@uclc
1293   \bbl@ifunset{\language @bbl@uclc}% and resumes it
1294   {##1}%
1295   {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1296     \csname\language @bbl@uclc\endcsname}%
1297     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1298   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1299   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
1300 <<(*More package options)>> ≡
1301 \DeclareOption{nocase}{}
1302 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1303 <<(*More package options)>> ≡
1304 \let\bbl@opt@strings\@nnil % accept strings=value
1305 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1306 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1307 \def\BabelStringsDefault{generic}
1308 <</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.

```

1309 \@onlypreamble\StartBabelCommands
1310 \def\StartBabelCommands{%
1311   \begingroup
1312   \bbl@recatcode{11}%
1313   <<Macros local to BabelCommands>>
1314   \def\bbl@provstring##1##2{%
1315     \providecommand##1{##2}%
1316     \bbl@toglobal##1}%
1317   \global\let\bbl@scafter\@empty
1318   \let\StartBabelCommands\bbl@startcmds
1319   \ifx\BabelLanguages\relax
1320     \let\BabelLanguages\CurrentOption
1321   \fi
1322   \begingroup
1323   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1324   \StartBabelCommands}
1325 \def\bbl@startcmds{%
1326   \ifx\bbl@screset\@nnil\else
1327     \bbl@usehooks{stopcommands}{}%
1328   \fi
1329   \endgroup
1330   \begingroup
1331   \@ifstar
1332   {\ifx\bbl@opt@strings\@nnil
1333     \let\bbl@opt@strings\BabelStringsDefault
1334   \fi
1335     \bbl@startcmds@i}%
1336   \bbl@startcmds@i}
1337 \def\bbl@startcmds@i#1#2{%
1338   \edef\bbl@L{\zap@space#1 \@empty}%
1339   \edef\bbl@G{\zap@space#2 \@empty}%

```

```

1340 \bbl@startcmds@ii}
1341 \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.

```

1342 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1343 \let\SetString@gobbletwo
1344 \let\bbl@stringdef@gobbletwo
1345 \let\AfterBabelCommands@gobble
1346 \ifx\@empty#1%
1347 \def\bbl@sc@label{generic}%
1348 \def\bbl@encstring##1##2{%
1349 \ProvideTextCommandDefault##1{##2}%
1350 \bbl@tglobal##1%
1351 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1352 \let\bbl@sctest\in@true
1353 \else
1354 \let\bbl@sc@charset\space % <- zapped below
1355 \let\bbl@sc@fontenc\space % <- " "
1356 \def\bbl@tempa##1=##2\@nil{%
1357 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1358 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1359 \def\bbl@tempa##1 ##2{% space -> comma
1360 ##1%
1361 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1362 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1363 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1364 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1365 \def\bbl@encstring##1##2{%
1366 \bbl@foreach\bbl@sc@fontenc{%
1367 \bbl@ifunset{T####1}%
1368 }%
1369 {\ProvideTextCommand##1{####1}{##2}%
1370 \bbl@tglobal##1%
1371 \expandafter
1372 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1373 \def\bbl@sctest{%
1374 \bbl@xin@{,\bbl@opt@strings,},{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1375 \fi
1376 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1377 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1378 \let\AfterBabelCommands\bbl@aftercmds
1379 \let\SetString\bbl@setstring
1380 \let\bbl@stringdef\bbl@encstring
1381 \else % ie, strings=value
1382 \bbl@sctest
1383 \ifin@
1384 \let\AfterBabelCommands\bbl@aftercmds
1385 \let\SetString\bbl@setstring
1386 \let\bbl@stringdef\bbl@provstring

```

```

1387 \fi\fi\fi
1388 \bbl@scswitch
1389 \ifx\bbl@G\@empty
1390 \def\SetString##1##2{%
1391     \bbl@error{Missing group for string \string##1}%
1392     {You must assign strings to some category, typically\\%
1393     captions or extras, but you set none}}%
1394 \fi
1395 \ifx\@empty#1%
1396     \bbl@usehooks{defaultcommands}{}%
1397 \else
1398     \@expandtwoargs
1399     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1400 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group``<language>` 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<language>` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has been loaded).

```

1401 \def\bbl@forlang#1#2{%
1402     \bbl@for#1\bbl@L{%
1403         \bbl@xin@{, #1, }{, \BabelLanguages,}%
1404         \ifin@#2\relax\fi}}
1405 \def\bbl@scswitch{%
1406     \bbl@forlang\bbl@tempa{%
1407         \ifx\bbl@G\@empty\else
1408             \ifx\SetString\@gobbletwo\else
1409                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1410                 \bbl@xin@{, \bbl@GL, }{, \bbl@screset,}%
1411                 \ifin@\else
1412                     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1413                     \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1414                 \fi
1415             \fi
1416         \fi}}
1417 \AtEndOfPackage{%
1418     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1419     \let\bbl@scswitch\relax}
1420 \@onlypreamble\EndBabelCommands
1421 \def\EndBabelCommands{%
1422     \bbl@usehooks{stopcommands}{}%
1423     \endgroup
1424     \endgroup
1425     \bbl@scafter}
1426 \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.

```

1427 \def\bbl@setstring#1#2{%
1428   \bbl@forlang\bbl@tempa{%
1429     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1430     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1431     {\global\expandafter % TODO - con \bbl@exp ?
1432       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1433       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
1434     }%
1435   \def\BabelString{#2}%
1436   \bbl@usehooks{stringprocess}{}%
1437   \expandafter\bbl@stringdef
1438   \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`.

```

1439 \ifx\bbl@opt@strings\relax
1440   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1441   \bbl@patchuclc
1442   \let\bbl@encoded\relax
1443   \def\bbl@encoded@uclc#1{%
1444     \@inmathwarn#1%
1445     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1446       \expandafter\ifx\csname ?\string#1\endcsname\relax
1447         \TextSymbolUnavailable#1%
1448       \else
1449         \csname ?\string#1\endcsname
1450       \fi
1451     \else
1452       \csname\cf@encoding\string#1\endcsname
1453     \fi}
1454 \else
1455   \def\bbl@scset#1#2{\def#1{#2}}
1456 \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.

```

1457 <<(*Macros local to BabelCommands)>> ≡
1458 \def\SetStringLoop##1##2{%
1459   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1460   \count@\z@
1461   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1462     \advance\count@\@ne
1463     \toks@\expandafter{\bbl@tempa}%
1464     \bbl@exp{%
1465       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1466       \count@=\the\count@\relax}}}%
1467 <</Macros local to BabelCommands>>

```

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

```

1468 \def\bbl@aftercmds#1{%
1469   \toks@\expandafter{\bbl@scafter#1}%
1470   \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.

```

1471 <<*Macros local to BabelCommands>> ≡
1472 \newcommand\SetCase[3][\%
1473   \bbl@patchuclc
1474   \bbl@forlang\bbl@tempa\%
1475   \expandafter\bbl@encstring
1476   \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1477   \expandafter\bbl@encstring
1478   \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1479   \expandafter\bbl@encstring
1480   \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1481 <</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.

```

1482 <<*Macros local to BabelCommands>> ≡
1483 \newcommand\SetHyphenMap[1]{\%
1484   \bbl@forlang\bbl@tempa\%
1485   \expandafter\bbl@stringdef
1486   \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1487 <</Macros local to BabelCommands>>

```

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

```

1488 \newcommand\BabelLower[2]{\% one to one.
1489   \ifnum\lccode#1=#2\else
1490     \babel@savevariable{\lccode#1}%
1491     \lccode#1=#2\relax
1492   \fi}
1493 \newcommand\BabelLowerMM[4]{\% many-to-many
1494   \@tempcnta=#1\relax
1495   \@tempcntb=#4\relax
1496   \def\bbl@tempa{\%
1497     \ifnum\@tempcnta>#2\else
1498       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1499       \advance\@tempcnta#3\relax
1500       \advance\@tempcntb#3\relax
1501       \expandafter\bbl@tempa
1502     \fi}%
1503   \bbl@tempa}
1504 \newcommand\BabelLowerMO[4]{\% many-to-one
1505   \@tempcnta=#1\relax
1506   \def\bbl@tempa{\%
1507     \ifnum\@tempcnta>#2\else
1508       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1509       \advance\@tempcnta#3
1510       \expandafter\bbl@tempa
1511     \fi}%
1512   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1513 <<*More package options>> ≡
1514 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1515 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1516 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1517 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}

```

```

1518 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1519 <</More package options>>

```

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

```

1520 \AtEndOfPackage{%
1521   \ifx\bb1@opt@hyphenmap\undefined
1522     \bb1@xin{,}{\bb1@language@opts}%
1523     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
1524   \fi}

```

9.10 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.

```

1525 \bb1@trace{Macros related to glyphs}
1526 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z\hbox{#1}%
1527   \dimen\z\ht\z@ \advance\dimen\z@ -\ht\tw@%
1528   \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.

```

1529 \def\save@sf@q#1{\leavevmode
1530   \begingroup
1531   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1532   \endgroup}

```

9.11 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`.

9.11.1 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.

```

1533 \ProvideTextCommand{\quotedblbase}{OT1}{%
1534   \save@sf@q{\set@low@box{\textquotedblright\}}%
1535   \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.

```

1536 \ProvideTextCommandDefault{\quotedblbase}{%
1537   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1538 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1539   \save@sf@q{\set@low@box{\textquoteright\}}%
1540   \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.

```

1541 \ProvideTextCommandDefault{\quotesinglbase}{%
1542   \UseTextSymbol{OT1}{\quotesinglbase}}

```

`\guillemotleft` The guillemet characters are not available in OT1 encoding. They are faked.

```
\guillemotright 1543 \ProvideTextCommand{\guillemotleft}{OT1}{%
1544   \ifmmode
1545     \ll
1546   \else
1547     \save@sf@q{\nobreak
1548       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1549   \fi}
1550 \ProvideTextCommand{\guillemotright}{OT1}{%
1551   \ifmmode
1552     \gg
1553   \else
1554     \save@sf@q{\nobreak
1555       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1556   \fi}
```

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

```
1557 \ProvideTextCommandDefault{\guillemotleft}{%
1558   \UseTextSymbol{OT1}{\guillemotleft}}
1559 \ProvideTextCommandDefault{\guillemotright}{%
1560   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1561 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1562   \ifmmode
1563     <%
1564   \else
1565     \save@sf@q{\nobreak
1566       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1567   \fi}
1568 \ProvideTextCommand{\guilsinglright}{OT1}{%
1569   \ifmmode
1570     >%
1571   \else
1572     \save@sf@q{\nobreak
1573       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1574   \fi}
```

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

```
1575 \ProvideTextCommandDefault{\guilsinglleft}{%
1576   \UseTextSymbol{OT1}{\guilsinglleft}}
1577 \ProvideTextCommandDefault{\guilsinglright}{%
1578   \UseTextSymbol{OT1}{\guilsinglright}}
```

9.11.2 Letters

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

```
1579 \DeclareTextCommand{\ij}{OT1}{%
1580   i\kern-0.02em\bbl@allowhyphens j}
1581 \DeclareTextCommand{\IJ}{OT1}{%
1582   I\kern-0.02em\bbl@allowhyphens J}
1583 \DeclareTextCommand{\ij}{T1}{\char188}
1584 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
1585 \ProvideTextCommandDefault{\ij}{%
1586   \UseTextSymbol{OT1}{\ij}}
1587 \ProvideTextCommandDefault{\IJ}{%
1588   \UseTextSymbol{OT1}{\IJ}}
```

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

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

```
1589 \def\crrtic@{\hrule height0.1ex width0.3em}
1590 \def\crttic@{\hrule height0.1ex width0.33em}
1591 \def\ddj@{%
1592   \setbox0\hbox{d}\dimen@=\ht0
1593   \advance\dimen@1ex
1594   \dimen@.45\dimen@
1595   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1596   \advance\dimen@ii.5ex
1597   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1598 \def\DDJ@{%
1599   \setbox0\hbox{D}\dimen@=.55\ht0
1600   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1601   \advance\dimen@ii.15ex % correction for the dash position
1602   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1603   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1604   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1605 %
1606 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1607 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
1608 \ProvideTextCommandDefault{\dj}{%
1609   \UseTextSymbol{OT1}{\dj}}
1610 \ProvideTextCommandDefault{\DJ}{%
1611   \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.

```
1612 \DeclareTextCommand{\SS}{OT1}{SS}
1613 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.11.3 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 1614 \ProvideTextCommandDefault{\glq}{%
1615   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```

1616 \ProvideTextCommand{\grq}{T1}{%
1617   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
1618 \ProvideTextCommand{\grq}{TU}{%
1619   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1620 \ProvideTextCommand{\grq}{OT1}{%
1621   \save@sf@q{\kern-.0125em
1622     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1623     \kern.07em\relax}}
1624 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 1625 \ProvideTextCommandDefault{\glqq}{%
1626   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1627 \ProvideTextCommand{\grqq}{T1}{%
1628   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1629 \ProvideTextCommand{\grqq}{TU}{%
1630   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1631 \ProvideTextCommand{\grqq}{OT1}{%
1632   \save@sf@q{\kern-.07em
1633     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1634     \kern.07em\relax}}
1635 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

\flq The ‘french’ single guillemets.

```

\frq 1636 \ProvideTextCommandDefault{\flq}{%
1637   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1638 \ProvideTextCommandDefault{\frq}{%
1639   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq The ‘french’ double guillemets.

```

\frqq 1640 \ProvideTextCommandDefault{\flqq}{%
1641   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1642 \ProvideTextCommandDefault{\frqq}{%
1643   \textormath{\guillemotright}{\mbox{\guillemotright}}}

```

9.11.4 Umlauts and tremas

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

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

```

1644 \def\umlauthigh{%
1645   \def\bb1@umlauta##1{\leavevmode\bggroup%
1646     \expandafter\accent\csname\fontencoding dqpos\endcsname
1647     ##1\bb1@allowhyphens\egroup}%
1648   \let\bb1@umlaute\bb1@umlauta}
1649 \def\umlautlow{%
1650   \def\bb1@umlauta{\protect\lower@umlaut}}
1651 \def\umlautelow{%
1652   \def\bb1@umlaute{\protect\lower@umlaut}}
1653 \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.

```
1654 \expandafter\ifx\csname U@D\endcsname\relax
1655   \csname newdimen\endcsname\U@D
1656 \fi
```

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

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

```
1657 \def\lower@umlaut#1{%
1658   \leavevmode\bggroup
1659     \U@D 1ex%
1660     {\setbox\z@\hbox{%
1661       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1662       \dimen@ -.45ex\advance\dimen@\ht\z@
1663       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1664     \expandafter\accent\csname\fontencoding dqpos\endcsname
1665     \fontdimen5\font\U@D #1%
1666   \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).

```
1667 \AtBeginDocument{%
1668   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
1669   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
1670   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
1671   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
1672   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
1673   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
1674   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
1675   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
1676   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
1677   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
1678   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
1679 }
```

Finally, the default is to use English as the main language.

```
1680 \ifx\l@english\undefined
1681   \chardef\l@english\z@
1682 \fi
1683 \main@language{english}
```

9.12 Layout

Work in progress.

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

```

1684 \bbl@trace{Bidi layout}
1685 \providecommand\IfBabelLayout[3]{#3}%
1686 \newcommand\BabelPatchSection[1]{%
1687   \@ifundefined{#1}{}{%
1688     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1689     \@namedef{#1}{%
1690       \ifstar{\bbl@presec@s{#1}}%
1691       {\@dblarg{\bbl@presec@x{#1}}}}}%
1692 \def\bbl@presec@x#1[#2]#3{%
1693   \bbl@exp{%
1694     \\\select@language@x{\bbl@main@language}%
1695     \\\@nameuse{\bbl@sspre@#1}%
1696     \\\@nameuse{\bbl@ss@#1}%
1697     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
1698     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
1699     \\\select@language@x{\language}}}%
1700 \def\bbl@presec@s#1#2{%
1701   \bbl@exp{%
1702     \\\select@language@x{\bbl@main@language}%
1703     \\\@nameuse{\bbl@sspre@#1}%
1704     \\\@nameuse{\bbl@ss@#1}*%
1705     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
1706     \\\select@language@x{\language}}}%
1707 \IfBabelLayout{sectioning}%
1708   {\BabelPatchSection{part}%
1709    \BabelPatchSection{chapter}%
1710    \BabelPatchSection{section}%
1711    \BabelPatchSection{subsection}%
1712    \BabelPatchSection{subsubsection}%
1713    \BabelPatchSection{paragraph}%
1714    \BabelPatchSection{subparagraph}%
1715    \def\babel@toc#1{%
1716      \select@language@x{\bbl@main@language}}}%
1717 \IfBabelLayout{captions}%
1718   {\BabelPatchSection{caption}}}%

```

9.13 Load engine specific macros

```

1719 \bbl@trace{Input engine specific macros}
1720 \ifcase\bbl@engine
1721   \input txtbabel.def
1722 \or
1723   \input luababel.def
1724 \or
1725   \input xebabel.def
1726 \fi

```

9.14 Creating 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.

```

1727 \bbl@trace{Creating languages and reading ini files}
1728 \newcommand\babelprovide[2][{}]{%
1729   \let\bbl@savelangname\language
1730   \edef\bbl@savlocaleid{\the\localeid}%

```

```

1731 % Set name and locale id
1732 \edef\languagename{#2}%
1733 % \global\@namedef{bbl@lcname@#2}{#2}%
1734 \bbl@id@assign
1735 \let\bbl@KVP@captions\@nil
1736 \let\bbl@KVP@import\@nil
1737 \let\bbl@KVP@main\@nil
1738 \let\bbl@KVP@script\@nil
1739 \let\bbl@KVP@language\@nil
1740 \let\bbl@KVP@hyphenrules\@nil % only for provide@new
1741 \let\bbl@KVP@mapfont\@nil
1742 \let\bbl@KVP@maparabic\@nil
1743 \let\bbl@KVP@mapdigits\@nil
1744 \let\bbl@KVP@intraspace\@nil
1745 \let\bbl@KVP@intrapenalty\@nil
1746 \let\bbl@KVP@onchar\@nil
1747 \let\bbl@KVP@alph\@nil
1748 \let\bbl@KVP@Alph\@nil
1749 \bbl@forkv{#1}{% TODO - error handling
1750   \in@{/}{##1}%
1751   \ifin@
1752     \bbl@renewinikey##1\@{##2}%
1753   \else
1754     \bbl@csarg\def{KVP@##1}{##2}%
1755   \fi}%
1756 % == import, captions ==
1757 \ifx\bbl@KVP@import\@nil\else
1758   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1759   {\beginingroup
1760     \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1761     \InputIfFileExists{babel-#2.tex}{}{}%
1762     \endgroup}%
1763   {}%
1764 \fi
1765 \ifx\bbl@KVP@captions\@nil
1766   \let\bbl@KVP@captions\bbl@KVP@import
1767 \fi
1768 % Load ini
1769 \bbl@ifunset{date#2}%
1770   {\bbl@provide@new{#2}}%
1771   {\bbl@ifblank{#1}%
1772     {\bbl@error
1773       {If you want to modify `#2' you must tell how in\\%
1774         the optional argument. See the manual for the\\%
1775         available options.}%
1776       {Use this macro as documented}}%
1777     {\bbl@provide@renew{#2}}}%
1778 % Post tasks
1779 \bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}%
1780 \bbl@ifunset{bbl@ensure@\languagename}%
1781   {\bbl@exp{
1782     \\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1783       \\foreignlanguage{\languagename}%
1784       {###1}}}%
1785   {}%
1786 % At this point all parameters are defined if 'import'. Now we
1787 % execute some code depending on them. But what about if nothing was
1788 % imported? We just load the very basic parameters: ids and a few
1789 % more.

```



```

1790 \bbl@ifunset{bbl@lname@#2}%
1791 {\def\BabelBeforeIni##1##2{%
1792   \begingroup
1793     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\;=12 %
1794     \let\bbl@ini@captions@aux\@gobbletwo
1795     \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6{%
1796       \bbl@read@ini{##1}{basic data}%
1797       \bbl@exportkey{chrng}{characters.ranges}{}%
1798       \bbl@exportkey{dgnat}{numbers.digits.native}{}%
1799       \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
1800       \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1801       \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
1802       \bbl@exportkey{intsp}{typography.intraspace}{}%
1803       \endinput
1804     \endgroup}%
1805     boxed, to avoid extra spaces:
1806     {\setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
1807   }%
1808 % -
1809 % == script, language ==
1810 % Override the values from ini or defines them
1811 \ifx\bbl@KVP@script\@nil\else
1812   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1813 \fi
1814 \ifx\bbl@KVP@language\@nil\else
1815   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1816 \fi
1817 % == onchar ==
1818 \ifx\bbl@KVP@onchar\@nil\else
1819   \bbl@luaahyphenate
1820   \directlua{
1821     if Babel.locale_mapped == nil then
1822       Babel.locale_mapped = true
1823       Babel.linebreaking.add_before(Babel.locale_map)
1824       Babel.loc_to_scr = {}
1825       Babel.chr_to_loc = Babel.chr_to_loc or {}
1826     end}%
1827   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
1828 \ifin@
1829   \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
1830     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
1831   \fi
1832   \bbl@exp{\bbl@add\bbl@starthyphens
1833     {\bbl@patterns@lua{\language}}}%
1834   % TODO - error/warning if no script
1835   \directlua{
1836     if Babel.script_blocks['\bbl@cs{sbcpr}\language'] then
1837       Babel.loc_to_scr[\the\localeid] =
1838         Babel.script_blocks['\bbl@cs{sbcpr}\language']
1839       Babel.locale_props[\the\localeid].lc = \the\localeid\space
1840       Babel.locale_props[\the\localeid].lg = \the@nameuse{l@\language}\space
1841     end
1842   }%
1843 \fi
1844 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
1845 \ifin@
1846   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{%
1847     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{%
1848       \directlua{
1849         if Babel.script_blocks['\bbl@cs{sbcpr}\language'] then

```

```

1849         Babel.loc_to_scr[\the\localeid] =
1850         Babel.script_blocks['\bbl@cs{sbcpr@languagename}']
1851     end}%
1852 \ifx\bbl@mapselect@undefined
1853     \AtBeginDocument{%
1854         \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1855         {\selectfont}}%
1856     \def\bbl@mapselect{%
1857         \let\bbl@mapselect\relax
1858         \edef\bbl@prefontid{\fontid\font}}%
1859     \def\bbl@mapdir##1{%
1860         {\def\languagename{##1}%
1861         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
1862         \bbl@switchfont
1863         \directlua{
1864             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
1865             [\bbl@prefontid'] = \fontid\font\space}}}%
1866     \fi
1867     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
1868 \fi
1869 % TODO - catch non-valid values
1870 \fi
1871 % == mapfont ==
1872 % For bidi texts, to switch the font based on direction
1873 \ifx\bbl@KVP@mapfont\@nil\else
1874     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{%
1875         {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
1876             mapfont. Use 'direction'.%
1877             {See the manual for details.}}}%
1878     \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys\languagename}}{%
1879     \bbl@ifunset{\bbl@wdir\languagename}{\bbl@provide@dirs\languagename}}{%
1880 \ifx\bbl@mapselect@undefined
1881     \AtBeginDocument{%
1882         \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1883         {\selectfont}}%
1884     \def\bbl@mapselect{%
1885         \let\bbl@mapselect\relax
1886         \edef\bbl@prefontid{\fontid\font}}%
1887     \def\bbl@mapdir##1{%
1888         {\def\languagename{##1}%
1889         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
1890         \bbl@switchfont
1891         \directlua{Babel.fontmap
1892             [\the\csname bbl@wdir@##1\endcsname]%
1893             [\bbl@prefontid]=\fontid\font}}}%
1894     \fi
1895     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
1896 \fi
1897 % == intraspace, intrapenalty ==
1898 % For CJK, East Asian, Southeast Asian, if interspace in ini
1899 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
1900     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
1901 \fi
1902 \bbl@provide@intraspace
1903 % == hyphenate.other ==
1904 \bbl@ifunset{\bbl@hyoth\languagename}{}%
1905     {\bbl@csarg\bbl@replace{hyoth\languagename}{ }{,}}%
1906     \bbl@startcommands*\languagename}%
1907     \bbl@csarg\bbl@foreach{hyoth\languagename}{%

```

