

Babel

Version 3.51.2203
2020/11/26

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	2
1	The user interface	2
1.1	Monolingual documents	2
1.2	Multilingual documents	4
1.3	Mostly monolingual documents	6
1.4	Modifiers	6
1.5	Troubleshooting	7
1.6	Plain	7
1.7	Basic language selectors	7
1.8	Auxiliary language selectors	8
1.9	More on selection	9
1.10	Shorthands	10
1.11	Package options	14
1.12	The base option	16
1.13	ini files	17
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	Dates	33
1.19	Accessing language info	33
1.20	Hyphenation and line breaking	35
1.21	Selection based on BCP 47 tags	37
1.22	Selecting scripts	38
1.23	Selecting directions	39
1.24	Language attributes	43
1.25	Hooks	43
1.26	Languages supported by babel with ldf files	44
1.27	Unicode character properties in luatex	46
1.28	Tweaking some features	46
1.29	Tips, workarounds, known issues and notes	46
1.30	Current and future work	48
1.31	Tentative and experimental code	48
2	Loading languages with language.dat	48
2.1	Format	49
3	The interface between the core of babel and the language definition files	50
3.1	Guidelines for contributed languages	51
3.2	Basic macros	51
3.3	Skeleton	53
3.4	Support for active characters	54
3.5	Support for saving macro definitions	54
3.6	Support for extending macros	54
3.7	Macros common to a number of languages	55
3.8	Encoding-dependent strings	55
4	Changes	59
4.1	Changes in babel version 3.9	59
II	Source code	59

5	Identification and loading of required files	59
6	locale directory	60
7	Tools	60
7.1	Multiple languages	65
7.2	The Package File (L ^A T _E X, babel.sty)	65
7.3	base	67
7.4	Conditional loading of shorthands	69
7.5	Cross referencing macros	71
7.6	Marks	74
7.7	Preventing clashes with other packages	75
7.7.1	ifthen	75
7.7.2	varioref	75
7.7.3	hhline	76
7.7.4	hyperref	76
7.7.5	fancyhdr	76
7.8	Encoding and fonts	77
7.9	Basic bidi support	79
7.10	Local Language Configuration	84
8	The kernel of Babel (babel.def, common)	88
8.1	Tools	88
9	Multiple languages	89
9.1	Selecting the language	92
9.2	Errors	100
9.3	Hooks	103
9.4	Setting up language files	105
9.5	Shorthands	107
9.6	Language attributes	117
9.7	Support for saving macro definitions	119
9.8	Short tags	120
9.9	Hyphens	120
9.10	Multiencoding strings	122
9.11	Macros common to a number of languages	129
9.12	Making glyphs available	129
9.12.1	Quotation marks	129
9.12.2	Letters	130
9.12.3	Shorthands for quotation marks	131
9.12.4	Umlauts and tremas	132
9.13	Layout	134
9.14	Load engine specific macros	134
9.15	Creating and modifying languages	135
10	Adjusting the Babel bahavior	154
11	Loading hyphenation patterns	156
12	Font handling with fontspec	161

13	Hooks for XeTeX and LuaTeX	165
13.1	XeTeX	165
13.2	Layout	167
13.3	LuaTeX	169
13.4	Southeast Asian scripts	175
13.5	CJK line breaking	178
13.6	Automatic fonts and ids switching	179
13.7	Layout	189
13.8	Auto bidi with basic and basic-r	192
14	Data for CJK	202
15	The ‘nil’ language	203
16	Support for Plain T_EX (plain.def)	203
16.1	Not renaming hyphen.tex	203
16.2	Emulating some L ^A T _E X features	204
16.3	General tools	205
16.4	Encoding related macros	209
17	Acknowledgements	211

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	3
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	4
You are loading directly a language style	7
Unknown language ‘LANG’	7
Argument of \language@active@arg” has an extra }	11
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

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

```

NOTE Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.21 for further details.

1.3 Mostly monolingual documents

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

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

EXAMPLE A trivial document is:

LUATEX/XETEX

```

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

NOTE Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or tree-letter word is a valid name for a language (eg, `yi`). See section 1.21 for further details.

1.4 Modifiers

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

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

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

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

1.5 Troubleshooting

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

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

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

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

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

1.6 Plain

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

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

WARNING Not all languages provide a sty file and some of them are not compatible with Plain.⁴

1.7 Basic language selectors

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

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

`\selectlanguage` {*<language>*}

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

New 3.43 However, if the macro name does not match any language, it will get expanded as expected.

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

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

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

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

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

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

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

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

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

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

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

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

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

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

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

The package inputenc as well as xetex and luatex have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now pdfTeX provides \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 can start with, say:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Basque " ' ~

Breton : ; ? !

Catalan " ' `

Czech " -

Esperanto ^

Estonian " ~

French (all varieties) : ; ? !

Galician " . ' ~ < >

Greek ~

Hungarian `

Kurmanji ^

Latin " ^ =

Slovak " ^ ' -

Spanish " . < > ' ~

Turkish : ! =

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

⁷Thanks to Enrico Gregorio

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

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

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

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

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

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

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

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

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

1.11 Package options

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

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

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

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

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

⁹You can use alternatively the package `silence`.

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

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

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

New 3.14 Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.23.

layout=

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

1.12 The base option

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

\AfterBabelLanguage `{\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}

```

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

1.13 ini files

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

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

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

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

LUATEX/XETEX

```

\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

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

Or also:

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

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

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

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

Hebrew Niqqud marks seem to work in both engines, but cantillation marks are misplaced (xetex or luatex with Harfbuzz seems better, but still problematic).

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

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

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

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

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

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

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

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

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

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

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

ff	Fulah	ksb	Shambala
fi	Finnish ^{ul}	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French ^{ul}	ky	Kyrgyz
fr-BE	French ^{ul}	lag	Langi
fr-CA	French ^{ul}	lb	Luxembourgish
fr-CH	French ^{ul}	lg	Ganda
fr-LU	French ^{ul}	lkt	Lakota
fur	Friulian ^{ul}	ln	Lingala
fy	Western Frisian	lo	Lao ^{ul}
ga	Irish ^{ul}	lrc	Northern Luri
gd	Scottish Gaelic ^{ul}	lt	Lithuanian ^{ul}
gl	Galician ^{ul}	lu	Luba-Katanga
grc	Ancient Greek ^{ul}	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian ^{ul}
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa ¹	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ¹
hy	Armenian ^u	ms-SG	Malay ¹
ia	Interlingua ^{ul}	ms	Malay ^{ul}
id	Indonesian ^{ul}	mt	Maltese
ig	Igbo	mua	Mundang
ii	Sichuan Yi	my	Burmese
is	Icelandic ^{ul}	mzn	Mazanderani
it	Italian ^{ul}	naq	Nama
ja	Japanese	nb	Norwegian Bokmål ^{ul}
jgo	Ngomba	nd	North Ndebele
jmc	Machame	ne	Nepali
ka	Georgian ^{ul}	nl	Dutch ^{ul}
kab	Kabyle	nmg	Kwasio
kam	Kamba	nn	Norwegian Nynorsk ^{ul}
kde	Makonde	nnh	Ngiemboon
kea	Kabuverdianu	nus	Nuer
khq	Koyra Chiini	nyn	Nyankole
ki	Kikuyu	om	Oromo
kk	Kazakh	or	Odia
kkj	Kako	os	Ossetic
kl	Kalaallisut	pa-Arab	Punjabi
klj	Kalenjin	pa-Guru	Punjabi
km	Khmer	pa	Punjabi
kn	Kannada ^{ul}	pl	Polish ^{ul}
ko	Korean	pms	Piedmontese ^{ul}
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese ^{ul}

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

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

aghem	american
akan	amharic
albanian	ancientgreek

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

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

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

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

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

yiddish
yoruba

zarma
zulu afrikaans

Modifying and adding values to ini files

New 3.39 There is a way to modify the values of ini files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghijkl`. Keys may be added, too. Without `import` you may modify the identification keys. This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}
```

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

```
Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

```

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

```

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

EXAMPLE If you need a language named arhinish:

```

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

```

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

```

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

```

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

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

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

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

import= *<language-tag>*

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

```

\babelprovide[import=hu]{hungarian}

```

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

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

```

\babelprovide[import]{hungarian}

```

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

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

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

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

Loads only the strings. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

alph= $\langle counter-name \rangle$

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

Alph= $\langle counter-name \rangle$

Same for `\Alph`.