```

1908         \ifcase\bbl@engine
1909             \ifnum##1<257
1910                 \SetHyphenMap{\BabelLower{##1}{##1}}%
1911             \fi
1912         \else
1913             \SetHyphenMap{\BabelLower{##1}{##1}}%
1914         \fi}%
1915     \bbl@endcommands}
1916 % == maparabic ==
1917 % Native digits, if provided in ini (TeX level, xe and lua)
1918 \ifcase\bbl@engine\else
1919     \bbl@ifunset{\bbl@dgnat@\languagename}{}%
1920     {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
1921         \expandafter\expandafter\expandafter
1922         \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
1923         \ifx\bbl@KVP@maparabic\@nil\else
1924             \ifx\bbl@latinarabic\@undefined
1925                 \expandafter\let\expandafter\@arabic
1926                 \csname bbl@counter@\languagename\endcsname
1927             \else % ie, if layout=counters, which redefines \@arabic
1928                 \expandafter\let\expandafter\bbl@latinarabic
1929                 \csname bbl@counter@\languagename\endcsname
1930             \fi
1931         \fi
1932     \fi}%
1933 \fi
1934 % == mapdigits ==
1935 % Native digits (lua level).
1936 \ifodd\bbl@engine
1937     \ifx\bbl@KVP@mapdigits\@nil\else
1938         \bbl@ifunset{\bbl@dgnat@\languagename}{}%
1939         {\RequirePackage{luatexbase}%
1940         \bbl@activate@preotf
1941         \directlua{
1942             Babel = Babel or {} %%% -> presets in luababel
1943             Babel.digits_mapped = true
1944             Babel.digits = Babel.digits or {}
1945             Babel.digits[\the\localeid] =
1946             table.pack(string.utfvalue('\bbl@cs{dgnat@\languagename}'))
1947             if not Babel.numbers then
1948                 function Babel.numbers(head)
1949                     local LOCALE = luatexbase.registernumber'bbl@attr@locale'
1950                     local GLYPH = node.id'glyph'
1951                     local inmath = false
1952                     for item in node.traverse(head) do
1953                         if not inmath and item.id == GLYPH then
1954                             local temp = node.get_attribute(item, LOCALE)
1955                             if Babel.digits[temp] then
1956                                 local chr = item.char
1957                                 if chr > 47 and chr < 58 then
1958                                     item.char = Babel.digits[temp][chr-47]
1959                                 end
1960                             end
1961                         elseif item.id == node.id'math' then
1962                             inmath = (item.subtype == 0)
1963                         end
1964                     end
1965                     return head
1966                 end

```

```

1967         end
1968     }}%
1969 \fi
1970 \fi
1971 % == alph, Alph ==
1972 % What if extras<lang> contains a \babel@save\@alph? It won't be
1973 % restored correctly when exiting the language, so we ignore
1974 % this change with the \bbl@alph@saved trick.
1975 \ifx\bbl@KVP@alph\@nil\else
1976     \toks@\expandafter\expandafter\expandafter{%
1977         \csname extras\language\endcsname}%
1978     \bbl@exp{%
1979         \def<extras\language>{%
1980             \let\\bbl@alph@saved\\@alph
1981             \the\toks@
1982             \let\\@alph\\bbl@alph@saved
1983             \\babel@save\\@alph
1984             \let\\@alph<bbl@cntr@bbl@KVP@alph @\language>}}%
1985 \fi
1986 \ifx\bbl@KVP@Alph\@nil\else
1987     \toks@\expandafter\expandafter\expandafter{%
1988         \csname extras\language\endcsname}%
1989     \bbl@exp{%
1990         \def<extras\language>{%
1991             \let\\bbl@Alph@saved\\@Alph
1992             \the\toks@
1993             \let\\@Alph\\bbl@Alph@saved
1994             \\babel@save\\@Alph
1995             \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language>}}%
1996 \fi
1997 % == require.babel in ini ==
1998 % To load or reload the babel-*.tex, if require.babel in ini
1999 \bbl@ifunset{bbl@rqtex@\language}{}%
2000     {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
2001         \let\BabelBeforeIni@gobbletwo
2002         \chardef\atcatcode=\catcode\@
2003         \catcode\@=11\relax
2004         \InputIfFileExists{babel-\bbl@cs{rqtex@\language}.tex}{}%
2005         \catcode\@=\atcatcode
2006         \let\atcatcode\relax
2007     \fi}%
2008 % == main ==
2009 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
2010     \let\language\bbl@savelangname
2011     \chardef\localeid\bbl@savelocaleid\relax
2012 \fi}

```

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 .

```

2013 \def\bbl@setdigits#1#2#3#4#5{%
2014     \bbl@exp{%
2015         \def<\language digits>####1{% ie, \langdigits
2016             <bbl@digits@\language>####1\\@nil}%
2017         \def<\language counter>####1{% ie, \langcounter
2018             \\expandafter<bbl@counter@\language>%
2019             \\csname c@####1\endcsname}%
2020         \def<bbl@counter@\language>####1{% ie, \bbl@counter@lang
2021             \\expandafter<bbl@digits@\language>%
2022             \\number####1\\@nil}}%

```

```

2023 \def\bbl@tempa##1##2##3##4##5{%
2024   \bbl@exp{%    Wow, quite a lot of hashes! :-(
2025     \def\<bbl@digits@\language\>#####1{%
2026       \ifx#####1\\\@nil          % ie, \bbl@digits@lang
2027       \else
2028         \ifx0#####1#1%
2029         \else\ifx1#####1#2%
2030         \else\ifx2#####1#3%
2031         \else\ifx3#####1#4%
2032         \else\ifx4#####1#5%
2033         \else\ifx5#####1##1%
2034         \else\ifx6#####1##2%
2035         \else\ifx7#####1##3%
2036         \else\ifx8#####1##4%
2037         \else\ifx9#####1##5%
2038         \else#####1%
2039         \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
2040         \expandafter\<bbl@digits@\language\>%
2041       \fi}}}%
2042 \bbl@tempa}

```

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

```

2043 \def\bbl@provide@new#1{%
2044   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2045   \@namedef{extras#1}{}%
2046   \@namedef{noextras#1}{}%
2047   \bbl@startcommands*{#1}{captions}%
2048   \ifx\bbl@KVP@captions\@nil %    and also if import, implicit
2049     \def\bbl@tempb##1{%          elt for \bbl@captionslist
2050       \ifx##1\@empty\else
2051         \bbl@exp{%
2052           \SetString\##1{%
2053             \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2054           \expandafter\bbl@tempb
2055         \fi}%
2056       \expandafter\bbl@tempb\bbl@captionslist\@empty
2057     \else
2058       \bbl@read@ini{\bbl@KVP@captions}{data}% Here all letters cat = 11
2059       \bbl@after@ini
2060       \bbl@savestrings
2061     \fi
2062   \StartBabelCommands*{#1}{date}%
2063   \ifx\bbl@KVP@import\@nil
2064     \bbl@exp{%
2065       \SetString\today{\bbl@nocaption{today}{#1today}}}%
2066   \else
2067     \bbl@savetoday
2068     \bbl@savestate
2069   \fi
2070   \bbl@endcommands
2071   \bbl@exp{%
2072     \def\<#1hyphenmins>{%
2073       {\bbl@ifunset{\bbl@ifthm@#1}{2}{\@nameuse{\bbl@ifthm@#1}}}%
2074       {\bbl@ifunset{\bbl@rgthm@#1}{3}{\@nameuse{\bbl@rgthm@#1}}}}}%
2075   \bbl@provide@hyphens{#1}%
2076   \ifx\bbl@KVP@main\@nil\else
2077     \expandafter\main@language\expandafter{#1}%
2078   \fi}
2079 \def\bbl@provide@renew#1{%

```

```

2080 \ifx\bb1@KVP@captions\@nil\else
2081   \StartBabelCommands*{#1}{captions}%
2082   \bb1@read@ini{\bb1@KVP@captions}{data}% Here all letters cat = 11
2083   \bb1@after@ini
2084   \bb1@savestrings
2085   \EndBabelCommands
2086 \fi
2087 \ifx\bb1@KVP@import\@nil\else
2088   \StartBabelCommands*{#1}{date}%
2089   \bb1@savetoday
2090   \bb1@savedate
2091   \EndBabelCommands
2092 \fi
2093 % == hyphenrules ==
2094 \bb1@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2095 \def\bb1@provide@hyphens#1{%
2096   \let\bb1@tempa\relax
2097   \ifx\bb1@KVP@hyphenrules\@nil\else
2098     \bb1@replace\bb1@KVP@hyphenrules{ }{,}%
2099     \bb1@foreach\bb1@KVP@hyphenrules{%
2100       \ifx\bb1@tempa\relax % if not yet found
2101         \bb1@ifsamestring{##1}{+}%
2102         {\bb1@exp{\addlanguage\<l@##1>}}}%
2103       {}%
2104       \bb1@ifunset{l@##1}%
2105       {}%
2106       {\bb1@exp{\let\bb1@tempa\<l@##1>}}}%
2107     \fi}%
2108 \fi
2109 \ifx\bb1@tempa\relax % if no opt or no language in opt found
2110   \ifx\bb1@KVP@import\@nil\else % if importing
2111     \bb1@exp{%
2112       \bb1@ifblank{\@nameuse{bb1@hyphr@#1}}%
2113       {}%
2114       {\let\bb1@tempa\<l@\@nameuse{bb1@hyphr@\language}\>}}%
2115     \fi
2116 \fi
2117 \bb1@ifunset{bb1@tempa}% ie, relax or undefined
2118 {\bb1@ifunset{l@#1}% no hyphenrules found - fallback
2119   {\bb1@exp{\adddialect\<l@#1>\language}}%
2120   {}% so, l@<lang> is ok - nothing to do
2121   {\bb1@exp{\adddialect\<l@#1>\bb1@tempa}}% found in opt list or ini
2122 \bb1@ifunset{bb1@prehc@\language}%
2123 {}% TODO - XeTeX, based on \babelfont and HyphenChar?
2124 {\ifodd\bb1@engine\bb1@exp{%
2125   \bb1@ifblank{\@nameuse{bb1@prehc@#1}}%
2126   {}%
2127   {\AddBabelHook[\language]{babel-prehc-\language}{patterns}%
2128     {\prehyphenchar=\@nameuse{bb1@prehc@\language}\relax}}}%
2129 \fi}}

```

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

```

2130 \ifx\bb1@readstream\@undefined
2131   \csname newread\endcsname\bb1@readstream
2132 \fi
2133 \def\bb1@read@ini#1#2{%

```

```

2134 \global\@namedef{bbl@lini@\language}\language}\{#1}%
2135 \openin\bbl@readstream=babel-#1.ini
2136 \ifeof\bbl@readstream
2137   \bbl@error
2138     {There is no ini file for the requested language\%
2139      (#1). Perhaps you misspelled it or your installation\%
2140      is not complete.}%
2141     {Fix the name or reinstall babel.}%
2142 \else
2143   \let\bbl@section\@empty
2144   \let\bbl@savestrings\@empty
2145   \let\bbl@savetoday\@empty
2146   \let\bbl@savestate\@empty
2147   \def\bbl@inipreread##1=##2\@@{%
2148     \bbl@trim\def\bbl@tempa{##1}% Redundant below !!
2149     % Move trims here ??
2150     \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
2151     {\expandafter\bbl@inireader\bbl@tempa=##2\@@}%
2152     {}}%
2153   \let\bbl@inireader\bbl@iniskip
2154   \bbl@info{Importing #2 for \language\%
2155     from babel-#1.ini. Reported}%
2156   \loop
2157   \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2158     \newlinechar\m@ne
2159     \read\bbl@readstream to \bbl@line
2160     \newlinechar\^^M
2161     \ifx\bbl@line\@empty\else
2162       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2163     \fi
2164   \repeat
2165   \bbl@foreach\bbl@renewlist{%
2166     \bbl@ifunset{bbl@renew@##1}{\bbl@inisec[##1]\@@}%
2167   \global\let\bbl@renewlist\@empty
2168   % Ends last section. See \bbl@inisec
2169   \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
2170   \@nameuse{bbl@renew@\bbl@section}%
2171   \global\bbl@csarg\let{renew@\bbl@section}\relax
2172   \@nameuse{bbl@secpost@\bbl@section}%
2173 %   \bbl@csarg\bbl@tglobal{inikeys@\language}%
2174 \fi}
2175 \def\bbl@iniline#1\bbl@iniline{%
2176   \@ifnextchar[\bbl@inisec{\@ifnextchar\bbl@iniskip\bbl@inipreread}#1\@@}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

2177 \def\bbl@iniskip#1\@@{%      if starts with ;
2178 \def\bbl@inisec[#1]#2\@@{%   if starts with opening bracket
2179   \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
2180   \@nameuse{bbl@renew@\bbl@section}%
2181   \global\bbl@csarg\let{renew@\bbl@section}\relax
2182   \@nameuse{bbl@secpost@\bbl@section}% ends previous section
2183   \def\bbl@section{#1}%      starts current section
2184   \def\bbl@elt##1##2{%
2185     \@namedef{bbl@KVP@#1/##1}{}}%
2186   \@nameuse{bbl@renew@#1}%

```

```

2187 \nameuse{bbl@secp@#1}% pre-section 'hook'
2188 \bbl@ifunset{bbl@inikv@#1}%
2189 {\let\bbl@inireader\bbl@iniskip}%
2190 {\bbl@exp{\let\\bbl@inireader<bbl@inikv@#1>}}
2191 \let\bbl@renewlist\@empty
2192 \def\bbl@renewinikey#1/#2\@#3{%
2193 \bbl@ifunset{bbl@renew@#1}%
2194 {\bbl@add@list\bbl@renewlist{#1}}%
2195 }%
2196 \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}

```

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

```

2197 \def\bbl@inikv#1=#2\@#3% key=value
2198 \bbl@trim@def\bbl@tempa{#1}%
2199 \bbl@trim\toks@{#2}%
2200 \bbl@csarg\edef{@kv@\bbl@section.\bbl@tempa}{\the\toks@}

```

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

```

2201 \def\bbl@exportkey#1#2#3{%
2202 \bbl@ifunset{bbl@kv@#2}%
2203 {\bbl@csarg\gdef{#1@\language}\{#3}}%
2204 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2205 \bbl@csarg\gdef{#1@\language}\{#3}}%
2206 \else
2207 \bbl@exp{\global\let<bbl@#1@\language>\<bbl@kv@#2>}%
2208 \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@secp@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

2209 \def\bbl@iniwarning#1{%
2210 \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2211 {\bbl@warning%
2212 From babel-\nameuse{bbl@lini@\language}.ini:\\
2213 \nameuse{bbl@kv@identification.warning#1}\\
2214 Reported }}
2215 \let\bbl@inikv@identification\bbl@inikv
2216 \def\bbl@secp@identification{%
2217 \bbl@iniwarning}%
2218 \ifcase\bbl@engine
2219 \bbl@iniwarning{.pdflatex}%
2220 \or
2221 \bbl@iniwarning{.lualatex}%
2222 \or
2223 \bbl@iniwarning{.xelatex}%
2224 \fi%
2225 \bbl@exportkey{elname}{identification.name.english}{}%
2226 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2227 {\csname bbl@elname@\language\endcsname}}%
2228 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
2229 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2230 \bbl@exportkey{esname}{identification.script.name}{}%
2231 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2232 {\csname bbl@esname@\language\endcsname}}%
2233 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2234 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
2235 \let\bbl@inikv@typography\bbl@inikv

```



```

2236 \let\bbl@inikv@characters\bbl@inikv
2237 \let\bbl@inikv@numbers\bbl@inikv
2238 \def\bbl@inikv@counters#1=#2\@@{%
2239   \def\bbl@tempc{#1}%
2240   \bbl@trim@def{\bbl@tempb*}{#2}%
2241   \in@{.1$}{#1$}%
2242   \ifin@
2243     \bbl@replace\bbl@tempc{.1}{}%
2244     \bbl@csarg\xdef{cntr@\bbl@tempc @\language name}{%
2245       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2246   \fi
2247   \in@{.F.}{#1}%
2248   \ifin@else\in@{.S.}{#1}\fi
2249   \ifin@
2250     \bbl@csarg\xdef{cntr@#1@\language name}{\bbl@tempb*}%
2251   \else
2252     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2253     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \ \
2254     \bbl@csarg{\global\expandafter\let}{cntr@#1@\language name}\bbl@tempa
2255   \fi}
2256 \def\bbl@after@ini{%
2257   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2258   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2259   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2260   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2261   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2262   \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
2263   \bbl@exportkey{intsp}{typography.intraspace}{}%
2264   \bbl@exportkey{jstfy}{typography.justify}{w}%
2265   \bbl@exportkey{chrng}{characters.ranges}{}%
2266   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2267   \bbl@exportkey{rqtex}{identification.require.babel}{}%
2268   \bbl@tglobal\bbl@savetoday
2269   \bbl@tglobal\bbl@savestate}

```

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

```

2270 \ifcase\bbl@engine
2271   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
2272     \bbl@ini@captions@aux{#1}{#2}}
2273 \else
2274   \def\bbl@inikv@captions#1=#2\@@{%
2275     \bbl@ini@captions@aux{#1}{#2}}
2276 \fi

```

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

```

2277 \def\bbl@ini@captions@aux#1#2{%
2278   \bbl@trim@def\bbl@tempa{#1}%
2279   \bbl@ifblank{#2}%
2280   {\bbl@exp{%
2281     \toks@{\bbl@nocaption{\bbl@tempa}{\language name\bbl@tempa name}}}%
2282   {\bbl@trim\toks@{#2}}}%
2283   \bbl@exp{%
2284     \bbl@add\bbl@savestrings{%
2285       \SetString\<\bbl@tempa name>{\the\toks@}}}

```

But dates are more complex. The full date format is stores in date.gregorian, so we must read it in non-Unicode engines, too (saved months are just discarded when the LICR section is reached).

TODO. Remove copypaste pattern.

```

2286 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{% for defaults
2287 \bbl@inidate#1...\relax{#2}{}}
2288 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
2289 \bbl@inidate#1...\relax{#2}{islamic}}
2290 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
2291 \bbl@inidate#1...\relax{#2}{hebrew}}
2292 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
2293 \bbl@inidate#1...\relax{#2}{persian}}
2294 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
2295 \bbl@inidate#1...\relax{#2}{indian}}
2296 \ifcase\bbl@engine
2297 \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{% override
2298 \bbl@inidate#1...\relax{#2}{}}
2299 \bbl@csarg\def{secpre@date.gregorian.licr}{% discard uni
2300 \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
2301 \fi
2302 % eg: 1=months, 2=wide, 3=1, 4=dummy
2303 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2304 \bbl@trim@def\bbl@tempa{#1.#2}%
2305 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savestate
2306 {\bbl@trim@def\bbl@tempa{#3}%
2307 \bbl@trim\toks@{#5}%
2308 \bbl@exp{%
2309 \\\bbl@add\\\bbl@savestate{%
2310 \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}%
2311 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
2312 {\bbl@trim@def\bbl@toreplace{#5}%
2313 \bbl@TG@@date
2314 \global\bbl@csarg\let{date@\language name}\bbl@toreplace
2315 \bbl@exp{%
2316 \gdef\<\language name date>{\\\protect\<\language name date >}%
2317 \gdef\<\language name date >####1####2####3{%
2318 \\\bbl@usedategroupttrue
2319 \<\bbl@ensure@\language name>{%
2320 \<\bbl@date@\language name>{####1}{####2}{####3}}}%
2321 \\\bbl@add\\\bbl@savetoday{%
2322 \\\SetString\\today{%
2323 \<\language name date>{\\\the\year}{\\the\month}{\\the\day}}}}}%
2324 {}}}
```

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.

```

2325 \let\bbl@calendar\empty
2326 \newcommand\BabelDateSpace{\nobreakspace}
2327 \newcommand\BabelDateDot{.\@}
2328 \newcommand\BabelDated[1]{\number#1}
2329 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
2330 \newcommand\BabelDateM[1]{\number#1}
2331 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2332 \newcommand\BabelDateMMMM[1]{%
2333 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
2334 \newcommand\BabelDatey[1]{\number#1}%
2335 \newcommand\BabelDateyy[1]{%
2336 \ifnum#1<10 0\number#1 %
2337 \else\ifnum#1<100 \number#1 %
2338 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
```

```

2339 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2340 \else
2341   \bbl@error
2342   {Currently two-digit years are restricted to the\
2343    range 0-9999.}%
2344   {There is little you can do. Sorry.}%
2345 \fi\fi\fi\fi}}
2346 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2347 \def\bbl@replace@finish@iii#1{%
2348   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
2349 \def\bbl@TG@date{%
2350   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
2351   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
2352   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
2353   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
2354   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
2355   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
2356   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
2357   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
2358   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
2359   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
2360 % Note after \bbl@replace \toks@ contains the resulting string.
2361 % TODO - Using this implicit behavior doesn't seem a good idea.
2362   \bbl@replace@finish@iii\bbl@toreplace}

```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```

2363 \def\bbl@provide@lsys#1{%
2364   \bbl@ifunset{bbl@lname@#1}%
2365   {\bbl@ini@basic{#1}}%
2366   }%
2367   \bbl@csarg\let{lsys@#1}\@empty
2368   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
2369   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
2370   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2371   \bbl@ifunset{bbl@lname@#1}{%
2372     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2373     \bbl@csarg\bbl@toGLOBAL{lsys@#1}}

```

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).

```

2374 \def\bbl@ini@basic#1{%
2375   \def\BabelBeforeIni##1##2{%
2376     \begingroup
2377       \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
2378       \catcode`\[12 \catcode`\]=12 \catcode`\==12 \catcode`\;12 %
2379       \bbl@read@ini{##1}{font and identification data}%
2380       \endinput % babel- .tex may contain onlypreamble's
2381       \endgroup}% boxed, to avoid extra spaces:
2382   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}}

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

2383 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={
2384   \ifx\#1% % \ before, in case #1 is multiletter
2385     \bbl@exp{%
2386       \def\#1\bbl@tempa####1{%

```

```

2387 \<ifcase>###1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
2388 \else
2389 \toks@\expandafter{\the\toks@\or #1}%
2390 \expandafter\bb1@buildifcase
2391 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as a special case. for a fixed form (see babel-he.ini, for example).