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

onchar= `ids | fonts`

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

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

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

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

intrapenalty= $\langle penalty \rangle$

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

mapfont= `direction`

Assigns the font for the writing direction of this language (only with `bidi=basic`). Whenever possible, instead of this option use `onchar`, based on the script, which usually makes more sense. More precisely, what `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the `bidi` Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

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

1.17 Digits and counters

New 3.20 About thirty `ini` files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only `xetex` and

luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With the option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on `\arabic`.)

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

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

The styles are:

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

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

Arabic abjad, maghrebi.abjad
Belarusan, Bulgarian, Macedonian, Serbian lower, upper
Bengali alphabetic
Coptic epact, lower.letters
Hebrew letters (neither geresh nor gershayim yet)
Hindi alphabetic
Armenian lower.letter, upper.letter
Japanese hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Georgian letters
Greek lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
Khmer consonant
Korean consonant, syllable, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Marathi alphabetic
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Syriac letters
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\localeid

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.21 Selection based on BCP 47 tags

New 3.43 The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore `babel` will provide a set of tools to select the locales in different situations, adapted to the particular needs of each case. Currently, `babel` provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.

It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in `babel`. Instead the data is taken from the `ini` files, which means currently about 250 tags are already recognized. `Babel` performs a simple lookup in the following way: `fr-Latn-FR` → `fr-Latn` → `fr-FR` → `fr`. Languages with the same resolved name are considered the same. Case is normalized before, so that `fr-latn-fr` → `fr-Latn-FR`. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}
```



```
\end{document}
```

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

`autoload.bcp47` with values on and off.

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

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

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

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

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

1.22 Selecting scripts

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

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

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

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

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

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

¹⁷But still defined for backwards compatibility.

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

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

1.23 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```

\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

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

```

\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}

```

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

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

1.24 Language attributes

`\languageattribute`

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

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

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

1.25 Hooks

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

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

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

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.26 Languages supported by babel with ldf files

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

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton

Bulgarian bulgarian
Catalan catalan
Croatian croatian
Czech czech
Danish danish
Dutch dutch
English english, USenglish, american, UKenglish, british, canadian, australian, newzealand
Esperanto esperanto
Estonian estonian
Finnish finnish
French french, francais, canadien, acadian
Galician galician
German austrian, german, germanb, ngerman, naustrian
Greek greek, polutonikogreek
Hebrew hebrew
Icelandic icelandic
Indonesian indonesian (bahasa, indon, bahasai)
Interlingua interlingua
Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu (bahasam)
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, brazilian (portuges, brazil)¹⁹
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

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

```

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

```

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

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

1.27 Unicode character properties in luatex

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

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

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

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

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

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

1.28 Tweaking some features

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

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

1.29 Tips, workarounds, known issues and notes

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

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

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

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

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

(A recent version of inputenc is required.)

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

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

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

1.30 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²¹ But that is the easy part, because they don't require modifying the 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).

1.31 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

\babelprehyphenation

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

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

Performance is still somewhat poor.

2 Loading languages with language.dat

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

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

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

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

2.1 Format

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

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

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

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

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

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

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

A typical error when using babel is the following:

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

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

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

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

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

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

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

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

The following assumptions are made:

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

Some recommendations:

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

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

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

3.1 Guidelines for contributed languages

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

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

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

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

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

²⁶But not removed, for backward compatibility.

`\<lang>hyphenmins` this language as a ‘dialect’ of the language for which the patterns were loaded as `\language0`. Here “language” is used in the \TeX sense of set of hyphenation patterns. The macro `\<lang>hyphenmins` is used to store the values of the `\lefthyphenmin` and `\righthyphenmin`. Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Skeleton

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}
```

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

```
\AtEndOfPackage{%
  \RequirePackage{dingbat}%      Delay package
```