```

2392 \newcommand\localenumberal[2]{\@nameuse{bb1@cntr@#1@\language}\{#2}}
2393 \def\bb1@localecntr#1#2{\localenumberal{#2}{#1}}
2394 \newcommand\localecounter[2]{%
2395 \expandafter\bb1@localecntr\csname c@#2\endcsname{#1}}
2396 \def\bb1@alphnumberal#1#2{%
2397 \expandafter\bb1@alphnumberal@i\number#2 76543210@@{#1}}
2398 \def\bb1@alphnumberal@i#1#2#3#4#5#6#7#8\@@#9{%
2399 \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
2400 \bb1@alphnumberal@ii{#9}000000#1\or
2401 \bb1@alphnumberal@ii{#9}000000#1#2\or
2402 \bb1@alphnumberal@ii{#9}0000#1#2#3\or
2403 \bb1@alphnumberal@ii{#9}000#1#2#3#4\else
2404 \bb1@alphnum@invalid{>9999}%
2405 \fi}
2406 \def\bb1@alphnumberal@ii#1#2#3#4#5#6#7#8{%
2407 \bb1@ifunset{bb1@cntr@#1.F.\number#5#6#7#8@\language}%
2408 {\@nameuse{bb1@cntr@#1.4@\language}\{#5}%
2409 \@nameuse{bb1@cntr@#1.3@\language}\{#6}%
2410 \@nameuse{bb1@cntr@#1.2@\language}\{#7}%
2411 \@nameuse{bb1@cntr@#1.1@\language}\{#8}%
2412 \ifnum#6#7#8>\z@ % An ad hoc rule for Greek. Ugly. To be fixed.
2413 \bb1@ifunset{bb1@cntr@#1.S.321@\language}\{}}%
2414 {\@nameuse{bb1@cntr@#1.S.321@\language}\{}}%
2415 \fi}%
2416 {\@nameuse{bb1@cntr@#1.F.\number#5#6#7#8@\language}}}%
2417 \def\bb1@alphnum@invalid#1{%
2418 \bb1@error{Alphabetic numeral too large (#1)}%
2419 {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.

```

2420 \newcommand\localeinfo[1]{%
2421 \bb1@ifunset{bb1\csname bb1@info@#1\endcsname @\language}%
2422 {\bb1@error{I've found no info for the current locale.\%
2423 The corresponding ini file has not been loaded\%
2424 Perhaps it doesn't exist}%
2425 {See the manual for details.}}%
2426 {\@nameuse{bb1\csname bb1@info@#1\endcsname @\language}}}%
2427 % \namedef{bb1@info@name.locale}\{lname}
2428 \namedef{bb1@info@tag.ini}\{lini}
2429 \namedef{bb1@info@name.english}\{elname}
2430 \namedef{bb1@info@name.opentype}\{lname}
2431 \namedef{bb1@info@tag.bcp47}\{lbcp}
2432 \namedef{bb1@info@tag.opentype}\{lotf}
2433 \namedef{bb1@info@script.name}\{esname}
2434 \namedef{bb1@info@script.name.opentype}\{sname}

```

```

2435 \@namedef{bbl@info@script.tag.bcp47}{sbc}
2436 \@namedef{bbl@info@script.tag.opentype}{sotf}
2437 \let\bbl@ensureinfo\@gobble
2438 \newcommand\BabelEnsureInfo{%
2439   \def\bbl@ensureinfo##1{%
2440     \ifx\InputIfFileExists\undefined\else % not in plain
2441       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}%
2442     \fi}}

```

10 Adjusting the Babel bahavior

A generic high level inteface is provided to adjust some global and general settings.

```

2443 \newcommand\babeladjust[1]{% TODO. Error handling.
2444   \bbl@forkv{#1}{\@nameuse{bbl@ADJ@##1@##2}}
2445 %
2446 \def\bbl@adjust@lua#1#2{%
2447   \ifvmode
2448     \ifnum\currentgrouplevel=\z@
2449       \directlua{ Babel.#2 }%
2450       \expandafter\expandafter\expandafter\@gobble
2451     \fi
2452   \fi
2453   {\bbl@error % The error is gobbled if everything went ok.
2454     {Currently, #1 related features can be adjusted only\\%
2455       in the main vertical list.}%
2456     {Maybe things change in the future, but this is what it is.}}}
2457 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
2458   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
2459 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
2460   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
2461 \@namedef{bbl@ADJ@bidi.text@on}{%
2462   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
2463 \@namedef{bbl@ADJ@bidi.text@off}{%
2464   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
2465 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
2466   \bbl@adjust@lua{bidi}{digits_mapped=true}}
2467 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
2468   \bbl@adjust@lua{bidi}{digits_mapped=false}}
2469 %
2470 \@namedef{bbl@ADJ@linebreak.sea@on}{%
2471   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
2472 \@namedef{bbl@ADJ@linebreak.sea@off}{%
2473   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
2474 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
2475   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
2476 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
2477   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
2478 %
2479 \def\bbl@adjust@layout#1{%
2480   \ifvmode
2481     #1%
2482     \expandafter\@gobble
2483   \fi
2484   {\bbl@error % The error is gobbled if everything went ok.
2485     {Currently, layout related features can be adjusted only\\%
2486       in vertical mode.}%
2487     {Maybe things change in the future, but this is what it is.}}}

```

```

2488 \@namedef{bbl@ADJ@layout.tabular@on}{%
2489   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
2490 \@namedef{bbl@ADJ@layout.tabular@off}{%
2491   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
2492 \@namedef{bbl@ADJ@layout.lists@on}{%
2493   \bbl@adjust@layout{\let\list\bbl@NL@list}}
2494 \@namedef{bbl@ADJ@layout.lists@on}{%
2495   \bbl@adjust@layout{\let\list\bbl@OL@list}}
2496 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
2497   \bbl@activateposthyphen}

```

11 The kernel of Babel (babel.def for \LaTeX only)

11.1 The redefinition of the style commands

The rest of the code in this file can only be processed by \LaTeX , so we check the current format. If it is plain \TeX , processing should stop here. But, because of the need to limit the scope of the definition of `\format`, a macro that is used locally in the following `\if` statement, this comparison is done inside a group. To prevent \TeX from complaining about an unclosed group, the processing of the command `\endinput` is deferred until after the group is closed. This is accomplished by the command `\aftergroup`.

```

2498 {\def\format{lplain}
2499 \ifx\fmtname\format
2500 \else
2501   \def\format{LaTeX2e}
2502   \ifx\fmtname\format
2503   \else
2504     \aftergroup\endinput
2505   \fi
2506 \fi}

```

11.2 Cross referencing macros

The \LaTeX book states:

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

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

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

The only way to accomplish this in most cases is to use the trick described in the \TeX book [4] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, ‘`\meaning\A`’ with `\A` defined as ‘`\def\A#1{\B}`’ expands to the characters ‘`macro:#1->\B`’ with all category codes set to ‘other’ or ‘space’.

`\newlabel` The macro `\label` writes a line with a `\newlabel` command into the `.aux` file to define labels.

```

2507 %\bbl@redefine\newlabel#1#2{%
2508 %  \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}

```

`\@newl@bel` We need to change the definition of the \LaTeX -internal macro `\@newl@bel`. This is needed because we need to make sure that shorthand characters expand to their non-active version.

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

```
2509 <<*More package options>> ≡
2510 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2511 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2512 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2513 <</More package options>>
```

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.

```
2514 \bbl@trace{Cross referencing macros}
2515 \ifx\bbl@opt@safe\@empty\else
2516   \def\@newl@bel#1#2#3{%
2517     {\@safe@activestrue
2518       \bbl@ifunset{#1@#2}%
2519       \relax
2520       {\gdef\@multiplelabels{%
2521         \latex@warning@no@line{There were multiply-defined labels}}%
2522         \latex@warning@no@line{Label `#2' multiply defined}}%
2523       \global\@namedef{#1@#2}{#3}}}
```

`\@testdef` An internal \LaTeX macro used to test if the labels that have been written on the `.aux` file have changed. It is called by the `\enddocument` macro. This macro needs to be completely rewritten, using `\meaning`. The reason for this is that in some cases the expansion of `\#1@#2` contains the same characters as the `#3`; but the character codes differ. Therefore \LaTeX keeps reporting that the labels may have changed.

```
2524 \CheckCommand*\@testdef[3]{%
2525   \def\reserved@a{#3}%
2526   \expandafter\ifx\cname#1@#2\endcsname\reserved@a
2527   \else
2528     \@tempswatrue
2529   \fi}
```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’.

```
2530 \def\@testdef#1#2#3{%
2531   \@safe@activestrue
```

Then we use `\bbl@tempa` as an ‘alias’ for the macro that contains the label which is being checked.

```
2532   \expandafter\let\expandafter\bbl@tempa\cname #1@#2\endcsname
```

Then we define `\bbl@tempb` just as `\@newl@bel` does it.

```
2533   \def\bbl@tempb{#3}%
2534   \@safe@activesfalse
```

When the label is defined we replace the definition of `\bbl@tempa` by its meaning.

```
2535   \ifx\bbl@tempa\relax
2536   \else
2537     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2538   \fi
```

We do the same for `\bbl@tempb`.

```
2539   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

If the label didn't change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

```
2540 \ifx\bbl@tempa\bbl@tempb
2541 \else
2542 \@tempswattrue
2543 \fi}
2544 \fi
```

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

```
2545 \bbl@xin@{R}\bbl@opt@safe
2546 \ifin@
2547 \bbl@redefineroobust\ref#1{%
2548 \@safe@activetrue\org@ref{#1}\@safe@activfalse}
2549 \bbl@redefineroobust\pageref#1{%
2550 \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
2551 \else
2552 \let\org@ref\ref
2553 \let\org@pageref\pageref
2554 \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.

```
2555 \bbl@xin@{B}\bbl@opt@safe
2556 \ifin@
2557 \bbl@redefine\@citex[#1]#2{%
2558 \@safe@activetrue\edef\@tempa{#2}\@safe@activfalse
2559 \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.

```
2560 \AtBeginDocument{%
2561 \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.)

```
2562 \def\@citex[#1][#2]#3{%
2563 \@safe@activetrue\edef\@tempa{#3}\@safe@activfalse
2564 \org@@citex[#1][#2]{\@tempa}}%
2565 }{}}
```

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

```
2566 \AtBeginDocument{%
2567 \ifpackageloaded{cite}{%
2568 \def\@citex[#1]#2{%
2569 \@safe@activetrue\org@@citex[#1]{#2}\@safe@activfalse}%
2570 }{}}
```


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

```

2571 \bbl@redefine\nocite#1{%
2572   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

2573 \bbl@redefine\bibcite{%
2574   \bbl@cite@choice
2575   \bibcite}

```

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

```

2576 \def\bbl@bibcite#1#2{%
2577   \org@bibcite{#1}{\@safe@activesfalse#2}}

```

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

```

2578 \def\bbl@cite@choice{%
2579   \global\let\bibcite\bbl@bibcite

```

Then, when natbib is loaded we restore the original definition of `\bibcite`. For cite we do the same.

```

2580   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
2581   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%

```

Make sure this only happens once.

```

2582   \global\let\bbl@cite@choice\relax}

```

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

```

2583 \AtBeginDocument{\bbl@cite@choice}

```

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

```

2584 \bbl@redefine\@bibitem#1{%
2585   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
2586 \else
2587   \let\org@nocite\nocite
2588   \let\org@@citex\@citex
2589   \let\org@bibcite\bibcite
2590   \let\org@@bibitem\@bibitem
2591 \fi

```

11.3 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat.

We check whether the argument is empty; if it is, we just make sure the scratch token register is empty. Next, we store the argument to `\markright` in the scratch token register. This way these commands will not be expanded later, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while `\@safe@activestrue` is in effect.

```

2592 \bbl@trace{Marks}
2593 \IfBabelLayout{sectioning}
2594   {\ifx\bbl@opt@headfoot\@nnil
2595     \g@addto@macro\@resetactivechars{%
2596       \set@typeset@protect
2597       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2598       \let\protect\noexpand
2599       \edef\thepage{%
2600         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2601     \fi}
2602   {\ifbbl@single\else
2603     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
2604     \markright#1{%
2605       \bbl@ifblank{#1}%
2606       {\org@markright{}}}%
2607     {\toks@{#1}%
2608       \bbl@exp{%
2609         \\org@markright{\\protect\\foreignlanguage{\language}%
2610           {\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

2611   \ifx\mkboth\markboth
2612     \def\bbl@tempc{\let\mkboth\markboth}
2613   \else
2614     \def\bbl@tempc{
2615       \fi
2616       \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
2617       \markboth#1#2{%
2618         \protected@edef\bbl@tempb##1{%
2619           \protect\foreignlanguage
2620             {\language}{\protect\bbl@restore@actives##1}}%
2621         \bbl@ifblank{#1}%
2622         {\toks@{}}%
2623         {\toks@\expandafter{\bbl@tempb{#1}}}%
2624         \bbl@ifblank{#2}%
2625         {\@temptokena{}}%
2626         {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2627         \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
2628         \bbl@tempc
2629       \fi} % end ifbbl@single, end \IfBabelLayout

```

11.4 Preventing clashes with other packages

11.4.1 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.

The first thing we need to do is check if the package `ifthen` is loaded. This should be done at `\begin{document}` time.

```
2630 \bbl@trace{Preventing clashes with other packages}
2631 \bbl@xin@{R}\bbl@opt@safe
2632 \ifin@
2633   \AtBeginDocument{%
2634     \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```
2635       \bbl@redefine@long\ifthenelse#1#2#3{%
```

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.

```
2636       \let\bbl@temp@pref\pageref
2637       \let\pageref\org@pageref
2638       \let\bbl@temp@ref\ref
2639       \let\ref\org@ref
```

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. When the package wasn't loaded we do nothing.

```
2640       \@safe@activestrue
2641       \org@ifthenelse{#1}%
2642       {\let\pageref\bbl@temp@pref
2643        \let\ref\bbl@temp@ref
2644        \@safe@activesfalse
2645        #2}%
2646       {\let\pageref\bbl@temp@pref
2647        \let\ref\bbl@temp@ref
2648        \@safe@activesfalse
2649        #3}%
2650       }%
2651     }{}%
2652   }
```

11.4.2 varioref

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

`\Ref` The same needs to happen for `\vrefpagemum`.

```
2653   \AtBeginDocument{%
2654     \@ifpackageloaded{varioref}{%
```

```

2655 \bbl@redefine\@@vpageref#1[#2]#3{%
2656 \@safe@activestrue
2657 \org@@@vpageref{#1}[#2]#3}%
2658 \@safe@activesfalse}%
2659 \bbl@redefine\hrefpagenum#1#2{%
2660 \@safe@activestrue
2661 \org@hrefpagenum{#1}#2}%
2662 \@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.

```

2663 \expandafter\def\csname Ref\endcsname#1{%
2664 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2665 }{}%
2666 }
2667 \fi

```

11.4.3 `hhline`

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the ‘:’ character which is made active by the french support in `babel`. Therefore we need to *reload* the package when the ‘:’ is an active character.

So at `\begin{document}` we check whether `hhline` is loaded.

```

2668 \AtEndOfPackage{%
2669 \AtBeginDocument{%
2670 \ifpackageloaded{hhline}%

```

Then we check whether the expansion of `\normal@char:` is not equal to `\relax`.

```

2671 {\expandafter\ifx\csname normal@char\string\endcsname\relax
2672 \else

```

In that case we simply reload the package. 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.

```

2673 \makeatletter
2674 \def\@currname{hhline}\input{hhline.sty}\makeatother
2675 \fi}%
2676 {}}}

```

11.4.4 `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.

```

2677 \AtBeginDocument{%
2678 \ifx\pdfstringdefDisableCommands\undefined\else
2679 \pdfstringdefDisableCommands{\languageshortands{system}}%
2680 \fi}

```

11.4.5 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.

```
2681 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2682   \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.

```
2683 \def\substitutefontfamily#1#2#3{%
2684   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2685   \immediate\write15{%
2686     \string\ProvidesFile{#1#2.fd}%
2687     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2688     \space generated font description file]^J
2689     \string\DeclareFontFamily{#1}{#2}{^^J
2690     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
2691     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
2692     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
2693     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
2694     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
2695     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
2696     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
2697     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
2698     }%
2699     \closeout15
2700   }
```

This command should only be used in the preamble of a document.

```
2701 \@onlypreamble\substitutefontfamily
```

11.5 Encoding and fonts

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

`\ensureascii`

```
2702 \bbl@trace{Encoding and fonts}
2703 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
2704 \newcommand\BabelNonText{TS1,T3,TS3}
2705 \let\org@TeX\TeX
2706 \let\org@LaTeX\LaTeX
2707 \let\ensureascii\@firstofone
2708 \AtBeginDocument{%
2709   \in@false
2710   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2711     \ifin@false
2712       \lowercase{\bbl@xin@{,#1enc.def},{,\@filelist,}}%
2713       \fi}%
2714   \ifin@ % if a text non-ascii has been loaded
```

```

2715 \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2716 \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2717 \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2718 \def\bbl@tempb#1@@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@@}%
2719 \def\bbl@tempc#1ENC.DEF#2\@@{%
2720   \ifx\empty#2\else
2721     \bbl@ifunset{T@#1}%
2722     {}%
2723     {\bbl@xin@{, #1, }{\, \BabelNonASCII, \BabelNonText, }%
2724     \ifin@
2725       \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2726       \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2727     \else
2728       \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2729       \fi}%
2730   \fi}%
2731 \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
2732 \bbl@xin@{, \cf@encoding, }{\, \BabelNonASCII, \BabelNonText, }%
2733 \ifin@\else
2734   \edef\ensureascii#1{{%
2735     \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2736   \fi
2737 \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.

```

2738 \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.

```

2739 \AtBeginDocument{%
2740   \@ifpackageloaded{fontspec}%
2741   {\xdef\latinencoding{%
2742     \ifx\UTFencname\@undefined
2743       EU\ifcase\bbl@engine\or2\or1\fi
2744     \else
2745       \UTFencname
2746     \fi}}%
2747   {\gdef\latinencoding{OT1}%
2748     \ifx\cf@encoding\bbl@t@one
2749       \xdef\latinencoding{\bbl@t@one}%
2750     \else
2751       \ifx\@fontenc@load@list\@undefined
2752         \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
2753       \else
2754         \def\@elt#1{, #1,}%
2755         \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
2756         \let\@elt\relax
2757         \bbl@xin@{, T1, }\bbl@tempa
2758         \ifin@
2759           \xdef\latinencoding{\bbl@t@one}%

```

```

2760         \fi
2761     \fi
2762 \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.

```

2763 \DeclareRobustCommand{\latintext}{%
2764     \fontencoding{\latinencoding}\selectfont
2765     \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.

```

2766 \ifx\@undefined\DeclareTextFontCommand
2767     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2768 \else
2769     \DeclareTextFontCommand{\textlatin}{\latintext}
2770 \fi

```

11.6 Basic bidi support

Work in progress. This code is currently placed here for practical reasons.

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

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

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

```

2771 \bbl@trace{Basic (internal) bidi support}
2772 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2773 \def\bbl@rscripts{%
2774     ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2775     Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
2776     Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
2777     Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2778     Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2779     Old South Arabian,}%
2780 \def\bbl@provide@dirs#1{%
2781     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2782     \ifin@
2783         \global\bbl@csarg\chardef{wdir@#1}\@ne
2784         \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%

```

```

2785 \ifin@
2786 \global\bbbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2787 \fi
2788 \else
2789 \global\bbbl@csarg\chardef{wdir@#1}\z@
2790 \fi
2791 \ifodd\bbbl@engine
2792 \bbbl@csarg\ifcase{wdir@#1}%
2793 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
2794 \or
2795 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
2796 \or
2797 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
2798 \fi
2799 \fi}
2800 \def\bbbl@switchdir{%
2801 \bbbl@ifunset{bbbl@sys@\language}{\bbbl@provide@sys{\language}}{}%
2802 \bbbl@ifunset{bbbl@wdir@\language}{\bbbl@provide@dirs{\language}}{}%
2803 \bbbl@exp{\bbbl@setdirs\bbbl@cs{wdir@\language}}
2804 \def\bbbl@setdirs#1{% TODO - math
2805 \ifcase\bbbl@select@type % TODO - strictly, not the right test
2806 \bbbl@bodydir{#1}%
2807 \bbbl@pdir{#1}%
2808 \fi
2809 \bbbl@texdir{#1}}
2810 \ifodd\bbbl@engine % luatex=1
2811 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
2812 \DisableBabelHook{babel-bidi}
2813 \chardef\bbbl@thetexdir\z@
2814 \chardef\bbbl@thepardir\z@
2815 \def\bbbl@getluadir#1{%
2816 \directlua{
2817 if tex.#1dir == 'TLT' then
2818 tex.sprint('0')
2819 elseif tex.#1dir == 'TRT' then
2820 tex.sprint('1')
2821 end}}
2822 \def\bbbl@setluadir#1#2#3{% 1=text/par.. 2=\texdir.. 3=0 lr/1 rl
2823 \ifcase#3\relax
2824 \ifcase\bbbl@getluadir{#1}\relax\else
2825 #2 TLT\relax
2826 \fi
2827 \else
2828 \ifcase\bbbl@getluadir{#1}\relax
2829 #2 TRT\relax
2830 \fi
2831 \fi}
2832 \def\bbbl@texdir#1{%
2833 \bbbl@setluadir{tex}\texdir{#1}%
2834 \chardef\bbbl@thetexdir#1\relax
2835 \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*3+#1}}
2836 \def\bbbl@pdir#1{%
2837 \bbbl@setluadir{par}\pdir{#1}%
2838 \chardef\bbbl@thepardir#1\relax}
2839 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}
2840 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}
2841 \def\bbbl@dirparastext{\pdir\the\texdir\relax}% %%%
2842 % Sadly, we have to deal with boxes in math with basic.
2843 % Activated every math with the package option bidi=:

```



```

2844 \def\bbl@mathboxdir{%
2845   \ifcase\bbl@thetextdir\relax
2846     \everyhbox{\textdir TLT\relax}%
2847   \else
2848     \everyhbox{\textdir TRT\relax}%
2849   \fi}
2850 \else % pdftex=0, xetex=2
2851   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2852   \DisableBabelHook{babel-bidi}
2853   \newcount\bbl@dirlevel
2854   \chardef\bbl@thetextdir\z@
2855   \chardef\bbl@thepardir\z@
2856   \def\bbl@textdir#1{%
2857     \ifcase#1\relax
2858       \chardef\bbl@thetextdir\z@
2859       \bbl@textdir@i\beginL\endL
2860     \else
2861       \chardef\bbl@thetextdir\@ne
2862       \bbl@textdir@i\beginR\endR
2863     \fi}
2864   \def\bbl@textdir@i#1#2{%
2865     \ifhmode
2866       \ifnum\currentgrouplevel>\z@
2867         \ifnum\currentgrouplevel=\bbl@dirlevel
2868           \bbl@error{Multiple bidi settings inside a group}%
2869           {I'll insert a new group, but expect wrong results.}%
2870           \bgroup\aftergroup#2\aftergroup\egroup
2871         \else
2872           \ifcase\currentgrouptype\or % 0 bottom
2873             \aftergroup#2% 1 simple {}
2874           \or
2875             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2876           \or
2877             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2878           \or\or\or % vbox vtop align
2879           \or
2880             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2881           \or\or\or\or\or\or % output math disc insert vcent mathchoice
2882           \or
2883             \aftergroup#2% 14 \begingroup
2884           \else
2885             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2886           \fi
2887         \fi
2888         \bbl@dirlevel\currentgrouplevel
2889       \fi
2890       #1%
2891     \fi}
2892   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2893   \let\bbl@bodydir\@gobble
2894   \let\bbl@pagedir\@gobble
2895   \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the `par` direction. Note text and par dirs are decoupled to some extent (although not completely).