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

3.4 Support for active characters

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

<code>\initiate@active@char</code>	The internal macro <code>\initiate@active@char</code> is used in language definition files to instruct \TeX to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.
<code>\bbl@activate</code> <code>\bbl@deactivate</code>	The command <code>\bbl@activate</code> is used to change the way an active character expands. <code>\bbl@activate</code> ‘switches on’ the active behavior of the character. <code>\bbl@deactivate</code> lets the active character expand to its former (mostly) non-active self.
<code>\declare@shorthand</code>	The macro <code>\declare@shorthand</code> is used to define the various shorthands. It takes three arguments: the name for the collection of shorthands this definition belongs to; the character (sequence) that makes up the shorthand, i.e. <code>~</code> or <code>"a</code> ; and the code to be executed when the shorthand is encountered. (It does <i>not</i> raise an error if the shorthand character has not been “initiated”.)
<code>\bbl@add@special</code> <code>\bbl@remove@special</code>	The \TeX book states: “Plain \TeX includes a macro called <code>\dospecials</code> that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro <code>\dospecial</code> . \TeX adds another macro called <code>\@sanitize</code> representing the same character set, but without the curly braces. The macros <code>\bbl@add@special⟨char⟩</code> and <code>\bbl@remove@special⟨char⟩</code> add and remove the character <code>⟨char⟩</code> to these two sets.

3.5 Support for saving macro definitions

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

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

3.6 Support for extending macros

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

²⁷This mechanism was introduced by Bernd Raichle.

3.7 Macros common to a number of languages

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

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

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

²⁸In future releases further categories may be added.

```

\SetString\today{\number\day.\~%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

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

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

\EndBabelCommands Marks the end of the series of blocks.

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

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

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

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

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

\SetStringLoop $\{ \langle macro-name \rangle \} \{ \langle string-list \rangle \}$

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

```

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

```

#1 is replaced by the roman numeral.

\SetCase $[\langle map-list \rangle] \{ \langle toupper-code \rangle \} \{ \langle tolower-code \rangle \}$

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would typically be things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A $\langle map-list \rangle$ is a series of macros using the internal format of `@uclclist` (eg, `\bb\BB\cc\CC`). The mandatory

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

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

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

1 <<version=3.51.2203>>

2 <<date=2020/11/26>>

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

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

```

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

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

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

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

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

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

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

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

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

```

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


```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

\bbl@replace

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

161 \def\bbl@cased{%
162 \ifx\oe\OE
163 \expandafter\in@\expandafter
164 {\expandafter\OE\expandafter}\expandafter{\oe}%
165 \ifin@
166 \bbl@afterelse\expandafter\MakeUppercase
167 \else
168 \bbl@afterfi\expandafter\MakeLowercase
169 \fi
170 \else
171 \expandafter\@firstofone
172 \fi}
173 <</Basic macros>>

```

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

```

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

```

7.1 Multiple languages

`\language` Plain \TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in `switch.def` and `hyphen.cfg`; the latter may seem redundant, but remember `babel` doesn't require loading `switch.def` in the format.

```

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

```

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

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

```

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

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or \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.

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

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

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

The first two options are for debugging.

```

191 <*package>
192 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
193 \ProvidesPackage{babel}[\<date> \<version>] The Babel package]
194 \@ifpackagewith{babel}{debug}
195   {\providecommand\bb1@trace[1]{\message{^^J[ #1 ]}}%

```

```

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

```

```

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

```

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

```

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

```

7.3 base

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

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

```

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

```

```

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

```

345 \DeclareOption{KeepShorthandsActive}{}
346 \DeclareOption{activeacute}{}
347 \DeclareOption{activegrave}{}
348 \DeclareOption{debug}{}
349 \DeclareOption{noconfigs}{}
350 \DeclareOption{showlanguages}{}
351 \DeclareOption{silent}{}
352 \DeclareOption{mono}{}
353 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
354 \chardef\bbl@iniflag\z@

```