```

2896 \def\bbl@xebidipar{%
2897   \let\bbl@xebidipar\relax
2898   \TeXeTstate\@ne

```

```

2899 \def\bbl@xeverypar{%
2900 \ifcase\bbl@thepardir
2901 \ifcase\bbl@thetextdir\else\beginR\fi
2902 \else
2903 {\setbox\z@\lastbox\beginR\box\z@}%
2904 \fi}%
2905 \let\bbl@severypar\everypar
2906 \newtoks\everypar
2907 \everypar=\bbl@severypar
2908 \bbl@severypar{\bbl@xeverypar\the\everypar}}
2909 \def\bbl@tempb{%
2910 \let\bbl@textdir\i\@gobbletwo
2911 \let\bbl@xebidipar\@empty
2912 \AddBabelHook{bidi}{foreign}{%
2913 \def\bbl@tempa{\def\BabelText#####1}%
2914 \ifcase\bbl@thetextdir
2915 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{#####1}}}%
2916 \else
2917 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{#####1}}}%
2918 \fi}
2919 \def\bbl@pardir##1{\ifcase##1\relax\setLR\else\setRL\fi}}
2920 \@ifpackagewith{babel}{bidi=bidi}{\bbl@tempb}{}%
2921 \@ifpackagewith{babel}{bidi=bidi-l}{\bbl@tempb}{}%
2922 \@ifpackagewith{babel}{bidi=bidi-r}{\bbl@tempb}{}%
2923 \fi

```

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

```

2924 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}%
2925 \AtBeginDocument{%
2926 \ifx\pdfstringdefDisableCommands\@undefined\else
2927 \ifx\pdfstringdefDisableCommands\relax\else
2928 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2929 \fi
2930 \fi}

```

11.7 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`.

```

2931 \bbl@trace{Local Language Configuration}
2932 \ifx\loadlocalcfg\@undefined
2933 \@ifpackagewith{babel}{noconfigs}%
2934 {\let\loadlocalcfg\@gobble}%
2935 {\def\loadlocalcfg#1{%
2936 \InputIfFileExists{#1.cfg}%
2937 {\typeout{*****^J%
2938 * Local config file #1.cfg used^^J%
2939 *}}%
2940 \@empty}}
2941 \fi

```

Just to be compatible with L^AT_EX 2.09 we add a few more lines of code:

```

2942 \ifx\@unexpandable@protect\@undefined
2943 \def\@unexpandable@protect{\noexpand\protect\noexpand}

```

```

2944 \long\def\protected@write#1#2#3{%
2945   \begingroup
2946     \let\thepage\relax
2947     #2%
2948     \let\protect\@unexpandable@protect
2949     \edef\reserved@a{\write#1{#3}}%
2950     \reserved@a
2951   \endgroup
2952   \if@nobreak\ifvmode\nobreak\fi\fi}
2953 \fi
2954 </core>
2955 <*kernel>

```

12 Multiple languages (switch.def)

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

```

2956 <<Make sure ProvidesFile is defined>>
2957 \ProvidesFile{switch.def}[\<date>] [\<version>] Babel switching mechanism]
2958 <<Load macros for plain if not LaTeX>>
2959 <<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.

```

2960 \def\bbl@version{\<version>}
2961 \def\bbl@date{\<date>}
2962 \def\adddialect#1#2{%
2963   \global\chardef#1#2\relax
2964   \bbl@usehooks{adddialect}{\{#1\}{#2\}}%
2965   \begingroup
2966     \count@#1\relax
2967     \def\bbl@elt##1##2##3##4{%
2968       \ifnum\count@=##2\relax
2969         \bbl@info{\string#1 = using hyphenrules for ##1\%
2970           (\string\language\the\count@)}%
2971         \def\bbl@elt####1####2####3####4{%
2972           \fi}%
2973         \@nameuse{bbl@languages}%
2974       \endgroup}

```

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

```

2975 \def\bbl@fixname#1{%
2976   \begingroup
2977   \def\bbl@tempe{#1}%
2978   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2979   \bbl@tempd
2980   {\lowercase\expandafter{\bbl@tempd}%
2981    {\uppercase\expandafter{\bbl@tempd}%
2982    \@empty

```

```

2983      {\edef\bb1@tempd{\def\noexpand#1{#1}}}%
2984      \uppercase\expandafter{\bb1@tempd}}}%
2985      {\edef\bb1@tempd{\def\noexpand#1{#1}}}%
2986      \lowercase\expandafter{\bb1@tempd}}}%
2987      \@empty
2988      \edef\bb1@tempd{\endgroup\def\noexpand#1{#1}}}%
2989      \bb1@tempd}
2990 \def\bb1@iflanguage#1{%
2991   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

`\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.

```

2992 \def\iflanguage#1{%
2993   \bb1@iflanguage{#1}{%
2994     \ifnum\csname l@#1\endcsname=\language
2995       \expandafter\@firstoftwo
2996     \else
2997       \expandafter\@secondoftwo
2998     \fi}}

```

12.1 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.

To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character. To convert a control sequence to a string, we use the `\string` primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer `\escapechar` to a character number, we have to compare this number with the character of the string. To do this we have to use \TeX 's backquote notation to specify the character as a number. If the first character of the `\string`'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or `\escapechar` is set to a value outside of the character range 0–255.

If the user gives an empty argument, we provide a default argument for `\string`. This argument should expand to nothing.

```

2999 \let\bb1@select@type\z@
3000 \edef\selectlanguage{%
3001   \noexpand\protect
3002   \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`.

```

3003 \ifx\@undefined\protect\let\protect\relax\fi

```

As \LaTeX 2.09 writes to files *expanded* whereas \LaTeX 2_ε takes care *not* to expand the arguments of `\write` statements we need to be a bit clever about the way we add information to .aux files. Therefore we introduce the macro `\xstring` which should expand to the right amount of `\string`'s.

```

3004 \ifx\documentclass\@undefined
3005   \def\xstring{\string\string\string}
3006 \else
3007   \let\xstring\string
3008 \fi

```

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.

```

3009 \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:

```

3010 \def\bbl@push@language{%
3011   \xdef\bbl@language@stack{\language+\bbl@language@stack}

```

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 (delimited by '-') in its third argument.

```

3012 \def\bbl@pop@lang#1+#2-#3{%
3013   \edef\language{#1}\xdef#3{#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) followed by the '-'-sign and finally the reference to the stack.

```

3014 \let\bbl@ifrestoring\@secondoftwo
3015 \def\bbl@pop@language{%
3016   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
3017   \let\bbl@ifrestoring\@firstoftwo
3018   \expandafter\bbl@set@language\expandafter{\language}%
3019   \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.

```

3020 \chardef\localeid\z@
3021 \def\bbl@id@last{0} % No real need for a new counter
3022 \def\bbl@id@assign{%
3023   \bbl@ifunset{bbl@id@@\language}%
3024   {\count@bbl@id@last\relax
3025     \advance\count@one
3026     \bbl@csarg\chardef{id@@\language}\count@
3027     \edef\bbl@id@last{\the\count@}%
3028     \ifcase\bbl@engine\or
3029       \directlua{
3030         Babel = Babel or {}
3031         Babel.locale_props = Babel.locale_props or {}
3032         Babel.locale_props[\bbl@id@last] = {}
3033         Babel.locale_props[\bbl@id@last].name = '\language'
3034       }%
3035     \fi}%
3036   }%
3037   \chardef\localeid\@nameuse{bbl@id@@\language}}

```

The unprotected part of \selectlanguage.

```

3038 \expandafter\def\csname selectlanguage \endcsname#1{%
3039   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@fi
3040   \bbl@push@language
3041   \aftergroup\bbl@pop@language
3042   \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.

```

3043 \def\BabelContentsFiles{toc,lof,lot}
3044 \def\bbl@set@language#1{% from selectlanguage, pop@
3045   \edef\language{%
3046     \ifnum\escapechar=\expandafter`\string#1\@empty
3047     \else\string#1\@empty\fi}%
3048   \select@language{\language}%
3049   % write to auxs
3050   \expandafter\ifx\csname date\language\endcsname\relax\else
3051     \if@files
3052       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
3053         \protected@write\@auxout{}\string\babel@aux{\language}{}}%
3054       \fi
3055       \bbl@usehooks{write}{}%
3056     \fi
3057   \fi}
3058 \def\select@language#1{% from set@, babel@aux
3059   % set hmap
3060   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
3061   % set name
3062   \edef\language{#1}%
3063   \bbl@fixname\language

```

```

3064 \expandafter\ifx\csname date\language\endcsname\relax
3065   \IfFileExists{babel-\language.tex}%
3066     {\babelprovide{\language}}%
3067     {}%
3068 \fi
3069 \bbl@iflanguage\language{%
3070   \expandafter\ifx\csname date\language\endcsname\relax
3071     \bbl@error
3072     {Unknown language `#1'. Either you have\\%
3073       misspelled its name, it has not been installed,\\%
3074       or you requested it in a previous run. Fix its name,\\%
3075       install it or just rerun the file, respectively. In\\%
3076       some cases, you may need to remove the aux file}%
3077     {You may proceed, but expect wrong results}%
3078   \else
3079     % set type
3080     \let\bbl@select@type\z@
3081     \expandafter\bbl@switch\expandafter{\language}%
3082   \fi}}
3083 \def\babel@aux#1#2{%
3084   \expandafter\ifx\csname date#1\endcsname\relax
3085     \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
3086       \@namedef{bbl@auxwarn@#1}{}%
3087       \bbl@warning
3088       {Unknown language `#1'. Very likely you\\%
3089         requested it in a previous run. Expect some\\%
3090         wrong results in this run, which should vanish\\%
3091         in the next one. Reported}%
3092     \fi
3093   \else
3094     \select@language{#1}%
3095     \bbl@foreach\BabelContentsFiles{%
3096       \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
3097   \fi}
3098 \def\babel@toc#1#2{%
3099   \select@language{#1}}

```

A bit of optimization. Select in heads/foots the language only if necessary. The real thing is in `babel.def`.

```

3100 \let\select@language@x\select@language

```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring \TeX in a certain pre-defined state.

The name of the language is stored in the control sequence `\language`.

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\<lang>hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\<lang>hyphenmins` will be used.

```

3101 \newif\ifbbl@usedategroup
3102 \def\bbl@switch#1{% from select@, foreign@

```

```

3103 % make sure there is info for the language if so requested
3104 \bbl@ensureinfo{#1}%
3105 % restore
3106 \originalTeX
3107 \expandafter\def\expandafter\originalTeX\expandafter{%
3108   \csname noextras#1\endcsname
3109   \let\originalTeX\@empty
3110   \babel@beginsave}%
3111 \bbl@usehooks{afterreset}{}%
3112 \languageshorthands{none}%
3113 % set the locale id
3114 \bbl@id@assign
3115 % switch captions, date
3116 \ifcase\bbl@select@type
3117   \ifhmode
3118     \hskip\z@skip % trick to ignore spaces
3119     \csname captions#1\endcsname\relax
3120     \csname date#1\endcsname\relax
3121     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
3122   \else
3123     \csname captions#1\endcsname\relax
3124     \csname date#1\endcsname\relax
3125   \fi
3126 \else
3127   \ifbbl@usedategroup % if \foreign... within \<lang>date
3128     \bbl@usedategroupfalse
3129     \ifhmode
3130       \hskip\z@skip % trick to ignore spaces
3131       \csname date#1\endcsname\relax
3132       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
3133     \else
3134       \csname date#1\endcsname\relax
3135     \fi
3136   \fi
3137 \fi
3138 % switch extras
3139 \bbl@usehooks{beforeextras}{}%
3140 \csname extras#1\endcsname\relax
3141 \bbl@usehooks{afterextras}{}%
3142 % > babel-ensure
3143 % > babel-sh-<short>
3144 % > babel-bidi
3145 % > babel-fontspec
3146 % hyphenation - case mapping
3147 \ifcase\bbl@opt@hyphenmap\or
3148   \def\BabelLower##1##2{\lcode##1=##2\relax}%
3149   \ifnum\bbl@hymapsel>4\else
3150     \csname\language @bbl@hyphenmap\endcsname
3151   \fi
3152   \chardef\bbl@opt@hyphenmap\z@
3153 \else
3154   \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
3155     \csname\language @bbl@hyphenmap\endcsname
3156   \fi
3157 \fi
3158 \global\let\bbl@hymapsel\@cclv
3159 % hyphenation - patterns
3160 \bbl@patterns{#1}%
3161 % hyphenation - mins

```



```

3162 \babel@savevariable\lefthyphenmin
3163 \babel@savevariable\righthyphenmin
3164 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3165 \set@hyphenmins\tw@thr@\relax
3166 \else
3167 \expandafter\expandafter\expandafter\set@hyphenmins
3168 \csname #1hyphenmins\endcsname\relax
3169 \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.

```

3170 \long\def\otherlanguage#1{%
3171 \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\thr@\fi
3172 \csname selectlanguage\endcsname{#1}%
3173 \ignorespaces}

```

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

```

3174 \long\def\endotherlanguage{%
3175 \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`.

```

3176 \expandafter\def\csname otherlanguage*\endcsname#1{%
3177 \ifnum\bbl@hymapsel=\cclv\chardef\bbl@hymapsel4\relax\fi
3178 \foreign@language{#1}}

```

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”.

```

3179 \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.

```

3180 \providecommand\bbl@beforeforeign{
3181 \edef\foreignlanguage{%
3182   \noexpand\protect
3183   \expandafter\noexpand\csname foreignlanguage \endcsname}
3184 \expandafter\def\csname foreignlanguage \endcsname{%
3185   \@ifstar\bbl@foreign@s\bbl@foreign@x}
3186 \def\bbl@foreign@x#1#2{%
3187   \begingroup
3188     \let\BabelText\@firstofone
3189     \bbl@beforeforeign
3190     \foreign@language{#1}%
3191     \bbl@usehooks{foreign}{}%
3192     \BabelText{#2}% Now in horizontal mode!
3193   \endgroup}
3194 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
3195   \begingroup
3196     {\par}%
3197     \let\BabelText\@firstofone
3198     \foreign@language{#1}%
3199     \bbl@usehooks{foreign*}{}%
3200     \bbl@dirparastext
3201     \BabelText{#2}% Still in vertical mode!
3202     {\par}%
3203   \endgroup}

```

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

```

3204 \def\foreign@language#1{%
3205   % set name
3206   \edef\language#1%
3207   \bbl@fixname\language
3208   \expandafter\ifx\csname date\language\endcsname\relax
3209     \IfFileExists{babel-\language.tex}%
3210       {\babelprovide{\language}}%
3211       {}%
3212   \fi
3213   \bbl@iflanguage\language%
3214     \expandafter\ifx\csname date\language\endcsname\relax
3215       \bbl@warning % TODO - why a warning, not an error?
3216       {Unknown language `#1'. Either you have\\%
3217         misspelled its name, it has not been installed,\\%
3218         or you requested it in a previous run. Fix its name,\\%
3219         install it or just rerun the file, respectively. In\\%
3220         some cases, you may need to remove the aux file.\\%
3221         I'll proceed, but expect wrong results.\\%
3222         Reported}%
3223     \fi
3224     % set type
3225     \let\bbl@select@type\@ne
3226     \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.

```

3227 \let\bbl@hyphlist\@empty
3228 \let\bbl@hyphenation@ \relax
3229 \let\bbl@pttnlist\@empty
3230 \let\bbl@patterns@ \relax
3231 \let\bbl@hymapset=\ccclv
3232 \def\bbl@patterns#1{%
3233   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3234     \csname l@#1\endcsname
3235     \edef\bbl@tempa{#1}%
3236   \else
3237     \csname l@#1:\f@encoding\endcsname
3238     \edef\bbl@tempa{#1:\f@encoding}%
3239   \fi
3240   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
3241   % > luatex
3242   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
3243     \begingroup
3244       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
3245       \ifin@else
3246         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
3247         \hyphenation{%
3248           \bbl@hyphenation@
3249           \@ifundefined{bbl@hyphenation@#1}%
3250             \@empty
3251             {\space\csname bbl@hyphenation@#1\endcsname}}%
3252         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
3253       \fi
3254     \endgroup}}

```

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

```

3255 \def\hyphenrules#1{%
3256   \edef\bbl@tempf{#1}%
3257   \bbl@fixname\bbl@tempf
3258   \bbl@iflanguage\bbl@tempf{%
3259     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
3260     \languageshortands{none}%
3261     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
3262       \set@hyphenmins\tw@\thr@@\relax
3263     \else
3264       \expandafter\expandafter\expandafter\set@hyphenmins
3265       \csname\bbl@tempf hyphenmins\endcsname\relax
3266     \fi}}
3267 \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 \langhyphenmins is already defined this command has no effect.

```

3268 \def\providehyphenmins#1#2{%

```

```

3269 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3270 \namedef{#1hyphenmins}{#2}%
3271 \fi}

```

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

```

3272 \def\set@hyphenmins#1#2{%
3273 \lefthyphenmin#1\relax
3274 \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.

```

3275 \ifx\ProvidesFile\@undefined
3276 \def\ProvidesLanguage#1[#2 #3 #4]{%
3277 \wlog{Language: #1 #4 #3 <#2>}%
3278 }
3279 \else
3280 \def\ProvidesLanguage#1{%
3281 \begingroup
3282 \catcode`\ 10 %
3283 \@makeother\/%
3284 \@ifnextchar[%]
3285 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
3286 \def\@provideslanguage#1[#2]{%
3287 \wlog{Language: #1 #2}%
3288 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
3289 \endgroup}
3290 \fi

```

`\LdfInit` This macro is defined in two versions. The first version is to be part of the ‘kernel’ of babel, ie. the part that is loaded in the format; the second version is defined in `babel.def`. The version in the format just checks the category code of the ampersand and then loads `babel.def`. The category code of the ampersand is restored and the macro calls itself again with the new definition from `babel.def`

```

3291 \def\LdfInit{%
3292 \chardef\atcatcode=\catcode`\@
3293 \catcode`\@=11\relax
3294 \input babel.def\relax
3295 \catcode`\@=\atcatcode \let\atcatcode\relax
3296 \LdfInit}

```

`\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`.

```

3297 \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 initialises the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

3298 \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’:

```

3299 \providecommand\setlocale{%

```

```

3300 \bbl@error
3301 {Not yet available}%
3302 {Find an armchair, sit down and wait}}
3303 \let\uselocale\setlocale
3304 \let\locale\setlocale
3305 \let\selectlocale\setlocale
3306 \let\localename\setlocale
3307 \let\textlocale\setlocale
3308 \let\textlanguage\setlocale
3309 \let\languagetext\setlocale

```

12.2 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 $\text{\LaTeX 2}_{\epsilon}$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.
 Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

3310 \edef\bbl@nulllanguage{\string\language=0}
3311 \ifx\PackageError\@undefined
3312 \def\bbl@error#1#2{%
3313 \begingroup
3314 \newlinechar=`^^J
3315 \def\{^^J(babel) }%
3316 \errhelp{#2}\errmessage{\{#1}%
3317 \endgroup}
3318 \def\bbl@warning#1{%
3319 \begingroup
3320 \newlinechar=`^^J
3321 \def\{^^J(babel) }%
3322 \message{\{#1}%
3323 \endgroup}
3324 \let\bbl@infowarn\bbl@warning
3325 \def\bbl@info#1{%
3326 \begingroup
3327 \newlinechar=`^^J
3328 \def\{^^J}%
3329 \wlog{#1}%
3330 \endgroup}
3331 \else
3332 \def\bbl@error#1#2{%
3333 \begingroup
3334 \def\{\MessageBreak}%
3335 \PackageError{babel}{#1}{#2}%
3336 \endgroup}
3337 \def\bbl@warning#1{%
3338 \begingroup
3339 \def\{\MessageBreak}%
3340 \PackageWarning{babel}{#1}%
3341 \endgroup}

```

```

3342 \def\bbl@infowarn#1{%
3343   \begingroup
3344     \def\{\MessageBreak}%
3345     \GenericWarning
3346       {(babel) \@spaces\@spaces\@spaces}%
3347       {Package babel Info: #1}%
3348   \endgroup}
3349 \def\bbl@info#1{%
3350   \begingroup
3351     \def\{\MessageBreak}%
3352     \PackageInfo{babel}{#1}%
3353   \endgroup}
3354 \fi
3355 \@ifpackagewith{babel}{silent}
3356   {\let\bbl@info@gobble
3357    \let\bbl@infowarn@gobble
3358    \let\bbl@warning@gobble}
3359 {}
3360 \def\bbl@nocaption{\protect\bbl@nocaption@i}
3361 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
3362   \global\@namedef{#2}{\textbf{?#1?}}%
3363   \@nameuse{#2}%
3364   \bbl@warning{%
3365     \@backslashchar#2 not set. Please, define\\%
3366     it in the preamble with something like:\\%
3367     \string\renewcommand\@backslashchar#2{..}\\%
3368     Reported}}
3369 \def\bbl@tentative{\protect\bbl@tentative@i}
3370 \def\bbl@tentative@i#1{%
3371   \bbl@warning{%
3372     Some functions for '#1' are tentative.\\%
3373     They might not work as expected and their behavior\\%
3374     could change in the future.\\%
3375     Reported}}
3376 \def\@nolanerr#1{%
3377   \bbl@error
3378   {You haven't defined the language #1\space yet}%
3379   {Your command will be ignored, type <return> to proceed}}
3380 \def\@nopatterns#1{%
3381   \bbl@warning
3382   {No hyphenation patterns were preloaded for\\%
3383    the language '#1' into the format.\\%
3384    Please, configure your TeX system to add them and\\%
3385    rebuild the format. Now I will use the patterns\\%
3386    preloaded for \bbl@nulllanguage\space instead}}
3387 \let\bbl@usehooks@gobbletwo
3388 \</kernel>
3389 \<*patterns>

```

13 Loading hyphenation patterns

The following code is meant to be read by $\text{\texttt{iniTeX}}$ because it should instruct $\text{\texttt{TeX}}$ 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.

We want to add a message to the message $\text{\texttt{L\TeX 2.09}}$ puts in the `\everyjob` register. This could be done by the following code:

```

\let\orgeveryjob\everyjob
\def\everyjob#1{%
  \orgeveryjob{#1}%
  \orgeveryjob\expandafter{\the\orgeveryjob\immediate\write16{%
    hyphenation patterns for \the\loaded@patterns loaded.}}%
\let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}

```

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before \LaTeX fills the register. There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with \LaTeX the above scheme won't work. The reason is that \LaTeX overwrites the contents of the `\everyjob` register with its own message.
- Plain \TeX does not use the `\everyjob` register so the message would not be displayed.

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, `\dump`. Therefore the original `\dump` is saved in `\org@dump` and a new definition is supplied. 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.

```

3390 <<Make sure ProvidesFile is defined>>
3391 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
3392 \xdef\bb1@format{\jobname}
3393 \ifx\AtBeginDocument\@undefined
3394   \def\@empty{}
3395   \let\orig@dump\dump
3396   \def\dump{%
3397     \ifx\@ztryfc\@undefined
3398     \else
3399       \toks0=\expandafter{\@preamblecmds}%
3400       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3401       \def\@begindocumenthook{}%
3402     \fi
3403     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3404 \fi
3405 <<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.

```

3406 \def\process@line#1#2 #3 #4 {%
3407   \ifx=#1%
3408     \process@synonym{#2}%
3409   \else
3410     \process@language{#1#2}{#3}{#4}%
3411   \fi
3412   \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.

```
3413 \toks@{}
3414 \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.