```

355 \DeclareOption{provide=*}{\chardef\bb@iniflag\@ne} % main -> +1
356 \DeclareOption{provide+=*}{\chardef\bb@iniflag\tw@} % add = 2
357 \DeclareOption{provide*=*}{\chardef\bb@iniflag\thr@@} % add + main
358 % A separate option
359 \let\bb@autoload@options\@empty
360 \DeclareOption{provide@=*}{\def\bb@autoload@options{import}}
361 % Don't use. Experimental. TODO.
362 \newif\ifbb@single
363 \DeclareOption{selectors=off}{\bb@singletrue}
364 <<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.

```

365 \let\bb@opt@shorthands\@nnil
366 \let\bb@opt@config\@nnil
367 \let\bb@opt@main\@nnil
368 \let\bb@opt@headfoot\@nnil
369 \let\bb@opt@layout\@nnil

```

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

```

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

```

381 \let\bb@language@opts\@empty
382 \DeclareOption*{%
383   \bb@xin@{\string=}{\CurrentOption}%
384   \ifin@
385     \expandafter\bb@tempa\CurrentOption\bb@tempa
386   \else
387     \bb@add@list\bb@language@opts{\CurrentOption}%
388   \fi}

```

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

```

389 \ProcessOptions*

```

7.4 Conditional loading of shorthands

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

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


```

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

```

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

```

404 \def\bbl@ifshorthand#1{%
405   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
406   \ifin@
407     \expandafter\@firstoftwo
408   \else
409     \expandafter\@secondoftwo
410   \fi}

```

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

```

411 \edef\bbl@opt@shorthands{%
412   \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.

```

413 \bbl@ifshorthand{'}%
414   {\PassOptionsToPackage{activeacute}{babel}}{}
415 \bbl@ifshorthand{`}%
416   {\PassOptionsToPackage{activegrave}{babel}}{}
417 \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.

```

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

```

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

```

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

```

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

```

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

```

444 \input babel.def

```

7.5 Cross referencing macros

The \LaTeX book states:

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

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

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

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

```

445 <<More package options>> ≡
446 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
447 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
448 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
449 <</More package options>>

```

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

```

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

```

460 \CheckCommand*\@testdef[3]{%
461   \def\reserved@a{#3}%

```

```

462 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
463 \else
464 \@tempwatrue
465 \fi}

```

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

```

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

```

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

```

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

```

491 \bbl@xin@{B}\bbl@opt@safe
492 \ifin@
493 \bbl@redefine\@citex[#1]#2{%
494 \@safe@activetrue\edef\@tempa{#2}\@safe@activetrue}
495 \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.

```

496 \AtBeginDocument{%
497 \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.)

```
498 \def\@citex[#1][#2]#3{%
499   \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
500   \org@@citex[#1][#2]{\@tempa}}%
501 }{}}
```

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

```
502 \AtBeginDocument{%
503   \ifpackageloaded{cite}{%
504     \def\@citex[#1]#2{%
505       \@safe@activestrue\org@@citex[#1][#2]\@safe@activesfalse}%
506     }{}}
```

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

```
507 \bbl@redefine\nocite#1{%
508   \@safe@activestrue\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@activestrue` 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.

```
509 \bbl@redefine\bibcite{%
510   \bbl@cite@choice
511   \bibcite}
```

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

```
512 \def\bbl@bibcite#1#2{%
513   \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.

```
514 \def\bbl@cite@choice{%
515   \global\let\bibcite\bbl@bibcite
516   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
517   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
518   \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.

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

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

```
520 \bbl@redefine\@bibitem#1{%
```

```

521 \safe@activetrue\org@@bibitem{#1}\safe@activesfalse}
522 \else
523 \let\org@nocite\nocite
524 \let\org@@citex\citex
525 \let\org@bibcite\bibcite
526 \let\org@@bibitem\bibitem
527 \fi

```

7.6 Marks

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

```

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

```

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

```

549 \ifx\@mkboth\markboth
550 \def\bbl@tempc{\let\@mkboth\markboth}
551 \else
552 \def\bbl@tempc{}
553 \fi
554 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
555 \markboth#1#2{%
556 \protected@edef\bbl@tempb##1{%
557 \protect\foreignlanguage
558 {\language}{\protect\bbl@restore@actives##1}}%
559 \bbl@ifblank{#1}%
560 {\toks@{}}%
561 {\toks@\expandafter{\bbl@tempb{#1}}}%

```