```
3415 \def\process@synonym#1{%
3416   \ifnum\last@language=\m@ne
3417     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3418   \else
3419     \expandafter\chardef\csname l@#1\endcsname\last@language
3420     \wlog{\string\l@#1=\string\language\the\last@language}%
3421     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3422       \csname\language\hyphenmins\endcsname
3423     \let\bbl@elt\relax
3424     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
3425   \fi}
```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` and `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```
3426 \def\process@language#1#2#3{%
3427   \expandafter\addlanguage\csname l@#1\endcsname
```



```

3428 \expandafter\language\csname l@#1\endcsname
3429 \edef\languagename{#1}%
3430 \bbl@hook@everylanguage{#1}%
3431 % > luatex
3432 \bbl@get@enc#1::\@@@
3433 \begingroup
3434 \lefthyphenmin\m@ne
3435 \bbl@hook@loadpatterns{#2}%
3436 % > luatex
3437 \ifnum\lefthyphenmin=\m@ne
3438 \else
3439 \expandafter\xdef\csname #1hyphenmins\endcsname{%
3440 \the\lefthyphenmin\the\righthyphenmin}%
3441 \fi
3442 \endgroup
3443 \def\bbl@tempa{#3}%
3444 \ifx\bbl@tempa\@empty\else
3445 \bbl@hook@loadexceptions{#3}%
3446 % > luatex
3447 \fi
3448 \let\bbl@elt\relax
3449 \edef\bbl@languages{%
3450 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3451 \ifnum\the\language=\z@
3452 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3453 \set@hyphenmins\tw@\thr@@\relax
3454 \else
3455 \expandafter\expandafter\expandafter\set@hyphenmins
3456 \csname #1hyphenmins\endcsname
3457 \fi
3458 \the\toks@
3459 \toks@{}%
3460 \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.

```

3461 \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.

```

3462 \def\bbl@hook@everylanguage#1{}
3463 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3464 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3465 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
3466 \begingroup
3467 \def\AddBabelHook#1#2{%
3468 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3469 \def\next{\toks1}%
3470 \else
3471 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3472 \fi
3473 \next}
3474 \ifx\directlua\undefined
3475 \ifx\XeTeXinputencoding\undefined\else
3476 \input xebabel.def
3477 \fi
3478 \else
3479 \input luababel.def

```

```

3480 \fi
3481 \openin1 = babel-\bbl@format.cfg
3482 \ifeof1
3483 \else
3484 \input babel-\bbl@format.cfg\relax
3485 \fi
3486 \closein1
3487 \endgroup
3488 \bbl@hook@loadkernel{switch.def}

```

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

```

3489 \openin1 = language.dat

```

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

```

3490 \def\languagename{english}%
3491 \ifeof1
3492 \message{I couldn't find the file language.dat,\space
3493         I will try the file hyphen.tex}
3494 \input hyphen.tex\relax
3495 \chardef\l@english\z@
3496 \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`.

```

3497 \last@language\m@ne

```

We now read lines from the file until the end is found

```

3498 \loop

```

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.

```

3499 \endlinechar\m@ne
3500 \read1 to \bbl@line
3501 \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.

```

3502 \if T\ifeof1F\fi T\relax
3503 \ifx\bbl@line\@empty\else
3504 \edef\bbl@line{\bbl@line\space\space\space}%
3505 \expandafter\process@line\bbl@line\relax
3506 \fi
3507 \repeat

```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns.

```

3508 \begingroup
3509 \def\bbl@elt#1#2#3#4{%
3510 \global\language=#2\relax
3511 \gdef\languagename{#1}%
3512 \def\bbl@elt##1##2##3##4{}}%
3513 \bbl@languages
3514 \endgroup
3515 \fi

```

and close the configuration file.

```
3516 \closein1
```

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

```
3517 \if/\the\toks@/\else
3518 \errhelp{language.dat loads no language, only synonyms}
3519 \errmessage{Orphan language synonym}
3520 \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.

```
3521 \let\bbl@line\@undefined
3522 \let\process@line\@undefined
3523 \let\process@synonym\@undefined
3524 \let\process@language\@undefined
3525 \let\bbl@get@enc\@undefined
3526 \let\bbl@hyph@enc\@undefined
3527 \let\bbl@tempa\@undefined
3528 \let\bbl@hook@loadkernel\@undefined
3529 \let\bbl@hook@everylanguage\@undefined
3530 \let\bbl@hook@loadpatterns\@undefined
3531 \let\bbl@hook@loadexceptions\@undefined
3532 \</patterns>
```

Here the code for `iniTEX` ends.

14 Font handling with fontspec

Add the bidi handler just before `luaoftload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
3533 <<(*More package options)>> ≡
3534 \ifodd\bbl@engine
3535 \DeclareOption{bidi=basic-r}%
3536 {\ExecuteOptions{bidi=basic}}
3537 \DeclareOption{bidi=basic}%
3538 {\let\bbl@beforeforeign\leavevmode
3539 % TODO - to locale_props, not as separate attribute
3540 \newattribute\bbl@attr@dir
3541 % I don't like it, hackish:
3542 \frozen@everymath\expandafter{%
3543 \expandafter\bbl@mathboxdir\the\frozen@everymath}%
3544 \frozen@everydisplay\expandafter{%
3545 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}%
3546 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
3547 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
3548 \else
3549 \DeclareOption{bidi=basic-r}%
3550 {\ExecuteOptions{bidi=basic}}
3551 \DeclareOption{bidi=basic}%
3552 {\bbl@error
3553 {The bidi method 'basic' is available only in\\%
3554 luatex. I'll continue with 'bidi=default', so\\%
3555 expect wrong results}%
3556 {See the manual for further details.}%}
```

```

3557 \let\bbl@beforeforeign\leavevmode
3558 \AtEndOfPackage{%
3559 \EnableBabelHook{babel-bidi}%
3560 \bbl@xebidipar}}
3561 \def\bbl@loadxebidi#1{%
3562 \ifx\RTLfootnotetext\@undefined
3563 \AtEndOfPackage{%
3564 \EnableBabelHook{babel-bidi}%
3565 \ifx\fontspec\@undefined
3566 \usepackage{fontspec}% bidi needs fontspec
3567 \fi
3568 \usepackage#1{bidi}}%
3569 \fi}
3570 \DeclareOption{bidi=bidi}%
3571 {\bbl@tentative{bidi=bidi}%
3572 \bbl@loadxebidi{}}
3573 \DeclareOption{bidi=bidi-r}%
3574 {\bbl@tentative{bidi=bidi-r}%
3575 \bbl@loadxebidi{[rldocument]}}
3576 \DeclareOption{bidi=bidi-l}%
3577 {\bbl@tentative{bidi=bidi-l}%
3578 \bbl@loadxebidi{}}
3579 \fi
3580 \DeclareOption{bidi=default}%
3581 {\let\bbl@beforeforeign\leavevmode
3582 \ifodd\bbl@engine
3583 \newattribute\bbl@attr@dir
3584 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
3585 \fi
3586 \AtEndOfPackage{%
3587 \EnableBabelHook{babel-bidi}%
3588 \ifodd\bbl@engine\else
3589 \bbl@xebidipar
3590 \fi}}
3591 <</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`.

```

3592 <<{*Font selection}>> ≡
3593 \bbl@trace{Font handling with fontspec}
3594 \@onlypreamble\babelfont
3595 \newcommand\babelfont[2][{}% 1=langs/scripts 2=fam
3596 \bbl@foreach{#1}{%
3597 \expandafter\ifx\csname date##1\endcsname\relax
3598 \IfFileExists{babel-##1.tex}%
3599 {\babelprovide{##1}}%
3600 {}%
3601 \fi}%
3602 \edef\bbl@tempa{#1}%
3603 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3604 \ifx\fontspec\@undefined
3605 \usepackage{fontspec}%
3606 \fi
3607 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3608 \bbl@bblfont}
3609 \newcommand\bbl@bblfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
3610 \bbl@ifunset{\bbl@tempb family}%
3611 {\bbl@providedefam{\bbl@tempb}}%

```

```

3612 {\bbl@exp{%
3613   \\\bbl@sreplace\<\bbl@tempb family >%
3614   {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3615 % For the default font, just in case:
3616 \bbl@ifunset{\bbl@lsys\language}\bbl@provide@lsys{\language}}}%
3617 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3618 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}}% save bbl@rmdflt@
3619 \bbl@exp{%
3620   \let\<\bbl@\bbl@tempb dflt@\language>\<\bbl@\bbl@tempb dflt@>%
3621   \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\language>%
3622   \<\bbl@tempb default>\<\bbl@tempb family>}}}%
3623 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
3624   \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:

```

3625 \def\bbl@providfam#1{%
3626   \bbl@exp{%
3627     \\\newcommand\<#1default>{}% Just define it
3628     \\\bbl@add@list\\bbl@font@fams{#1}%
3629     \\\DeclareRobustCommand\<#1family>{%
3630       \\\not@math@alphabet\<#1family>\relax
3631       \\\fontfamily\<#1default>\selectfont}%
3632       \\\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.

```

3633 \def\bbl@nostdfont#1{%
3634   \bbl@ifunset{\bbl@WFF@f@family}%
3635   {\bbl@csarg\gdef{\bbl@WFF@f@family}}}% Flag, to avoid dupl warns
3636   \bbl@infowarn{The current font is not a babel standard family:\%
3637     #1%
3638     \fontname\font\%
3639     There is nothing intrinsically wrong with this warning, and\%
3640     you can ignore it altogether if you do not need these\%
3641     families. But if they are used in the document, you should be\%
3642     aware 'babel' will no set Script and Language for them, so\%
3643     you may consider defining a new family with \string\babelfont.\%
3644     See the manual for further details about \string\babelfont.\%
3645     Reported}}
3646   {}}}%
3647 \gdef\bbl@switchfont{%
3648   \bbl@ifunset{\bbl@lsys\language}\bbl@provide@lsys{\language}}}%
3649   \bbl@exp{% eg Arabic -> arabic
3650     \lowercase{\edef\\bbl@tempa{\bbl@cs{sname}\language}}}%
3651   \bbl@foreach\bbl@font@fams{%
3652     \bbl@ifunset{\bbl@##1dflt\language}% (1) language?
3653     {\bbl@ifunset{\bbl@##1dflt*\bbl@tempa}% (2) from script?
3654       {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
3655         {}}% 123=F - nothing!
3656         {\bbl@exp{% 3=T - from generic
3657           \global\let\<\bbl@##1dflt\language>%
3658           \<\bbl@##1dflt@>}}}%
3659         {\bbl@exp{% 2=T - from script
3660           \global\let\<\bbl@##1dflt\language>%
3661           \<\bbl@##1dflt*\bbl@tempa>}}}%
3662         {}}% 1=T - language, already defined
3663   \def\bbl@tempa{\bbl@nostdfont}}}%
3664   \bbl@foreach\bbl@font@fams{% don't gather with prev for
3665     \bbl@ifunset{\bbl@##1dflt\language}%

```

```

3666      {\bbl@cs{famrst@##1}%
3667      \global\bbl@csarg\let{famrst@##1}\relax}%
3668      {\bbl@exp{% order is relevant
3669      \\bbl@add\\originalTeX{%
3670      \\bbl@font@rst{\bbl@cs{##1dflt@language}}%
3671      \<##1default>\<##1family>{##1}}}%
3672      \\bbl@font@set\<bbl@##1dflt@language>% the main part!
3673      \<##1default>\<##1family>}}}%
3674      \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.

```

3675 \ifx\fbfamily\undefined\else % if latex
3676 \ifcase\bbl@engine % if pdftex
3677 \let\bbl@cckstdfonts\relax
3678 \else
3679 \def\bbl@cckstdfonts{%
3680 \begingroup
3681 \global\let\bbl@cckstdfonts\relax
3682 \let\bbl@tempa\empty
3683 \bbl@foreach\bbl@font@fams{%
3684 \bbl@ifunset{bbl@##1dflt@}%
3685 {\nameuse{##1family}%
3686 \bbl@csarg\gdef{WFF@fbfamily}}}% Flag
3687 \bbl@exp{\\bbl@add\\bbl@tempa* \<##1family>= fbfamily\\}%
3688 \space\space\fontname\font\\}%
3689 \bbl@csarg\xdef{##1dflt@}{fbfamily}%
3690 \expandafter\xdef\csname ##1default\endcsname{fbfamily}%
3691 }%
3692 \ifx\bbl@tempa\empty\else
3693 \bbl@infowarn{The following fonts are not babel standard families:\\%
3694 \bbl@tempa
3695 There is nothing intrinsically wrong with it, but\\%
3696 'babel' will no set Script and Language. Consider\\%
3697 defining a new family with \string\babelfont.\\%
3698 Reported}%
3699 \fi
3700 \endgroup}
3701 \fi
3702 \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.

```

3703 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
3704 \bbl@xin@{<>}{#1}%
3705 \ifin@
3706 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
3707 \fi
3708 \bbl@exp{%
3709 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
3710 \\bbl@ifsamestring{#2}{fbfamily}{\\#3\let\\bbl@tempa\relax}}}%
3711 % TODO - next should be global?, but even local does its job. I'm
3712 % still not sure -- must investigate:
3713 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
3714 \let\bbl@tempe\bbl@mapselect
3715 \let\bbl@mapselect\relax

```

```

3716 \let\bbl@temp@fam#4%          eg, '\rmfamily', to be restored below
3717 \let#4\relax                  % So that can be used with \newfontfamily
3718 \bbl@exp{%
3719   \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
3720   \<keys_if_exist:nnF>{fontspec-opentype}%
3721     {Script/\bbl@cs{sname@language}}%
3722     {\newfontscript\bbl@cs{sname@language}}%
3723     {\bbl@cs{sotf@language}}}%
3724   \<keys_if_exist:nnF>{fontspec-opentype}%
3725     {Language/\bbl@cs{lname@language}}%
3726     {\newfontlanguage\bbl@cs{lname@language}}%
3727     {\bbl@cs{lotf@language}}}%
3728   \newfontfamily\\#4%
3729   [\bbl@cs{lsys@language},#2]{#3}% ie \bbl@exp{.}{#3}
3730 \begingroup
3731   #4%
3732   \xdef#1{\f@family}%          eg, \bbl@rmdflt@lang{FreeSerif(0)}
3733 \endgroup
3734 \let#4\bbl@temp@fam
3735 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
3736 \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.

```

3737 \def\bbl@font@rst#1#2#3#4{%
3738   \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.

```

3739 \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 :-).

```

3740 \newcommand\babelFSstore[2][{%
3741   \bbl@ifblank{#1}%
3742   {\bbl@csarg\def{sname@#2}{Latin}}%
3743   {\bbl@csarg\def{sname@#2}{#1}}%
3744   \bbl@provide@dirs{#2}%
3745   \bbl@csarg\ifnum{wdir@#2}>\z@
3746     \let\bbl@beforeforeign\leavevmode
3747     \EnableBabelHook{babel-bidi}%
3748   \fi
3749   \bbl@foreach{#2}{%
3750     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
3751     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
3752     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
3753 \def\bbl@FSstore#1#2#3#4{%
3754   \bbl@csarg\edef{#2default#1}{#3}%
3755   \expandafter\addto\csname extras#1\endcsname{%
3756     \let#4#3%
3757     \ifx#3\f@family
3758       \edef#3{\csname bbl@#2default#1\endcsname}%
3759       \fontfamily{#3}\selectfont
3760     \else
3761       \edef#3{\csname bbl@#2default#1\endcsname}%
3762       \fi}%
3763   \expandafter\addto\csname noextras#1\endcsname{%
3764     \ifx#3\f@family

```

```

3765     \fontfamily{#4}\selectfont
3766     \fi
3767     \let#3#4}}
3768 \let\bbl@langfeatures\@empty
3769 \def\babelFSfeatures{% make sure \fontspec is redefined once
3770   \let\bbl@ori@fontspec\fontspec
3771   \renewcommand\fontspec[1][{}]{%
3772     \bbl@ori@fontspec[\bbl@langfeatures##1]}
3773   \let\babelFSfeatures\bbl@FSfeatures
3774   \babelFSfeatures}
3775 \def\bbl@FSfeatures#1#2{%
3776   \expandafter\addto\csname extras#1\endcsname{%
3777     \babel@save\bbl@langfeatures
3778     \edef\bbl@langfeatures{#2,}}}}
3779 <</Font selection>>

```

15 Hooks for XeTeX and LuaTeX

15.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

LaTeX sets many “codes” just before loading hyphen.cfg. That is not a problem in luatex, but in xetex they must be reset to the proper value. Most of the work is done in xe(la)tex.ini, so here we just “undo” some of the changes done by LaTeX. Anyway, for consistency LuaTeX also resets the catcodes.

```

3780 <<(*Restore Unicode catcodes before loading patterns)>> ≡
3781 \begingroup
3782   % Reset chars "80-"C0 to category "other", no case mapping:
3783   \catcode\@=11 \count@=128
3784   \loop\ifnum\count@<192
3785     \global\uccode\count@=0 \global\lccode\count@=0
3786     \global\catcode\count@=12 \global\sffcode\count@=1000
3787     \advance\count@ by 1 \repeat
3788   % Other:
3789   \def\O ##1 {%
3790     \global\uccode"##1=0 \global\lccode"##1=0
3791     \global\catcode"##1=12 \global\sffcode"##1=1000 }%
3792   % Letter:
3793   \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3794     \global\uccode"##1="##2
3795     \global\lccode"##1="##3
3796     % Uppercase letters have sffcode=999:
3797     \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
3798   % Letter without case mappings:
3799   \def\l ##1 {\L ##1 ##1 ##1 }%
3800   \l 00AA
3801   \L 00B5 039C 00B5
3802   \l 00BA
3803   \O 00D7
3804   \l 00DF
3805   \O 00F7
3806   \L 00FF 0178 00FF
3807 \endgroup
3808 \input #1\relax
3809 <</Restore Unicode catcodes before loading patterns>>

```


Some more common code.

```

3810 <<{*Footnote changes}>> ≡
3811 \bbl@trace{Bidi footnotes}
3812 \ifx\bbl@beforeforeign\leavevmode
3813   \def\bbl@footnote#1#2#3{%
3814     \@ifnextchar[%
3815       {\bbl@footnote@o{#1}{#2}{#3}}%
3816       {\bbl@footnote@x{#1}{#2}{#3}}}
3817   \def\bbl@footnote@x#1#2#3#4{%
3818     \bgroup
3819     \select@language@x{\bbl@main@language}%
3820     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3821     \egroup}
3822   \def\bbl@footnote@o#1#2#3[#4]#5{%
3823     \bgroup
3824     \select@language@x{\bbl@main@language}%
3825     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3826     \egroup}
3827   \def\bbl@footnotetext#1#2#3{%
3828     \@ifnextchar[%
3829       {\bbl@footnotetext@o{#1}{#2}{#3}}%
3830       {\bbl@footnotetext@x{#1}{#2}{#3}}}
3831   \def\bbl@footnotetext@x#1#2#3#4{%
3832     \bgroup
3833     \select@language@x{\bbl@main@language}%
3834     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3835     \egroup}
3836   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3837     \bgroup
3838     \select@language@x{\bbl@main@language}%
3839     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3840     \egroup}
3841   \def\BabelFootnote#1#2#3#4{%
3842     \ifx\bbl@fn@footnote\undefined
3843       \let\bbl@fn@footnote\footnote
3844     \fi
3845     \ifx\bbl@fn@footnotetext\undefined
3846       \let\bbl@fn@footnotetext\footnotetext
3847     \fi
3848     \bbl@ifblank{#2}%
3849     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3850      \@namedef{\bbl@stripslash#1text}%
3851       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3852     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3853      \@namedef{\bbl@stripslash#1text}%
3854       {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3855   \fi
3856 <</Footnote changes>>

```

Now, the code.

```

3857 <{*xetex}>
3858 \def\BabelStringsDefault{unicode}
3859 \let\xebbl@stop\relax
3860 \AddBabelHook{xetex}{encodedcommands}{%
3861   \def\bbl@tempa{#1}%
3862   \ifx\bbl@tempa\empty
3863     \XeTeXinputencoding"bytes"%
3864   \else
3865     \XeTeXinputencoding"#1"%

```

```

3866 \fi
3867 \def\xebbl@stop{\XeTeXinputencoding"utf8"}
3868 \AddBabelHook{xetex}{stopcommands}{%
3869 \xebbl@stop
3870 \let\xebbl@stop\relax}
3871 \def\bbl@intraspace#1 #2 #3\@@{%
3872 \bbl@csarg\gdef{\xeisp@bbl@cs{sbc@}\languagename}}%
3873 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3874 \def\bbl@intrapenalty#1\@@{%
3875 \bbl@csarg\gdef{\xeipn@bbl@cs{sbc@}\languagename}}%
3876 {\XeTeXlinebreakpenalty #1\relax}}
3877 \def\bbl@provide@intraspace{%
3878 \bbl@xin@{\bbl@cs{sbc@}\languagename}}{Thai,Lao,Khmr}%
3879 \ifin@ % sea (currently ckj not handled)
3880 \bbl@ifunset{bbl@intsp@}\languagename}{%
3881 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
3882 \ifx\bbl@KVP@intraspace\@nil
3883 \bbl@exp{%
3884 \bbl@intraspace\bbl@cs{intsp@\languagename}\@@}%
3885 \fi
3886 \ifx\bbl@KVP@intrapenalty\@nil
3887 \bbl@intrapenalty0\@@
3888 \fi
3889 \fi
3890 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
3891 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
3892 \fi
3893 \ifx\bbl@KVP@intrapenalty\@nil\else
3894 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
3895 \fi
3896 \ifx\bbl@ispace@size\@undefined
3897 \AtBeginDocument{%
3898 \expandafter\bbl@add
3899 \csname selectfont\endcsname{\bbl@ispace@size}}%
3900 \def\bbl@ispace@size{\bbl@cs{\xeisp@bbl@cs{sbc@}\languagename}}}%
3901 \fi}%
3902 \fi}
3903 \AddBabelHook{xetex}{loadkernel}{%
3904 <<Restore Unicode catcodes before loading patterns>>}
3905 \ifx\DisableBabelHook\@undefined\endinput\fi
3906 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3907 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
3908 \DisableBabelHook{babel-fontspec}
3909 <<Font selection>>
3910 \input txtbabel.def
3911 </xetex>

```

15.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titles, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the \TeX 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.

```

3912 (*texxet)
3913 \providecommand\bbl@provide@intraspace{}
3914 \bbl@trace{Redefinitions for bidi layout}
3915 \def\bbl@sspre@caption{%
3916   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}}
3917 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
3918 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3919 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3920 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3921   \def\@hangfrom#1{%
3922     \setbox\@tempboxa\hbox{#1}%
3923     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3924     \noindent\box\@tempboxa}
3925 \def\raggedright{%
3926   \let\@centercr
3927   \bbl@startskip\z@skip
3928   \@rightskip\@flushglue
3929   \bbl@endskip\@rightskip
3930   \parindent\z@
3931   \parfillskip\bbl@startskip}
3932 \def\raggedleft{%
3933   \let\@centercr
3934   \bbl@startskip\@flushglue
3935   \bbl@endskip\z@skip
3936   \parindent\z@
3937   \parfillskip\bbl@endskip}
3938 \fi
3939 \IfBabelLayout{lists}
3940   {\bbl@sreplace\list
3941     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
3942     \def\bbl@listleftmargin{%
3943       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
3944     \ifcase\bbl@engine
3945       \def\labelenumii{}\theenumii{}\% pdftex doesn't reverse ()
3946       \def\p@enumiii{\p@enumii}\theenumii{}\%
3947     \fi
3948     \bbl@sreplace\@verbatim
3949       {\leftskip\@totalleftmargin}%
3950       {\bbl@startskip\textwidth
3951         \advance\bbl@startskip-\linewidth}%
3952     \bbl@sreplace\@verbatim
3953       {\rightskip\z@skip}%
3954       {\bbl@endskip\z@skip}}%
3955   {}
3956 \IfBabelLayout{contents}
3957   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
3958     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
3959   {}
3960 \IfBabelLayout{columns}
3961   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
3962     \def\bbl@outputbox#1{%
3963       \hb@xt@\textwidth{%
3964         \hskip\columnwidth
3965         \hfil
3966         {\normalcolor\vrule \@width\columnseprule}%
3967         \hfil
3968         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3969         \hskip-\textwidth
3970         \hb@xt@\columnwidth{\box\@outputbox \hss}%