```

562     \bbl@ifblank{#2}%
563     {\@temptokena{}}%
564     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
565     \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}%
566     \bbl@tempc
567     \fi} % end ifbbl@single, end \IfBabelLayout

```

7.7 Preventing clashes with other packages

7.7.1 ifthen

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

```

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

```

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

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

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

```

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

```

7.7.2 varioref

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

```

591 \AtBeginDocument{%

```

```

592 \ifpackageloaded{varioref}{%
593   \bbl@redefine\@vpageref#1[#2]#3{%
594     \@safe@activetrue
595     \org@@vpageref{#1}[#2]#3}%
596   \@safe@activetrue}%
597 \bbl@redefine\hrefpagenum#1#2{%
598   \@safe@activetrue
599   \org@hrefpagenum{#1}#2}%
600   \@safe@activetrue}%

```

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.

```

601 \expandafter\def\csname Ref \endcsname#1{%
602   \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
603 }{}%
604 }
605 \fi

```

7.7.3 `hhline`

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the `:` character which is made active by the french support in `babel`. Therefore we need to *reload* the package when the `:` is an active character. Note that this happens *after* the category code of the `@`-sign has been changed to other, so we need to temporarily change it to letter again.

```

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

```

7.7.4 `hyperref`

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

```

615 \AtBeginDocument{%
616   \ifx\pdfstringdefDisableCommands\@undefined\else
617     \pdfstringdefDisableCommands{\languageshorthands{system}}%
618   \fi}

```

7.7.5 `fancyhdr`

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

```

619 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
620   \lowercase{\foreignlanguage{#1}}}%

```

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

```

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

```

7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, `fontenc` deletes its package options, so we must guess which encodings has been loaded by traversing `\@filelist` to search for `\enc.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`

```

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

```



```

660      {}%
661      {\bbl@xin@{, #1, }, {\BabelNonASCII, \BabelNonText,}%
662      \ifin@
663        \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
664        \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
665      \else
666        \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
667      \fi}%
668    \fi}%
669    \bbl@foreach\@filelist{\bbl@tempb#1\@{}}% TODO - \@{ de mas??
670    \bbl@xin@{\cf@encoding,}{, \BabelNonASCII, \BabelNonText,}%
671    \ifin@else
672      \edef\ensureascii#1{%
673        \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}%
674    \fi
675    \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.

```

676 \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.

```

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

```

701 \DeclareRobustCommand{\latintext}{%

```

```

702 \fontencoding{\latinencoding}\selectfont
703 \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.

704 \ifx\@undefined\DeclareTextFontCommand
705 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
706 \else
707 \DeclareTextFontCommand{\textlatin}{\latintext}
708 \fi

```

7.9 Basic bidi support

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

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

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

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

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

```

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

```

```

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

```

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

```

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

```

```

781 \ifx\RTLfootnotetext\@undefined
782 \AtEndOfPackage{%
783 \EnableBabelHook{babel-bidi}%
784 \ifx\fontspec\@undefined
785 \bbl@loadfontspec % bidi needs fontspec
786 \fi
787 \usepackage#1{bidi}}%
788 \fi}
789 \ifnum\bbl@bidimode>200
790 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
791 \bbl@tentative{bidi=bidi}
792 \bbl@loadxebidi{}
793 \or
794 \bbl@loadxebidi{[rldocument]}
795 \or
796 \bbl@loadxebidi{}
797 \fi
798 \fi
799 \fi
800 \ifnum\bbl@bidimode=\@ne
801 \let\bbl@beforeforeign\leavevmode
802 \ifodd\bbl@engine
803 \newattribute\bbl@attr@dir
804 \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
805 \fi
806 \AtEndOfPackage{%
807 \EnableBabelHook{babel-bidi}%
808 \ifodd\bbl@engine\else
809 \bbl@xebidipar
810 \fi}
811 \fi

```

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

```

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