```

```

3971      \hskip\columnsep
3972      \hskip\columnwidth}}}%
3973  {}
3974  <<Footnote changes>>
3975  \IfBabelLayout{footnotes}%
3976    {\BabelFootnote\footnote\language\language\language}%
3977    \BabelFootnote\localfootnote\language\language}%
3978    \BabelFootnote\mainfootnote\language\language}%
3979  {}

```

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.

```

3980 \IfBabelLayout{counters}%
3981  {\let\bbl@latin@arabic=\@arabic
3982   \def\@arabic#1{\babelsublr{\bbl@latin@arabic#1}}}%
3983   \let\bbl@asci@roman=\@roman
3984   \def\@roman#1{\babelsublr{\ens@ureascii{\bbl@asci@roman#1}}}%
3985   \let\bbl@asci@Roman=\@Roman
3986   \def\@Roman#1{\babelsublr{\ens@ureascii{\bbl@asci@Roman#1}}}%
3987 </texxet>

```

15.3 LuaTeX

The new 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). 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.

```

3988 <*\luatex>
3989 \ifx\AddBabelHook\undefined

```

```

3990 \bbl@trace{Read language.dat}
3991 \ifx\bbl@readstream\@undefined
3992   \csname newread\endcsname\bbl@readstream
3993 \fi
3994 \begingroup
3995   \toks@{}
3996   \count@\z@ % 0=start, 1=0th, 2=normal
3997   \def\bbl@process@line#1#2 #3 #4 {%
3998     \ifx=#1%
3999       \bbl@process@synonym{#2}%
4000     \else
4001       \bbl@process@language{#1#2}{#3}{#4}%
4002     \fi
4003     \ignorespaces}
4004   \def\bbl@manylang{%
4005     \ifnum\bbl@last>\@ne
4006       \bbl@info{Non-standard hyphenation setup}%
4007     \fi
4008     \let\bbl@manylang\relax}
4009   \def\bbl@process@language#1#2#3{%
4010     \ifcase\count@
4011       \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4012     \or
4013       \count@\tw@
4014     \fi
4015     \ifnum\count@=\tw@
4016       \expandafter\addlanguage\csname l@#1\endcsname
4017       \language\allocationnumber
4018       \chardef\bbl@last\allocationnumber
4019       \bbl@manylang
4020       \let\bbl@elt\relax
4021       \xdef\bbl@languages{%
4022         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4023     \fi
4024     \the\toks@
4025     \toks@{}}
4026   \def\bbl@process@synonym@aux#1#2{%
4027     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4028     \let\bbl@elt\relax
4029     \xdef\bbl@languages{%
4030       \bbl@languages\bbl@elt{#1}{#2}{}}}%
4031   \def\bbl@process@synonym#1{%
4032     \ifcase\count@
4033       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4034     \or
4035       \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4036     \else
4037       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4038     \fi}
4039   \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4040     \chardef\l@english\z@
4041     \chardef\l@USenglish\z@
4042     \chardef\bbl@last\z@
4043     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4044     \gdef\bbl@languages{%
4045       \bbl@elt{english}{0}{hyphen.tex}}%
4046     \bbl@elt{USenglish}{0}{}{}
4047   \else
4048     \global\let\bbl@languages@format\bbl@languages

```

```

4049 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4050 \ifnum#2>\z@\else
4051 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4052 \fi}%
4053 \xdef\bbl@languages{\bbl@languages}%
4054 \fi
4055 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4056 \bbl@languages
4057 \openin\bbl@readstream=language.dat
4058 \ifeof\bbl@readstream
4059 \bbl@warning{I couldn't find language.dat. No additional\\%
4060 patterns loaded. Reported}%
4061 \else
4062 \loop
4063 \endlinechar\m@ne
4064 \read\bbl@readstream to \bbl@line
4065 \endlinechar\^^M
4066 \if T\ifeof\bbl@readstream F\fi T\relax
4067 \ifx\bbl@line\@empty\else
4068 \edef\bbl@line{\bbl@line\space\space\space}%
4069 \expandafter\bbl@process@line\bbl@line\relax
4070 \fi
4071 \repeat
4072 \fi
4073 \endgroup
4074 \bbl@trace{Macros for reading patterns files}
4075 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4076 % TODO - Harcoded value:
4077 \ifx\babelcatcodetablenum\undefined
4078 \def\babelcatcodetablenum{5211}
4079 \fi
4080 \def\bbl@luapatterns#1#2{%
4081 \bbl@get@enc#1::\@@@
4082 \setbox\z@\hbox\bgroup
4083 \begingroup
4084 \ifx\catcodetable\undefined
4085 \let\savecatcodetable\luatexsavecatcodetable
4086 \let\initcatcodetable\luatexinitcatcodetable
4087 \let\catcodetable\luatexcatcodetable
4088 \fi
4089 \savecatcodetable\babelcatcodetablenum\relax
4090 \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
4091 \catcodetable\numexpr\babelcatcodetablenum+1\relax
4092 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4093 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4094 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4095 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4096 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4097 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4098 \input #1\relax
4099 \catcodetable\babelcatcodetablenum\relax
4100 \endgroup
4101 \def\bbl@tempa{#2}%
4102 \ifx\bbl@tempa\@empty\else
4103 \input #2\relax
4104 \fi
4105 \egroup}%
4106 \def\bbl@patterns@lua#1{%
4107 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax

```

```

4108 \csname l@#1\endcsname
4109 \edef\bbl@tempa{#1}%
4110 \else
4111 \csname l@#1:\f@encoding\endcsname
4112 \edef\bbl@tempa{#1:\f@encoding}%
4113 \fi\relax
4114 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4115 \@ifundefined{bbl@hyphendata@the\language}%
4116 {\def\bbl@elt##1##2##3##4{%
4117 \ifnum##2=\csname l\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4118 \def\bbl@tempb{##3}%
4119 \ifx\bbl@tempb\empty\else % if not a synonymous
4120 \def\bbl@tempc{##3}{##4}%
4121 \fi
4122 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4123 \fi}%
4124 \bbl@languages
4125 \@ifundefined{bbl@hyphendata@the\language}%
4126 {\bbl@info{No hyphenation patterns were set for\%
4127 language '\bbl@tempa'. Reported}}%
4128 {\expandafter\expandafter\expandafter\bbl@luapatterns
4129 \csname bbl@hyphendata@the\language\endcsname}}}%
4130 \endinput\fi
4131 \begingroup
4132 \catcode`\%=12
4133 \catcode`\'=12
4134 \catcode`\ "=12
4135 \catcode`\:=12
4136 \directlua{
4137 Babel = Babel or {}
4138 function Babel.bytes(line)
4139 return line:gsub("(.)",
4140 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4141 end
4142 function Babel.begin_process_input()
4143 if luatexbase and luatexbase.add_to_callback then
4144 luatexbase.add_to_callback('process_input_buffer',
4145 Babel.bytes, 'Babel.bytes')
4146 else
4147 Babel.callback = callback.find('process_input_buffer')
4148 callback.register('process_input_buffer', Babel.bytes)
4149 end
4150 end
4151 function Babel.end_process_input ()
4152 if luatexbase and luatexbase.remove_from_callback then
4153 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4154 else
4155 callback.register('process_input_buffer', Babel.callback)
4156 end
4157 end
4158 function Babel.addpatterns(pp, lg)
4159 local lg = lang.new(lg)
4160 local pats = lang.patterns(lg) or ''
4161 lang.clear_patterns(lg)
4162 for p in pp:gmatch('[^%s]+') do
4163 ss = ''
4164 for i in string.utfcharacters(p:gsub('%d', '')) do
4165 ss = ss .. '%d?' .. i
4166 end

```

```

4167     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4168     ss = ss:gsub('%.%%d%?$', '%%.')
4169     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4170     if n == 0 then
4171         tex.sprint(
4172             [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4173             .. p .. [[]])
4174         pats = pats .. ' ' .. p
4175     else
4176         tex.sprint(
4177             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4178             .. p .. [[]])
4179     end
4180 end
4181 lang.patterns(lg, pats)
4182 end
4183 }
4184 \endgroup
4185 \ifx\newattribute\@undefined\else
4186   \newattribute\bbl@attr@locale
4187   \AddBabelHook{luatex}{beforeextras}{%
4188     \setattribute\bbl@attr@locale\localeid}
4189 \fi
4190 \def\BabelStringsDefault{unicode}
4191 \let\luabbl@stop\relax
4192 \AddBabelHook{luatex}{encodedcommands}{%
4193   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4194   \ifx\bbl@tempa\bbl@tempb\else
4195     \directlua{Babel.begin_process_input()}%
4196     \def\luabbl@stop{%
4197       \directlua{Babel.end_process_input()}}%
4198   \fi}%
4199 \AddBabelHook{luatex}{stopcommands}{%
4200   \luabbl@stop
4201   \let\luabbl@stop\relax}
4202 \AddBabelHook{luatex}{patterns}{%
4203   \@ifundefined{bbl@hyphendata@the\language}%
4204     {\def\bbl@elt##1##2##3##4{%
4205       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4206       \def\bbl@tempb{##3}%
4207       \ifx\bbl@tempb@empty\else % if not a synonymous
4208         \def\bbl@tempc{##3}{##4}%
4209       \fi
4210       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4211     \fi}%
4212   \bbl@languages
4213   \@ifundefined{bbl@hyphendata@the\language}%
4214     {\bbl@info{No hyphenation patterns were set for\%
4215       language '#2'. Reported}}%
4216     {\expandafter\expandafter\expandafter\bbl@luapatterns
4217       \csname bbl@hyphendata@the\language\endcsname}}}%
4218   \@ifundefined{bbl@patterns@}{}%
4219   \begingroup
4220     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
4221   \ifin\else
4222     \ifx\bbl@patterns@empty\else
4223       \directlua{ Babel.addpatterns(
4224         [[\bbl@patterns@]], \number\language) }%
4225     \fi

```



```

4226 \ifundefined{bbl@patterns@#1}%
4227 \empty
4228 {\directlua{ Babel.addpatterns(
4229 [[\space\csname bbl@patterns@#1\endcsname]],
4230 \number\language) }}%
4231 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4232 \fi
4233 \endgroup}}
4234 \AddBabelHook{luatex}{everylanguage}{%
4235 \def\process@language##1##2##3{%
4236 \def\process@line####1####2 ####3 ####4 {}}}
4237 \AddBabelHook{luatex}{loadpatterns}{%
4238 \input #1\relax
4239 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4240 {{#1}}}}
4241 \AddBabelHook{luatex}{loadexceptions}{%
4242 \input #1\relax
4243 \def\bbl@tempb##1##2{{#1}{#1}}%
4244 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4245 {\expandafter\expandafter\expandafter\bbl@tempb
4246 \csname bbl@hyphendata@the\language\endcsname}}

```

`\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.

```

4247 \@onlypreamble\babelpatterns
4248 \AtEndOfPackage{%
4249 \newcommand\babelpatterns[2][\empty]{%
4250 \ifx\bbl@patterns@relax
4251 \let\bbl@patterns@\empty
4252 \fi
4253 \ifx\bbl@pttnlist\empty\else
4254 \bbl@warning{%
4255 You must not intermingle \string\selectlanguage\space and\%
4256 \string\babelpatterns\space or some patterns will not\%
4257 be taken into account. Reported}%
4258 \fi
4259 \ifx\empty#1%
4260 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4261 \else
4262 \edef\bbl@tempb{\zap@space#1 \empty}%
4263 \bbl@for\bbl@tempa\bbl@tempb{%
4264 \bbl@fixname\bbl@tempa
4265 \bbl@iflanguage\bbl@tempa{%
4266 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4267 \@ifundefined{bbl@patterns@\bbl@tempa}%
4268 \empty
4269 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4270 #2}}}%
4271 \fi}}

```

15.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

In progress. 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.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4272 \directlua{
4273   Babel = Babel or {}
4274   Babel.linebreaking = Babel.linebreaking or {}
4275   Babel.linebreaking.before = {}
4276   Babel.linebreaking.after = {}
4277   Babel.locale = {} % Free to use, indexed with \localeid
4278   function Babel.linebreaking.add_before(func)
4279     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4280     table.insert(Babel.linebreaking.before, func)
4281   end
4282   function Babel.linebreaking.add_after(func)
4283     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4284     table.insert(Babel.linebreaking.after, func)
4285   end
4286 }
4287 \def\bbl@intraspace#1 #2 #3\@{
4288   \directlua{
4289     Babel = Babel or {}
4290     Babel.intraspaces = Babel.intraspaces or {}
4291     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
4292       {b = #1, p = #2, m = #3}
4293     Babel.locale_props[\the\localeid].intraspace = %
4294       {b = #1, p = #2, m = #3}
4295   }}
4296 \def\bbl@intrapenalty#1\@{
4297   \directlua{
4298     Babel = Babel or {}
4299     Babel.intrapenalties = Babel.intrapenalties or {}
4300     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4301     Babel.locale_props[\the\localeid].intrapenalty = #1
4302   }}
4303 \begingroup
4304 \catcode`\%=12
4305 \catcode`\^=14
4306 \catcode`\'=12
4307 \catcode`\~=12
4308 \gdef\bbl@seaintraspace{
4309   \let\bbl@seaintraspace\relax
4310   \directlua{
4311     Babel = Babel or {}
4312     Babel.sea_enabled = true
4313     Babel.sea_ranges = Babel.sea_ranges or {}
4314     function Babel.set_chranges (script, chrng)
4315       local c = 0
4316       for s, e in string.gmatch(chrng..' ', '(.-%).%.(.-)%s') do
4317         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4318         c = c + 1
4319       end
4320     end
4321     function Babel.sea_disc_to_space (head)
4322       local sea_ranges = Babel.sea_ranges
4323       local last_char = nil
4324       local quad = 655360 ^^ 10 pt = 655360 = 10 * 65536
4325       for item in node.traverse(head) do
4326         local i = item.id
4327         if i == node.id'glyph' then
4328           last_char = item

```

```

4329         elseif i == 7 and item.subtype == 3 and last_char
4330             and last_char.char > 0x0C99 then
4331             quad = font.getfont(last_char.font).size
4332             for lg, rg in pairs(sea_ranges) do
4333                 if last_char.char > rg[1] and last_char.char < rg[2] then
4334                     lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
4335                     local intraspace = Babel.intraspaces[lg]
4336                     local intrapenalty = Babel.intrapenalties[lg]
4337                     local n
4338                     if intrapenalty ~= 0 then
4339                         n = node.new(14, 0) ^^ penalty
4340                         n.penalty = intrapenalty
4341                         node.insert_before(head, item, n)
4342                     end
4343                     n = node.new(12, 13) ^^ (glue, spaceskip)
4344                     node.setglue(n, intraspace.b * quad,
4345                                     intraspace.p * quad,
4346                                     intraspace.m * quad)
4347                     node.insert_before(head, item, n)
4348                     node.remove(head, item)
4349                 end
4350             end
4351         end
4352     end
4353 end
4354 }^^
4355 \bbl@luahyphenate}
4356 \catcode`\%=14
4357 \gdef\bbl@cjkintraspaces{%
4358     \let\bbl@cjkintraspaces\relax
4359     \directlua{
4360         Babel = Babel or {}
4361         require'babel-data-cjk.lua'
4362         Babel.cjk_enabled = true
4363         function Babel.cjk_linebreak(head)
4364             local GLYPH = node.id'glyph'
4365             local last_char = nil
4366             local quad = 655360 % 10 pt = 655360 = 10 * 65536
4367             local last_class = nil
4368             local last_lang = nil
4369
4370             for item in node.traverse(head) do
4371                 if item.id == GLYPH then
4372
4373                     local lang = item.lang
4374
4375                     local LOCALE = node.get_attribute(item,
4376                                                         luatexbase.registernumber'bbl@attr@locale')
4377                     local props = Babel.locale_props[LOCALE]
4378
4379                     local class = Babel.cjk_class[item.char].c
4380
4381                     if class == 'cp' then class = 'cl' end % )] as CL
4382                     if class == 'id' then class = 'I' end
4383
4384                     local br = 0
4385                     if class and last_class and Babel.cjk_breaks[last_class][class] then
4386                         br = Babel.cjk_breaks[last_class][class]
4387                     end

```

```

4388
4389         if br == 1 and props.linebreak == 'c' and
4390             lang ~= \the\l@nohyphenation\space and
4391             last_lang ~= \the\l@nohyphenation then
4392             local intrapenalty = props.intrapenalty
4393             if intrapenalty ~= 0 then
4394                 local n = node.new(14, 0)    % penalty
4395                 n.penalty = intrapenalty
4396                 node.insert_before(head, item, n)
4397             end
4398             local intraspace = props.intraspace
4399             local n = node.new(12, 13)    % (glue, spaceskip)
4400             node.setglue(n, intraspace.b * quad,
4401                          intraspace.p * quad,
4402                          intraspace.m * quad)
4403             node.insert_before(head, item, n)
4404         end
4405
4406         quad = font.getfont(item.font).size
4407         last_class = class
4408         last_lang = lang
4409     else % if penalty, glue or anything else
4410         last_class = nil
4411     end
4412 end
4413 lang.hyphenate(head)
4414 end
4415 }%
4416 \bbl@luahyphenate}
4417 \gdef\bbl@luahyphenate{%
4418 \let\bbl@luahyphenate\relax
4419 \directlua{
4420     luatexbase.add_to_callback('hyphenate',
4421     function (head, tail)
4422         if Babel.linebreaking.before then
4423             for k, func in ipairs(Babel.linebreaking.before) do
4424                 func(head)
4425             end
4426         end
4427         if Babel.cjk_enabled then
4428             Babel.cjk_linebreak(head)
4429         end
4430         lang.hyphenate(head)
4431         if Babel.linebreaking.after then
4432             for k, func in ipairs(Babel.linebreaking.after) do
4433                 func(head)
4434             end
4435         end
4436         if Babel.sea_enabled then
4437             Babel.sea_disc_to_space(head)
4438         end
4439     end,
4440     'Babel.hyphenate')
4441 }
4442 }
4443 \endgroup
4444 \def\bbl@provide@intraspace{%
4445 \bbl@ifunset{\bbl@intsp@languagename}{}%
4446 {\expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else

```

```

4447 \bbl@xin@{\bbl@cs{lnbrk@\language}}{c}%
4448 \ifin@ % cjk
4449 \bbl@cjkintraspacespace
4450 \directlua{
4451   Babel = Babel or {}
4452   Babel.locale_props = Babel.locale_props or {}
4453   Babel.locale_props[\the\localeid].linebreak = 'c'
4454 }%
4455 \bbl@exp{\bbl@intraspacespace\bbl@cs{intsp@\language}}{\bbl@}%
4456 \ifx\bbl@KVP@intrapenalty\@nil
4457 \bbl@intrapenalty0\@@
4458 \fi
4459 \else % sea
4460 \bbl@seaintraspacespace
4461 \bbl@exp{\bbl@intraspacespace\bbl@cs{intsp@\language}}{\bbl@}%
4462 \directlua{
4463   Babel = Babel or {}
4464   Babel.sea_ranges = Babel.sea_ranges or {}
4465   Babel.set_chranges('\bbl@cs{sbcpr@\language}',
4466                     '\bbl@cs{chrng@\language}')
4467 }%
4468 \ifx\bbl@KVP@intrapenalty\@nil
4469 \bbl@intrapenalty0\@@
4470 \fi
4471 \fi
4472 \fi
4473 \ifx\bbl@KVP@intrapenalty\@nil\else
4474 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4475 \fi}}

```

15.5 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.

```

4476 \AddBabelHook{luatex}{loadkernel}{%
4477 <<Restore Unicode catcodes before loading patterns>>}
4478 \ifx\DisableBabelHook\@undefined\endinput\fi
4479 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4480 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4481 \DisableBabelHook{babel-fontspec}
4482 <<Font selection>>

```

15.6 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.

```

4483 \directlua{
4484 Babel.script_blocks = {
4485   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4486             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4487   ['Armn'] = {{0x0530, 0x058F}},
4488   ['Beng'] = {{0x0980, 0x09FF}},
4489   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
4490   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4491             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4492   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4493   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4494             {0xAB00, 0xAB2F}},
4495   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4496   ['Grek'] = {{0x0370, 0x03FF}, {0x1F00, 0x1FFF}},
4497   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4498             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4499             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4500             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4501             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4502             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4503   ['Hebr'] = {{0x0590, 0x05FF}},
4504   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4505             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4506   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4507   ['Knda'] = {{0x0C80, 0x0CFF}},
4508   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4509             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4510             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4511   ['Laoo'] = {{0x0E80, 0x0EFF}},
4512   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4513             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4514             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4515   ['Mahj'] = {{0x11150, 0x1117F}},
4516   ['Mlym'] = {{0x0D00, 0x0D7F}},
4517   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4518   ['Orya'] = {{0x0B00, 0x0B7F}},
4519   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4520   ['Taml'] = {{0x0B80, 0x0BFF}},
4521   ['Telu'] = {{0x0C00, 0x0C7F}},
4522   ['Tfng'] = {{0x2D30, 0x2D7F}},
4523   ['Thai'] = {{0x0E00, 0x0E7F}},
4524   ['Tibt'] = {{0x0F00, 0x0FFF}},
4525   ['Vaii'] = {{0xA500, 0xA63F}},
4526   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4527 }
4528
4529 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4530 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4531
4532 function Babel.locale_map(head)
4533   if not Babel.locale_mapped then return head end
4534
4535   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
4536   local GLYPH = node.id('glyph')
4537   local inmath = false

```

```

4538 for item in node.traverse(head) do
4539   local toloc
4540   if not inmath and item.id == GLYPH then
4541     % Optimization: build a table with the chars found
4542     if Babel.chr_to_loc[item.char] then
4543       toloc = Babel.chr_to_loc[item.char]
4544     else
4545       for lc, maps in pairs(Babel.loc_to_scr) do
4546         for _, rg in pairs(maps) do
4547           if item.char >= rg[1] and item.char <= rg[2] then
4548             Babel.chr_to_loc[item.char] = lc
4549             toloc = lc
4550             break
4551           end
4552         end
4553       end
4554     end
4555     % Now, take action
4556     if toloc and toloc > -1 then
4557       if Babel.locale_props[toloc].lg then
4558         item.lang = Babel.locale_props[toloc].lg
4559         node.set_attribute(item, LOCALE, toloc)
4560       end
4561       if Babel.locale_props[toloc]['/'..item.font] then
4562         item.font = Babel.locale_props[toloc]['/'..item.font]
4563       end
4564     end
4565   elseif not inmath and item.id == 7 then
4566     item.replace = item.replace and Babel.locale_map(item.replace)
4567     item.pre      = item.pre and Babel.locale_map(item.pre)
4568     item.post     = item.post and Babel.locale_map(item.post)
4569   elseif item.id == node.id'math' then
4570     inmath = (item.subtype == 0)
4571   end
4572 end
4573 return head
4574 end
4575 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

4576 \newcommand\babelcharproperty[1]{%
4577   \count@=#1\relax
4578   \ifvmode
4579     \expandafter\bbl@chprop
4580   \else
4581     \bbl@error{\string\babelcharproperty\space can be used only in\%
4582               vertical mode (preamble or between paragraphs)}%
4583     {See the manual for futher info}%
4584   \fi}
4585 \newcommand\bbl@chprop[3][\the\count@]{%
4586   \@tempcnta=#1\relax
4587   \bbl@ifunset{\bbl@chprop@#2}%
4588   {\bbl@error{No property named '#2'. Allowed values are\%
4589             direction (bc), mirror (bmg), and linebreak (lb)}%
4590    {See the manual for futher info}}%
4591   {}%
4592   \loop
4593     \@nameuse{\bbl@chprop@#2}{#3}%

```

```

4594 \ifnum\count@<\@tempcnta
4595 \advance\count@\@ne
4596 \repeat}
4597 \def\bbl@chprop@direction#1{%
4598 \directlua{
4599   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4600   Babel.characters[\the\count@]['d'] = '#1'
4601 }}
4602 \let\bbl@chprop@bc\bbl@chprop@direction
4603 \def\bbl@chprop@mirror#1{%
4604 \directlua{
4605   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4606   Babel.characters[\the\count@]['m'] = '\number#1'
4607 }}
4608 \let\bbl@chprop@bmg\bbl@chprop@mirror
4609 \def\bbl@chprop@linebreak#1{%
4610 \directlua{
4611   Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4612   Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4613 }}
4614 \let\bbl@chprop@lb\bbl@chprop@linebreak
4615 \def\bbl@chprop@locale#1{%
4616 \directlua{
4617   Babel.chr_to_loc = Babel.chr_to_loc or {}
4618   Babel.chr_to_loc[\the\count@] =
4619     \bbl@ifblank{#1}{-1000}{\the\@nameuse{bbl@id@#1}}\space
4620 }}

```

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 `tex.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.

```

4621 \begingroup
4622 \catcode`\#=12
4623 \catcode`\%=12
4624 \catcode`\&=14
4625 \directlua{
4626   Babel.linebreaking.replacements = {}
4627
4628   function Babel.str_to_nodes(fn, matches, base)
4629     local n, head, last
4630     if fn == nil then return nil end
4631     for s in string.utfvalues(fn(matches)) do
4632       if base.id == 7 then
4633         base = base.replace
4634       end
4635       n = node.copy(base)
4636       n.char = s
4637       if not head then

```



```

4638         head = n
4639     else
4640         last.next = n
4641     end
4642     last = n
4643 end
4644 return head
4645 end
4646
4647 function Babel.fetch_word(head, funct)
4648     local word_string = ''
4649     local word_nodes = {}
4650     local lang
4651     local item = head
4652
4653     while item do
4654
4655         if item.id == 29
4656             and not(item.char == 124) %% ie, not |
4657             and not(item.char == 61) %% ie, not =
4658             and (item.lang == lang or lang == nil) then
4659             lang = lang or item.lang
4660             word_string = word_string .. unicode.utf8.char(item.char)
4661             word_nodes[#word_nodes+1] = item
4662
4663         elseif item.id == 7 and item.subtype == 2 then
4664             word_string = word_string .. '='
4665             word_nodes[#word_nodes+1] = item
4666
4667         elseif item.id == 7 and item.subtype == 3 then
4668             word_string = word_string .. '|'
4669             word_nodes[#word_nodes+1] = item
4670
4671         elseif word_string == '' then
4672             %% pass
4673
4674         else
4675             return word_string, word_nodes, item, lang
4676         end
4677
4678         item = item.next
4679     end
4680 end
4681
4682 function Babel.post_hyphenate_replace(head)
4683     local u = unicode.utf8
4684     local lbkr = Babel.linebreaking.replacements
4685     local word_head = head
4686
4687     while true do
4688         local w, wn, nw, lang = Babel.fetch_word(word_head)
4689         if not lang then return head end
4690
4691         if not lbkr[lang] then
4692             break
4693         end
4694
4695         for k=1, #lbkr[lang] do
4696             local p = lbkr[lang][k].pattern

```

```

4697     local r = lbkr[lang][k].replace
4698
4699     while true do
4700         local matches = { u.match(w, p) }
4701         if #matches < 2 then break end
4702
4703         local first = table.remove(matches, 1)
4704         local last = table.remove(matches, #matches)
4705
4706         %% Fix offsets, from bytes to unicode.
4707         first = u.len(w:sub(1, first-1)) + 1
4708         last = u.len(w:sub(1, last-1))
4709
4710         local new %% used when inserting and removing nodes
4711         local changed = 0
4712
4713         %% This loop traverses the replace list and takes the
4714         %% corresponding actions
4715         for q = first, last do
4716             local crep = r[q-first+1]
4717             local char_node = wn[q]
4718             local char_base = char_node
4719
4720             if crep and crep.data then
4721                 char_base = wn[crep.data+first-1]
4722             end
4723
4724             if crep == {} then
4725                 break
4726             elseif crep == nil then
4727                 changed = changed + 1
4728                 node.remove(head, char_node)
4729             elseif crep and (crep.pre or crep.no or crep.post) then
4730                 changed = changed + 1
4731                 d = node.new(7, 0) %% (disc, discretionary)
4732                 d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
4733                 d.post = Babel.str_to_nodes(crep.post, matches, char_base)
4734                 d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
4735                 d.attr = char_base.attr
4736                 if crep.pre == nil then %% TeXbook p96
4737                     d.penalty = crep.penalty or tex.hyphenpenalty
4738                 else
4739                     d.penalty = crep.penalty or tex.exhyphenpenalty
4740                 end
4741                 head, new = node.insert_before(head, char_node, d)
4742                 node.remove(head, char_node)
4743                 if q == 1 then
4744                     word_head = new
4745                 end
4746             elseif crep and crep.string then
4747                 changed = changed + 1
4748                 local str = crep.string(matches)
4749                 if str == '' then
4750                     if q == 1 then
4751                         word_head = char_node.next
4752                     end
4753                     head, new = node.remove(head, char_node)
4754                 elseif char_node.id == 29 and u.len(str) == 1 then
4755                     char_node.char = string.utfvalue(str)

```

```

4756         else
4757             local n
4758             for s in string.utfvalues(str) do
4759                 if char_node.id == 7 then
4760                     log('Automatic hyphens cannot be replaced, just removed.')
4761                 else
4762                     n = node.copy(char_base)
4763                     end
4764                     n.char = s
4765                     if q == 1 then
4766                         head, new = node.insert_before(head, char_node, n)
4767                         word_head = new
4768                     else
4769                         node.insert_before(head, char_node, n)
4770                     end
4771                 end
4772             end
4773             node.remove(head, char_node)
4774             end %% string length
4775             end %% if char and char.string
4776         end %% for char in match
4777         if changed > 20 then
4778             texio.write('Too many changes. Ignoring the rest.')
4779         elseif changed > 0 then
4780             w, wn, nw = Babel.fetch_word(word_head)
4781         end
4782     end
4783     end %% for match
4784     end %% for patterns
4785     word_head = nw
4786 end %% for words
4787 return head
4788 end
4789
4790 %% The following functions belong to the next macro
4791
4792 %% This table stores capture maps, numbered consecutively
4793 Babel.capture_maps = {}
4794
4795 function Babel.capture_func(key, cap)
4796     local ret = "[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[" .. "]"
4797     ret = ret:gsub('{([0-9])|([^\]|+)|(-)}', Babel.capture_func_map)
4798     ret = ret:gsub("%[%[%]%%%.%", '')
4799     ret = ret:gsub("%.%[%[%]%%%", '')
4800     return key .. [[=function(m) return ]] .. ret .. [[ end]]
4801 end
4802
4803 function Babel.capt_map(from, mapno)
4804     return Babel.capture_maps[mapno][from] or from
4805 end
4806
4807 %% Handle the {n|abc|ABC} syntax in captures
4808 function Babel.capture_func_map(capno, from, to)
4809     local froms = {}
4810     for s in string.utfcharacters(from) do
4811         table.insert(froms, s)
4812     end
4813     local cnt = 1
4814     table.insert(Babel.capture_maps, {})

```

```

4815     local mlen = table.getn(Babel.capture_maps)
4816     for s in string.utfcharacters(to) do
4817         Babel.capture_maps[mlen][from[cnt]] = s
4818         cnt = cnt + 1
4819     end
4820     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
4821         (mlen) .. ").. " .. "["
4822 end
4823
4824 }

```

Now the TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ - becomes `function(m) return m[1]..m[1]..'-' end`, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to $m[1]$. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

4825 \catcode`\#=6
4826 \gdef\babelposthyphenation#1#2#3{&%
4827     \bbl@activateposthyphen
4828     \begingroup
4829     \def\babeltempa{\bbl@add@list\babeltempb}&%
4830     \let\babeltempb\@empty
4831     \bbl@foreach{#3}{&%
4832         \bbl@ifsamestring{##1}{remove}&%
4833         {\bbl@add@list\babeltempb{nil}}&%
4834         {\directlua{
4835             local rep = {[##1]}
4836             rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
4837             rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
4838             rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
4839             rep = rep:gsub( '(string)%s*=%s*([^\s,]*)', Babel.capture_func)
4840             tex.print([[\\string\babeltempa{}}] .. rep .. [[]]})
4841         }}&%
4842     \directlua{
4843         local lbkr = Babel.linebreaking.replacements
4844         local u = unicode.utf8
4845         &% Convert pattern:
4846         local patt = string.gsub([[#2]], '%s', '')
4847         if not u.find(patt, '()', nil, true) then
4848             patt = '()' .. patt .. '()'
4849         end
4850         patt = u.gsub(patt, '{(.)}',
4851             function (n)
4852                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
4853             end)
4854         lbkr[\\the\\csname l@#1\\endcsname] = lbkr[\\the\\csname l@#1\\endcsname] or {}
4855         table.insert(lbkr[\\the\\csname l@#1\\endcsname],
4856             { pattern = patt, replace = { \babeltempb } })
4857     }&%
4858     \endgroup}
4859 \endgroup
4860 \def\bbl@activateposthyphen{%
4861     \let\bbl@activateposthyphen\relax

```

```

4862 \directlua{
4863   Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
4864 }}

```

15.7 Layout

Work in progress.

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`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

4865 \bbl@trace{Redefinitions for bidi layout}
4866 \ifx\@eqnnum\undefined\else
4867   \ifx\bbl@attr@dir\undefined\else
4868     \edef\@eqnnum{%
4869       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
4870       \unexpanded\expandafter{\@eqnnum}}
4871   \fi
4872 \fi
4873 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
4874 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4875   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
4876     \bbl@exp{%
4877       \mathdir\the\bodydir
4878       #1% Once entered in math, set boxes to restore values
4879       \<ifmmode>%
4880         \everyvbox{%
4881           \the\everyvbox
4882           \bodydir\the\bodydir
4883           \mathdir\the\mathdir
4884           \everyhbox{\the\everyhbox}%
4885           \everyvbox{\the\everyvbox}}%
4886         \everyhbox{%
4887           \the\everyhbox
4888           \bodydir\the\bodydir
4889           \mathdir\the\mathdir
4890           \everyhbox{\the\everyhbox}%
4891           \everyvbox{\the\everyvbox}}%
4892       \<fi>}}%
4893   \def\@hangfrom#1{%
4894     \setbox\@tempboxa\hbox{{#1}}%
4895     \hangindent\wd\@tempboxa
4896     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4897       \shapemode\@ne
4898     \fi
4899     \noindent\box\@tempboxa}
4900 \fi
4901 \IfBabelLayout{tabular}
4902   {\let\bbl@OL@tabular\@tabular

```

```

4903 \bbl@replace\@tabular{$}\bbl@nextfake$}%
4904 \let\bbl@NL@tabular\@tabular
4905 \AtBeginDocument{%
4906   \ifx\bbl@NL@tabular\@tabular\else
4907     \bbl@replace\@tabular{$}\bbl@nextfake$}%
4908     \let\bbl@NL@tabular\@tabular
4909   \fi}}
4910 {}
4911 \IfBabelLayout{lists}
4912 {\let\bbl@OL@list\list
4913  \bbl@sreplace\list{\parshape}\bbl@listparshape}%
4914  \let\bbl@NL@list\list
4915  \def\bbl@listparshape#1#2#3{%
4916    \parshape #1 #2 #3 %
4917    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4918      \shapemode\tw@
4919    \fi}}
4920 {}
4921 \IfBabelLayout{graphics}
4922 {\let\bbl@pictresetdir\relax
4923  \def\bbl@pictsetdir{%
4924    \ifcase\bbl@thetextdir
4925      \let\bbl@pictresetdir\relax
4926    \else
4927      \textdir TLT\relax
4928      \def\bbl@pictresetdir{\textdir TRT\relax}%
4929    \fi}%
4930  \let\bbl@OL@picture\@picture
4931  \let\bbl@OL@put\put
4932  \bbl@sreplace\@picture{\hskip-}\bbl@pictsetdir\hskip-}%
4933  \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
4934    \@killglue
4935    \raise#2\unitlength
4936    \hbxt@z@{\kern#1\unitlength\bbl@pictresetdir#3}\hss}}%
4937  \AtBeginDocument
4938    {\ifx\tikz@atbegin@node\undefined\else
4939      \let\bbl@OL@pgfpicture\pgfpicture
4940      \bbl@sreplace\pgfpicture{\pgfpicturetrue}\bbl@pictsetdir\pgfpicturetrue}%
4941      \bbl@add\pgfsys@beginpicture\bbl@pictsetdir}%
4942      \bbl@add\tikz@atbegin@node\bbl@pictresetdir}%
4943    \fi}}
4944 {}

```

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.

```

4945 \IfBabelLayout{counters}%
4946 {\let\bbl@OL@textsuperscript\textsuperscript
4947  \bbl@sreplace\textsuperscript{\m@th}\m@th\mathdir\pagedir}%
4948  \let\bbl@latinarabic=\@arabic
4949  \let\bbl@OL@arabic\@arabic
4950  \def\@arabic#1{\babelsublr\bbl@latinarabic#1}}%
4951  \@ifpackagewith{babel}{bidi=default}%
4952    {\let\bbl@asciroman=\@roman
4953     \let\bbl@OL@roman\@roman
4954     \def\@roman#1{\babelsublr\ensureascii\bbl@asciroman#1}}}%
4955  \let\bbl@asciiRoman=\@Roman
4956  \let\bbl@OL@roman\@Roman
4957  \def\@Roman#1{\babelsublr\ensureascii\bbl@asciiRoman#1}}}%

```

```

4958 \let\bbl@OL@labelenumii\labelenumii
4959 \def\labelenumii{}\theenumii{}%
4960 \let\bbl@OL@p@enumiii\p@enumiii
4961 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
4962 <<Footnote changes>>
4963 \IfBabelLayout{footnotes}%
4964 {\let\bbl@OL@footnote\footnote
4965 \BabelFootnote\footnote\languagename{}\}\}%
4966 \BabelFootnote\localfootnote\languagename{}\}\}%
4967 \BabelFootnote\mainfootnote{}\}\}\}\}
4968 {}

```

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.

```

4969 \IfBabelLayout{extras}%
4970 {\let\bbl@OL@underline\underline
4971 \bbl@sreplace\underline{\$@@@underline}\bbl@nextfake$@@@underline}%
4972 \let\bbl@OL@LaTeX2e\LaTeX2e
4973 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
4974 \if b\expandafter\@car\f@series\@nil\boldmath\fi
4975 \babelsublr}%
4976 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
4977 {}
4978 </luatex>

```

15.8 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.

```

4979 (*basic-r)
4980 Babel = Babel or {}
4981
4982 Babel.bidi_enabled = true
4983
4984 require('babel-data-bidi.lua')
4985
4986 local characters = Babel.characters
4987 local ranges = Babel.ranges
4988
4989 local DIR = node.id("dir")
4990
4991 local function dir_mark(head, from, to, outer)
4992   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
4993   local d = node.new(DIR)
4994   d.dir = '+' .. dir
4995   node.insert_before(head, from, d)
4996   d = node.new(DIR)
4997   d.dir = '-' .. dir
4998   node.insert_after(head, to, d)
4999 end
5000
5001 function Babel.bidi(head, ispar)
5002   local first_n, last_n      -- first and last char with nums
5003   local last_es              -- an auxiliary 'last' used with nums
5004   local first_d, last_d      -- first and last char in L/R block
5005   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):

```

5006   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5007   local strong_lr = (strong == 'l') and 'l' or 'r'
5008   local outer = strong
5009
5010   local new_dir = false
5011   local first_dir = false
5012   local inmath = false
5013
5014   local last_lr
5015
5016   local type_n = ''
5017
5018   for item in node.traverse(head) do
5019
5020     -- three cases: glyph, dir, otherwise

```



```

5021   if item.id == node.id'glyph'
5022     or (item.id == 7 and item.subtype == 2) then
5023
5024     local itemchar
5025     if item.id == 7 and item.subtype == 2 then
5026       itemchar = item.replace.char
5027     else
5028       itemchar = item.char
5029     end
5030     local chardata = characters[itemchar]
5031     dir = chardata and chardata.d or nil
5032     if not dir then
5033       for nn, et in ipairs(ranges) do
5034         if itemchar < et[1] then
5035           break
5036         elseif itemchar <= et[2] then
5037           dir = et[3]
5038           break
5039         end
5040       end
5041     end
5042     dir = dir or 'l'
5043     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).

```

5044   if new_dir then
5045     attr_dir = 0
5046     for at in node.traverse(item.attr) do
5047       if at.number == luatexbase.registernumber'bbl@attr@dir' then
5048         attr_dir = at.value % 3
5049       end
5050     end
5051     if attr_dir == 1 then
5052       strong = 'r'
5053     elseif attr_dir == 2 then
5054       strong = 'al'
5055     else
5056       strong = 'l'
5057     end
5058     strong_lr = (strong == 'l') and 'l' or 'r'
5059     outer = strong_lr
5060     new_dir = false
5061   end
5062
5063   if dir == 'nsm' then dir = strong end -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

5064   dir_real = dir -- We need dir_real to set strong below
5065   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:

```

5066   if strong == 'al' then
5067     if dir == 'en' then dir = 'an' end -- W2
5068     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6

```

```

5069         strong_lr = 'r'                                -- W3
5070     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

5071     elseif item.id == node.id'dir' and not inmath then
5072         new_dir = true
5073         dir = nil
5074     elseif item.id == node.id'math' then
5075         inmath = (item.subtype == 0)
5076     else
5077         dir = nil          -- Not a char
5078     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>.

```

5079     if dir == 'en' or dir == 'an' or dir == 'et' then
5080         if dir ~= 'et' then
5081             type_n = dir
5082         end
5083         first_n = first_n or item
5084         last_n = last_es or item
5085         last_es = nil
5086     elseif dir == 'es' and last_n then -- W3+W6
5087         last_es = item
5088     elseif dir == 'cs' then          -- it's right - do nothing
5089     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5090         if strong_lr == 'r' and type_n ~= '' then
5091             dir_mark(head, first_n, last_n, 'r')
5092         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5093             dir_mark(head, first_n, last_n, 'r')
5094             dir_mark(head, first_d, last_d, outer)
5095             first_d, last_d = nil, nil
5096         elseif strong_lr == 'l' and type_n ~= '' then
5097             last_d = last_n
5098         end
5099         type_n = ''
5100         first_n, last_n = nil, nil
5101     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:

```

5102     if dir == 'l' or dir == 'r' then
5103         if dir ~= outer then
5104             first_d = first_d or item
5105             last_d = item
5106         elseif first_d and dir ~= strong_lr then
5107             dir_mark(head, first_d, last_d, outer)
5108             first_d, last_d = nil, nil
5109         end
5110     end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it’s clearly <r> and <l>, resp’tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn’t hurt, but should not be done.

```

5111   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5112       item.char = characters[item.char] and
5113           characters[item.char].m or item.char
5114   elseif (dir or new_dir) and last_lr ~= item then
5115       local mir = outer .. strong_lr .. (dir or outer)
5116       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5117           for ch in node.traverse(node.next(last_lr)) do
5118               if ch == item then break end
5119               if ch.id == node.id'glyph' and characters[ch.char] then
5120                   ch.char = characters[ch.char].m or ch.char
5121               end
5122           end
5123       end
5124   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).

```

5125   if dir == 'l' or dir == 'r' then
5126       last_lr = item
5127       strong = dir_real          -- Don't search back - best save now
5128       strong_lr = (strong == 'l') and 'l' or 'r'
5129   elseif new_dir then
5130       last_lr = nil
5131   end
5132 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

5133   if last_lr and outer == 'r' then
5134       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5135           if characters[ch.char] then
5136               ch.char = characters[ch.char].m or ch.char
5137           end
5138       end
5139   end
5140   if first_n then
5141       dir_mark(head, first_n, last_n, outer)
5142   end
5143   if first_d then
5144       dir_mark(head, first_d, last_d, outer)
5145   end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

5146   return node.prev(head) or head
5147 end
5148 </basic-r>

```

And here the Lua code for bidi=basic:

```

5149 <(*basic)
5150 Babel = Babel or {}
5151
5152 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5153

```

```

5154 Babel.fontmap = Babel.fontmap or {}
5155 Babel.fontmap[0] = {}      -- l
5156 Babel.fontmap[1] = {}      -- r
5157 Babel.fontmap[2] = {}      -- al/an
5158
5159 Babel.bidi_enabled = true
5160 Babel.mirroring_enabled = true
5161
5162 require('babel-data-bidi.lua')
5163
5164 local characters = Babel.characters
5165 local ranges = Babel.ranges
5166
5167 local DIR = node.id('dir')
5168 local GLYPH = node.id('glyph')
5169
5170 local function insert_implicit(head, state, outer)
5171   local new_state = state
5172   if state.sim and state.eim and state.sim ~= state.eim then
5173     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5174     local d = node.new(DIR)
5175     d.dir = '+' .. dir
5176     node.insert_before(head, state.sim, d)
5177     local d = node.new(DIR)
5178     d.dir = '-' .. dir
5179     node.insert_after(head, state.eim, d)
5180   end
5181   new_state.sim, new_state.eim = nil, nil
5182   return head, new_state
5183 end
5184
5185 local function insert_numeric(head, state)
5186   local new
5187   local new_state = state
5188   if state.san and state.ean and state.san ~= state.ean then
5189     local d = node.new(DIR)
5190     d.dir = '+TLT'
5191     _, new = node.insert_before(head, state.san, d)
5192     if state.san == state.sim then state.sim = new end
5193     local d = node.new(DIR)
5194     d.dir = '-TLT'
5195     _, new = node.insert_after(head, state.ean, d)
5196     if state.ean == state.eim then state.eim = new end
5197   end
5198   new_state.san, new_state.ean = nil, nil
5199   return head, new_state
5200 end
5201
5202 -- TODO - \hbox with an explicit dir can lead to wrong results
5203 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5204 -- was s made to improve the situation, but the problem is the 3-dir
5205 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5206 -- well.
5207
5208 function Babel.bidi(head, ispar, hdir)
5209   local d -- d is used mainly for computations in a loop
5210   local prev_d = ''
5211   local new_d = false
5212