```

```

837 \or
838 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
839 \fi
840 \fi}
841 \def\bbl@switchdir{%
842 \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}{}%
843 \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs{\language}}{}%
844 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
845 \def\bbl@setdirs#1{% TODO - math
846 \ifcase\bbl@select@type % TODO - strictly, not the right test
847 \bbl@bodydir{#1}%
848 \bbl@pdir{#1}%
849 \fi
850 \bbl@texdir{#1}}
851 % TODO. Only if \bbl@bidimode > 0?:
852 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
853 \DisableBabelHook{babel-bidi}

```

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

```

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

```

```

894 \frozen@everydisplay\expandafter{%
895 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
896 \else % pdftex=0, xetex=2
897 \newcount\bbl@dirlevel
898 \chardef\bbl@thetextdir\z@
899 \chardef\bbl@thepardir\z@
900 \def\bbl@textdir#1{%
901 \ifcase#1\relax
902 \chardef\bbl@thetextdir\z@
903 \bbl@textdir@i\beginL\endL
904 \else
905 \chardef\bbl@thetextdir\@ne
906 \bbl@textdir@i\beginR\endR
907 \fi}
908 \def\bbl@textdir@i#1#2{%
909 \ifhmode
910 \ifnum\currentgrouplevel>\z@
911 \ifnum\currentgrouplevel=\bbl@dirlevel
912 \bbl@error{Multiple bidi settings inside a group}%
913 {I'll insert a new group, but expect wrong results.}%
914 \bgroup\aftergroup#2\aftergroup\egroup
915 \else
916 \ifcase\currentgrouptype\or % 0 bottom
917 \aftergroup#2% 1 simple {}
918 \or
919 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
920 \or
921 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
922 \or\or\or % vbox vtop align
923 \or
924 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
925 \or\or\or\or\or\or % output math disc insert vcent mathchoice
926 \or
927 \aftergroup#2% 14 \begingroup
928 \else
929 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
930 \fi
931 \fi
932 \bbl@dirlevel\currentgrouplevel
933 \fi
934 #1%
935 \fi}
936 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
937 \let\bbl@bodydir\@gobble
938 \let\bbl@pagedir\@gobble
939 \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).

```

940 \def\bbl@xebidipar{%
941 \let\bbl@xebidipar\relax
942 \TeXeTstate\@ne
943 \def\bbl@xeeverypar{%
944 \ifcase\bbl@thepardir
945 \ifcase\bbl@thetextdir\else\beginR\fi
946 \else
947 {\setbox\z@\lastbox\beginR\box\z@}%
948 \fi}%

```

```

949 \let\bbl@severypar\everypar
950 \newtoks\everypar
951 \everypar=\bbl@severypar
952 \bbl@severypar{\bbl@xeverypar\the\everypar}}
953 \ifnum\bbl@bidimode>200
954 \let\bbl@textdir@i\@gobbletwo
955 \let\bbl@xebidipar\@empty
956 \AddBabelHook{bidi}{foreign}{%
957   \def\bbl@tempa{\def\BabelText####1}%
958   \ifcase\bbl@thetextdir
959     \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
960   \else
961     \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
962   \fi}
963 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
964 \fi
965 \fi

```

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

```

966 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
967 \AtBeginDocument{%
968   \ifx\pdfstringdefDisableCommands\@undefined\else
969     \ifx\pdfstringdefDisableCommands\relax\else
970       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
971     \fi
972   \fi}

```

7.10 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `norsk.cfg` will be loaded when the language definition file `norsk.ldf` is loaded.

For plain-based formats we don't want to override the definition of `\loadlocalcfg` from `plain.def`.