```

```

5213 local nodes = {}
5214 local outer_first = nil
5215 local inmath = false
5216
5217 local glue_d = nil
5218 local glue_i = nil
5219
5220 local has_en = false
5221 local first_et = nil
5222
5223 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5224
5225 local save_outer
5226 local temp = node.get_attribute(head, ATDIR)
5227 if temp then
5228     temp = temp % 3
5229     save_outer = (temp == 0 and 'l') or
5230                  (temp == 1 and 'r') or
5231                  (temp == 2 and 'al')
5232 elseif ispar then -- Or error? Shouldn't happen
5233     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5234 else -- Or error? Shouldn't happen
5235     save_outer = ('TRT' == hdir) and 'r' or 'l'
5236 end
5237 -- when the callback is called, we are just _after_ the box,
5238 -- and the textdir is that of the surrounding text
5239 -- if not ispar and hdir ~= tex.textdir then
5240 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5241 -- end
5242 local outer = save_outer
5243 local last = outer
5244 -- 'al' is only taken into account in the first, current loop
5245 if save_outer == 'al' then save_outer = 'r' end
5246
5247 local fontmap = Babel.fontmap
5248
5249 for item in node.traverse(head) do
5250
5251     -- In what follows, #node is the last (previous) node, because the
5252     -- current one is not added until we start processing the neutrals.
5253
5254     -- three cases: glyph, dir, otherwise
5255     if item.id == GLYPH
5256         or (item.id == 7 and item.subtype == 2) then
5257
5258         local d_font = nil
5259         local item_r
5260         if item.id == 7 and item.subtype == 2 then
5261             item_r = item.replace -- automatic discs have just 1 glyph
5262         else
5263             item_r = item
5264         end
5265         local chardata = characters[item_r.char]
5266         d = chardata and chardata.d or nil
5267         if not d or d == 'nsm' then
5268             for nn, et in ipairs(ranges) do
5269                 if item_r.char < et[1] then
5270                     break
5271                 elseif item_r.char <= et[2] then

```

```

5272         if not d then d = et[3]
5273         elseif d == 'nsm' then d_font = et[3]
5274         end
5275         break
5276     end
5277 end
5278 end
5279 d = d or 'l'
5280
5281 -- A short 'pause' in bidi for mapfont
5282 d_font = d_font or d
5283 d_font = (d_font == 'l' and 0) or
5284           (d_font == 'nsm' and 0) or
5285           (d_font == 'r' and 1) or
5286           (d_font == 'al' and 2) or
5287           (d_font == 'an' and 2) or nil
5288 if d_font and fontmap and fontmap[d_font][item_r.font] then
5289     item_r.font = fontmap[d_font][item_r.font]
5290 end
5291
5292 if new_d then
5293     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5294     if inmath then
5295         attr_d = 0
5296     else
5297         attr_d = node.get_attribute(item, ATDIR)
5298         attr_d = attr_d % 3
5299     end
5300     if attr_d == 1 then
5301         outer_first = 'r'
5302         last = 'r'
5303     elseif attr_d == 2 then
5304         outer_first = 'r'
5305         last = 'al'
5306     else
5307         outer_first = 'l'
5308         last = 'l'
5309     end
5310     outer = last
5311     has_en = false
5312     first_et = nil
5313     new_d = false
5314 end
5315
5316 if glue_d then
5317     if (d == 'l' and 'l' or 'r') ~= glue_d then
5318         table.insert(nodes, {glue_i, 'on', nil})
5319     end
5320     glue_d = nil
5321     glue_i = nil
5322 end
5323
5324 elseif item.id == DIR then
5325     d = nil
5326     new_d = true
5327
5328 elseif item.id == node.id'glue' and item.subtype == 13 then
5329     glue_d = d
5330     glue_i = item

```

```

5331     d = nil
5332
5333     elseif item.id == node.id'math' then
5334         inmath = (item.subtype == 0)
5335
5336     else
5337         d = nil
5338     end
5339
5340     -- AL <= EN/ET/ES      -- W2 + W3 + W6
5341     if last == 'al' and d == 'en' then
5342         d = 'an'          -- W3
5343     elseif last == 'al' and (d == 'et' or d == 'es') then
5344         d = 'on'          -- W6
5345     end
5346
5347     -- EN + CS/ES + EN      -- W4
5348     if d == 'en' and #nodes >= 2 then
5349         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5350             and nodes[#nodes-1][2] == 'en' then
5351             nodes[#nodes][2] = 'en'
5352         end
5353     end
5354
5355     -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
5356     if d == 'an' and #nodes >= 2 then
5357         if (nodes[#nodes][2] == 'cs')
5358             and nodes[#nodes-1][2] == 'an' then
5359             nodes[#nodes][2] = 'an'
5360         end
5361     end
5362
5363     -- ET/EN                -- W5 + W7->1 / W6->on
5364     if d == 'et' then
5365         first_et = first_et or (#nodes + 1)
5366     elseif d == 'en' then
5367         has_en = true
5368         first_et = first_et or (#nodes + 1)
5369     elseif first_et then    -- d may be nil here !
5370         if has_en then
5371             if last == 'l' then
5372                 temp = 'l'    -- W7
5373             else
5374                 temp = 'en'   -- W5
5375             end
5376         else
5377             temp = 'on'       -- W6
5378         end
5379         for e = first_et, #nodes do
5380             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5381         end
5382         first_et = nil
5383         has_en = false
5384     end
5385
5386     if d then
5387         if d == 'al' then
5388             d = 'r'
5389             last = 'al'

```

```

5390     elseif d == 'l' or d == 'r' then
5391         last = d
5392     end
5393     prev_d = d
5394     table.insert(nodes, {item, d, outer_first})
5395 end
5396
5397 outer_first = nil
5398
5399 end
5400
5401 -- TODO -- repeated here in case EN/ET is the last node. Find a
5402 -- better way of doing things:
5403 if first_et then      -- dir may be nil here !
5404     if has_en then
5405         if last == 'l' then
5406             temp = 'l'    -- W7
5407         else
5408             temp = 'en'   -- W5
5409         end
5410     else
5411         temp = 'on'      -- W6
5412     end
5413     for e = first_et, #nodes do
5414         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5415     end
5416 end
5417
5418 -- dummy node, to close things
5419 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5420
5421 ----- NEUTRAL -----
5422
5423 outer = save_outer
5424 last = outer
5425
5426 local first_on = nil
5427
5428 for q = 1, #nodes do
5429     local item
5430
5431     local outer_first = nodes[q][3]
5432     outer = outer_first or outer
5433     last = outer_first or last
5434
5435     local d = nodes[q][2]
5436     if d == 'an' or d == 'en' then d = 'r' end
5437     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5438
5439     if d == 'on' then
5440         first_on = first_on or q
5441     elseif first_on then
5442         if last == d then
5443             temp = d
5444         else
5445             temp = outer
5446         end
5447         for r = first_on, q - 1 do
5448             nodes[r][2] = temp

```



```

5449         item = nodes[r][1]    -- MIRRORING
5450         if Babel.mirroring_enabled and item.id == GLYPH
5451             and temp == 'r' and characters[item.char] then
5452             local font_mode = font.fonts[item.font].properties.mode
5453             if font_mode ~= 'harf' and font_mode ~= 'plug' then
5454                 item.char = characters[item.char].m or item.char
5455             end
5456         end
5457     end
5458     first_on = nil
5459 end
5460
5461     if d == 'r' or d == 'l' then last = d end
5462 end
5463
5464 ----- IMPLICIT, REORDER -----
5465
5466 outer = save_outer
5467 last = outer
5468
5469 local state = {}
5470 state.has_r = false
5471
5472 for q = 1, #nodes do
5473     local item = nodes[q][1]
5474
5475     outer = nodes[q][3] or outer
5476
5477     local d = nodes[q][2]
5478
5479     if d == 'nsm' then d = last end          -- W1
5480     if d == 'en' then d = 'an' end
5481     local isdir = (d == 'r' or d == 'l')
5482
5483     if outer == 'l' and d == 'an' then
5484         state.san = state.san or item
5485         state.ean = item
5486     elseif state.san then
5487         head, state = insert_numeric(head, state)
5488     end
5489
5490     if outer == 'l' then
5491         if d == 'an' or d == 'r' then      -- im -> implicit
5492             if d == 'r' then state.has_r = true end
5493             state.sim = state.sim or item
5494             state.eim = item
5495         elseif d == 'l' and state.sim and state.has_r then
5496             head, state = insert_implicit(head, state, outer)
5497         elseif d == 'l' then
5498             state.sim, state.eim, state.has_r = nil, nil, false
5499         end
5500     else
5501         if d == 'an' or d == 'l' then
5502             if nodes[q][3] then -- nil except after an explicit dir
5503                 state.sim = item -- so we move sim 'inside' the group
5504             else
5505                 state.sim = state.sim or item
5506             end
5507         end

```

```

5508         state.eim = item
5509     elseif d == 'r' and state.sim then
5510         head, state = insert_implicit(head, state, outer)
5511     elseif d == 'r' then
5512         state.sim, state.eim = nil, nil
5513     end
5514 end
5515
5516 if isdir then
5517     last = d          -- Don't search back - best save now
5518 elseif d == 'on' and state.san then
5519     state.san = state.san or item
5520     state.ean = item
5521 end
5522
5523 end
5524
5525 return node.prev(head) or head
5526 end
5527 </basic>

```

16 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.

17 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.

```

5528 <nil>
5529 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
5530 \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.

```

5531 \ifx\l@nil\@undefined
5532   \newlanguage\l@nil
5533   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
5534   \let\bbl@elt\relax
5535   \edef\bbl@languages{% Add it to the list of languages
5536     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}
5537 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
5538 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 5539 \let\captionnil\@empty
5540 \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.

```
5541 \ldf@finish{nil}
5542 \</nil>
```

18 Support for Plain T_EX (plain.def)

18.1 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`

```
5543 \*bplain | blplain)
5544 \catcode`\{=1 % left brace is begin-group character
5545 \catcode`\}=2 % right brace is end-group character
5546 \catcode`\#=6 % hash mark is macro parameter character
```

Now let’s see if a file called `hyphen.cfg` can be found somewhere on T_EX’s input path by trying to open it for reading...

```
5547 \openin 0 hyphen.cfg
```

If the file wasn’t found the following test turns out true.

```
5548 \ifeof0
5549 \else
```

When `hyphen.cfg` could be opened we make sure that *it* will be read instead of the file `hyphen.tex` which should (according to Don Knuth’s ruling) contain the american English hyphenation patterns and nothing else.

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).

```
5550 \let\ainput
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
5551 \def\input #1 {%  
5552   \let\input\acommand  
5553   \acommand hyphen.cfg
```

Once that's done the original meaning of `\input` can be restored and the definition of `\acommand` can be forgotten.

```
5554   \let\acommand\undefined  
5555 }  
5556 \fi  
5557 </bplain | bplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
5558 <bplain>\acommand plain.tex  
5559 <bplain>\acommand 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.

```
5560 <bplain>\def\fmtname{babel-plain}  
5561 <bplain>\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `bplain.tex`, rename it and replace `plain.tex` with the name of your format file.

18.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of $\text{\LaTeX} 2_{\epsilon}$ that are needed for `babel`.

```
5562 <*plain>  
5563 \def\@empty{}  
5564 \def\loadlocalcfg#1{%  
5565   \openin0#1.cfg  
5566   \ifeof0  
5567     \closein0  
5568   \else  
5569     \closein0  
5570     {\immediate\write16{*****}%  
5571       \immediate\write16{* Local config file #1.cfg used}%  
5572       \immediate\write16{*}%  
5573     }  
5574     \input #1.cfg\relax  
5575   \fi  
5576   \@endofldf}
```

18.3 General tools

A number of \LaTeX macro's that are needed later on.

```
5577 \long\def\@firstofone#1{#1}  
5578 \long\def\@firstoftwo#1#2{#1}  
5579 \long\def\@secondoftwo#1#2{#2}  
5580 \def\@nnil{\@nil}  
5581 \def\@gobbletwo#1#2{}  
5582 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}  
5583 \def\@star@or@long#1{%  
5584   \@ifstar  
5585   {\let\l@ngrel@x\relax#1}%  
5586   {\let\l@ngrel@x\long#1}}
```

```

5587 \let\l@ngrel@x\relax
5588 \def\@car#1#2\@nil{#1}
5589 \def\@cdr#1#2\@nil{#2}
5590 \let\@typeset@protect\relax
5591 \let\protected@edef\edef
5592 \long\def\@gobble#1{}
5593 \edef\@backslashchar{\expandafter\@gobble\string\}
5594 \def\strip@prefix#1>{}
5595 \def\g@addto@macro#1#2{%
5596     \toks@\expandafter{#1#2}%
5597     \xdef#1{\the\toks@}}
5598 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
5599 \def\@nameuse#1{\csname #1\endcsname}
5600 \def\@ifundefined#1{%
5601     \expandafter\ifx\csname#1\endcsname\relax
5602     \expandafter\@firstoftwo
5603     \else
5604     \expandafter\@secondoftwo
5605     \fi}
5606 \def\@expandtwoargs#1#2#3{%
5607     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
5608 \def\zap@space#1 #2{%
5609     #1%
5610     \ifx#2\@empty\else\expandafter\zap@space\fi
5611     #2}

```

$\text{\LaTeX} 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

5612 \ifx\@preamblecmds\@undefined
5613     \def\@preamblecmds{}
5614 \fi
5615 \def\@onlypreamble#1{%
5616     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
5617         \@preamblecmds\do#1}}
5618 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

5619 \def\begin{document}{%
5620     \@begin{document}hook
5621     \global\let\@begin{document}hook\@undefined
5622     \def\do##1{\global\let##1\@undefined}%
5623     \@preamblecmds
5624     \global\let\do\noexpand}

5625 \ifx\@begin{document}hook\@undefined
5626     \def\@begin{document}hook{}
5627 \fi
5628 \@onlypreamble\@begin{document}hook
5629 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

5630 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
5631 \@onlypreamble\AtEndOfPackage
5632 \def\@endoflfd{}
5633 \@onlypreamble\@endoflfd
5634 \let\bbl@afterlang\@empty
5635 \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.

```
5636 \ifx\if@filesw\@undefined
5637 \expandafter\let\csname if@filesw\expandafter\endcsname
5638 \csname iffalse\endcsname
5639 \fi
```

Mimick L^AT_EX's commands to define control sequences.

```
5640 \def\newcommand{\@star@or@long\new@command}
5641 \def\new@command#1{%
5642 \@testopt{\@newcommand#1}0}
5643 \def\@newcommand#1[#2]{%
5644 \@ifnextchar [{\@xargdef#1[#2]}%
5645 {\@argdef#1[#2]}}
5646 \long\def\@argdef#1[#2]#3{%
5647 \@yargdef#1\@ne{#2}{#3}}
5648 \long\def\@xargdef#1[#2][#3]#4{%
5649 \expandafter\def\expandafter#1\expandafter{%
5650 \expandafter\@protected@testopt\expandafter #1%
5651 \csname\string#1\expandafter\endcsname{#3}}%
5652 \expandafter\@yargdef \csname\string#1\endcsname
5653 \tw@{#2}{#4}}
5654 \long\def\@yargdef#1#2#3{%
5655 \@tempcnta#3\relax
5656 \advance \@tempcnta \@ne
5657 \let\@hash@\relax
5658 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
5659 \@tempcntb #2%
5660 \@whilenum \@tempcntb <\@tempcnta
5661 \do{%
5662 \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
5663 \advance\@tempcntb \@ne}%
5664 \let\@hash@###
5665 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
5666 \def\providecommand{\@star@or@long\provide@command}
5667 \def\provide@command#1{%
5668 \begingroup
5669 \escapechar\m@ne\xdef\@gtempa{\string#1}%
5670 \endgroup
5671 \expandafter\@ifundefined\@gtempa
5672 {\def\reserved@a{\new@command#1}}%
5673 {\let\reserved@a\relax
5674 \def\reserved@a{\new@command\reserved@a}}%
5675 \reserved@a}%

5676 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
5677 \def\declare@robustcommand#1{%
5678 \edef\reserved@a{\string#1}%
5679 \def\reserved@b{#1}%
5680 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
5681 \edef#1{%
5682 \ifx\reserved@a\reserved@b
5683 \noexpand\x@protect
5684 \noexpand#1%
5685 \fi
5686 \noexpand\protect
5687 \expandafter\noexpand\csname
5688 \expandafter\@gobble\string#1 \endcsname
5689 }%
```

```

5690 \expandafter\new@command\csname
5691 \expandafter\@gobble\string#1 \endcsname
5692 }
5693 \def\x@protect#1{%
5694 \ifx\protect\@typeset@protect\else
5695 \x@protect#1%
5696 \fi
5697 }
5698 \def\x@protect#1\fi#2#3{%
5699 \fi\protect#1%
5700 }

```

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`.

```

5701 \def\bbl@tempa{\csname newif\endcsname\ifin@}
5702 \ifx\in@\@undefined
5703 \def\in@#1#2{%
5704 \def\in@##1##2##3\in@{%
5705 \ifx\in@##2\in@false\else\in@true\fi}%
5706 \in@#2#1\in@\in@}
5707 \else
5708 \let\bbl@tempa\@empty
5709 \fi
5710 \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).

```

5711 \def\@ifpackagewith#1#2#3#4{#3}

```

The \LaTeX macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```

5712 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\text{\LaTeX} 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

5713 \ifx\@tempcnta\@undefined
5714 \csname newcount\endcsname\@tempcnta\relax
5715 \fi
5716 \ifx\@tempcntb\@undefined
5717 \csname newcount\endcsname\@tempcntb\relax
5718 \fi

```

To prevent wasting two counters in $\text{\LaTeX} 2.09$ (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

5719 \ifx\bye\@undefined
5720 \advance\count10 by -2\relax
5721 \fi
5722 \ifx\@ifnextchar\@undefined
5723 \def\@ifnextchar#1#2#3{%
5724 \let\reserved@d=#1%
5725 \def\reserved@a{#2}\def\reserved@b{#3}%

```

```

5726 \futurelet\@let@token\@ifnch}
5727 \def\@ifnch{%
5728 \ifx\@let@token\@sptoken
5729 \let\reserved@c\@xifnch
5730 \else
5731 \ifx\@let@token\reserved@d
5732 \let\reserved@c\reserved@a
5733 \else
5734 \let\reserved@c\reserved@b
5735 \fi
5736 \fi
5737 \reserved@c}
5738 \def\:\let\@sptoken= } \: % this makes \@sptoken a space token
5739 \def\:\@xifnch} \expandafter\def\:\{\futurelet\@let@token\@ifnch}
5740 \fi
5741 \def\@testopt#1#2{%
5742 \@ifnextchar[{\#1}{\#1[#2]}}
5743 \def\@protected@testopt#1{%
5744 \ifx\protect\@typeset@protect
5745 \expandafter\@testopt
5746 \else
5747 \@x@protect#1%
5748 \fi}
5749 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{\#1\relax
5750 #2\relax}\fi}
5751 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
5752 \else\expandafter\@gobble\fi{\#1}}

```

18.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

5753 \def\DeclareTextCommand{%
5754 \@dec@text@cmd\providecommand
5755 }
5756 \def\ProvideTextCommand{%
5757 \@dec@text@cmd\providecommand
5758 }
5759 \def\DeclareTextSymbol#1#2#3{%
5760 \@dec@text@cmd\chardef#1{\#2}\#3\relax
5761 }
5762 \def\@dec@text@cmd#1#2#3{%
5763 \expandafter\def\expandafter#2%
5764 \expandafter{%
5765 \csname#3-cmd\expandafter\endcsname
5766 \expandafter#2%
5767 \csname#3\string#2\endcsname
5768 }%
5769 % \let\@ifdefinable\rc@ifdefinable
5770 \expandafter#1\csname#3\string#2\endcsname
5771 }
5772 \def\@current@cmd#1{%
5773 \ifx\protect\@typeset@protect\else
5774 \noexpand#1\expandafter\@gobble
5775 \fi
5776 }
5777 \def\@changed@cmd#1#2{%
5778 \ifx\protect\@typeset@protect
5779 \expandafter\ifx\csname#1\endcsname\relax

```



```

5780         \expandafter\ifx\csname ?\string#1\endcsname\relax
5781         \expandafter\def\csname ?\string#1\endcsname{%
5782             \@changed@x@err{#1}%
5783         }%
5784     \fi
5785     \global\expandafter\let
5786     \csname\cf@encoding\string#1\expandafter\endcsname
5787     \csname ?\string#1\endcsname
5788 \fi
5789 \csname\cf@encoding\string#1%
5790 \expandafter\endcsname
5791 \else
5792     \noexpand#1%
5793 \fi
5794 }
5795 \def\@changed@x@err#1{%
5796     \errhelp{Your command will be ignored, type <return> to proceed}%
5797     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
5798 \def\DeclareTextCommandDefault#1{%
5799     \DeclareTextCommand#1?%
5800 }
5801 \def\ProvideTextCommandDefault#1{%
5802     \ProvideTextCommand#1?%
5803 }
5804 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
5805 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
5806 \def\DeclareTextAccent#1#2#3{%
5807     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
5808 }
5809 \def\DeclareTextCompositeCommand#1#2#3#4{%
5810     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
5811     \edef\reserved@b{\string##1}%
5812     \edef\reserved@c{%
5813         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
5814     \ifx\reserved@b\reserved@c
5815         \expandafter\expandafter\expandafter\ifx
5816         \expandafter\@car\reserved@a\relax\relax\@nil
5817         \@text@composite
5818     \else
5819         \edef\reserved@b##1{%
5820             \def\expandafter\noexpand
5821             \csname#2\string#1\endcsname####1{%
5822                 \noexpand\@text@composite
5823                 \expandafter\noexpand\csname#2\string#1\endcsname
5824                 ####1\noexpand\@empty\noexpand\@text@composite
5825                 {##1}%
5826             }%
5827         }%
5828         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
5829     \fi
5830     \expandafter\def\csname\expandafter\string\csname
5831     #2\endcsname\string#1-\string#3\endcsname{#4}
5832 \else
5833     \errhelp{Your command will be ignored, type <return> to proceed}%
5834     \errmessage{\string\DeclareTextCompositeCommand\space used on
5835         inappropriate command \protect#1}
5836 \fi
5837 }
5838 \def\@text@composite#1#2#3\@text@composite{%

```

```

5839 \expandafter\@text@composite@x
5840 \csname\string#1-\string#2\endcsname
5841 }
5842 \def\@text@composite@x#1#2{%
5843 \ifx#1\relax
5844 #2%
5845 \else
5846 #1%
5847 \fi
5848 }
5849 %
5850 \def\@strip@args#1:#2-#3\@strip@args{#2}
5851 \def\DeclareTextComposite#1#2#3#4{%
5852 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
5853 \bgroup
5854 \lccode`\@=#4%
5855 \lowercase{%
5856 \egroup
5857 \reserved@a @%
5858 }%
5859 }
5860 %
5861 \def\UseTextSymbol#1#2{%
5862 % \let\@curr@enc\cf@encoding
5863 % \@use@text@encoding{#1}%
5864 #2%
5865 % \@use@text@encoding\@curr@enc
5866 }
5867 \def\UseTextAccent#1#2#3{%
5868 % \let\@curr@enc\cf@encoding
5869 % \@use@text@encoding{#1}%
5870 #2{\@use@text@encoding\@curr@enc\selectfont#3}%
5871 % \@use@text@encoding\@curr@enc
5872 }
5873 \def\@use@text@encoding#1{%
5874 % \edef\font@name{#1}%
5875 % \xdef\font@name{%
5876 % \csname\curr@fontshape/\font@size\endcsname
5877 % }%
5878 % \pickup@font
5879 % \font@name
5880 % \@@enc@update
5881 }
5882 \def\DeclareTextSymbolDefault#1#2{%
5883 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
5884 }
5885 \def\DeclareTextAccentDefault#1#2{%
5886 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
5887 }
5888 \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.

```

5889 \DeclareTextAccent{"}{OT1}{127}
5890 \DeclareTextAccent{'}{OT1}{19}
5891 \DeclareTextAccent{^}{OT1}{94}
5892 \DeclareTextAccent{\`}{OT1}{18}
5893 \DeclareTextAccent{\~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```
5894 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
5895 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
5896 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
5897 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
5898 \DeclareTextSymbol{\i}{OT1}{16}
5899 \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`.

```
5900 \ifx\scriptsize\undefined
5901   \let\scriptsize\sevenrm
5902 \fi
5903 </plain>
```

19 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomititis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).