```

973 \bbl@trace{Local Language Configuration}
974 \ifx\loadlocalcfg\@undefined
975   \@ifpackagewith{babel}{noconfigs}%
976   {\let\loadlocalcfg\@gobble}%
977   {\def\loadlocalcfg#1{%
978     \InputIfFileExists{#1.cfg}%
979     {\typeout{*****^J%
980               * Local config file #1.cfg used^^J%
981               *}}%
982     \@empty}}
983 \fi

```

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

```

984 \ifx\@unexpandable@protect\@undefined
985   \def\@unexpandable@protect{\noexpand\protect\noexpand}
986   \long\def\protected@write#1#2#3{%
987     \begingroup
988       \let\thepage\relax
989       #2%
990       \let\protect\@unexpandable@protect
991       \edef\reserved@a{\write#1{#3}}%

```

```

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

```

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

```

1024 \def\bbl@try@load@lang#1#2#3{%
1025 \IfFileExists{\CurrentOption.ldf}%
1026 {\bbl@load@language{\CurrentOption}}%
1027 {#1\bbl@load@language{#2}#3}}
1028 \DeclareOption{hebrew}{%
1029 \input{rlbabel.def}%
1030 \bbl@load@language{hebrew}}
1031 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1032 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1033 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1034 \DeclareOption{polutonikogreek}{%
1035 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1036 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1037 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1038 \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.

```

1039 \ifx\bbl@opt@config\@nnil

```



```

1040 \ifpackagewith{babel}{noconfigs}{}%
1041 {\InputIfFileExists{bblopts.cfg}%
1042  {\typeout{*****^J%
1043           * Local config file bblopts.cfg used^^J%
1044           *}}%
1045  }%
1046 \else
1047 \InputIfFileExists{\bbl@opt@config.cfg}%
1048 {\typeout{*****^J%
1049           * Local config file \bbl@opt@config.cfg used^^J%
1050           *}}%
1051 {\bbl@error{%
1052   Local config file '\bbl@opt@config.cfg' not found}%
1053   Perhaps you misspelled it.}%
1054 \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.

```

1055 \let\bbl@tempc\relax
1056 \bbl@foreach\bbl@language@opts{%
1057   \ifcase\bbl@iniflag
1058     \bbl@ifunset{ds@#1}%
1059     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1060     {}%
1061   \or
1062     \@gobble % case 2 same as 1
1063   \or
1064     \bbl@ifunset{ds@#1}%
1065     {\IfFileExists{#1.ldf}{}%
1066      {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}}}%
1067      {}%
1068     \bbl@ifunset{ds@#1}%
1069     {\def\bbl@tempc{#1}%
1070      \DeclareOption{#1}{%
1071        \ifnum\bbl@iniflag>\@ne
1072          \bbl@ldfinit
1073          \babelprovide[import]{#1}%
1074          \bbl@afterldf}%
1075        \else
1076          \bbl@load@language{#1}%
1077        \fi}%
1078     {}%
1079   \or
1080     \def\bbl@tempc{#1}%
1081     \bbl@ifunset{ds@#1}%
1082     {\DeclareOption{#1}{%
1083      \bbl@ldfinit
1084      \babelprovide[import]{#1}%
1085      \bbl@afterldf}}}%
1086     {}%
1087 \fi}

```

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

```

1088 \let\bbl@tempb\@nnil
1089 \bbl@foreach\@classoptionslist{%
1090   \bbl@ifunset{ds@#1}%
1091     {\IfFileExists{#1.ldf}{}%
1092      {\IfFileExists{babel-#1.tex}{%\DeclareOption{#1}{}}}%
1093     }%
1094   \bbl@ifunset{ds@#1}%
1095     {\def\bbl@tempb{#1}%
1096      \DeclareOption{#1}{%
1097        \ifnum\bbl@iniflag>\@ne
1098          \bbl@ldfinit
1099          \babelprovide[import]{#1}%
1100          \bbl@afterldf}%
1101      \else
1102        \bbl@load@language{#1}%
1103      \fi}%
1104   {}%

```

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

```

1105 \ifnum\bbl@iniflag=\z@ \else
1106   \ifx\bbl@opt@main\@nnil
1107     \ifx\bbl@tempc\relax
1108       \let\bbl@opt@main\bbl@tempb
1109     \else
1110       \let\bbl@opt@main\bbl@tempc
1111     \fi
1112   \fi
1113 \fi
1114 \ifx\bbl@opt@main\@nnil \else
1115   \expandafter
1116   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1117   \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1118 \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):

```

1119 \def\AfterBabelLanguage#1{%
1120   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1121 \DeclareOption*{}
1122 \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`.

```

1123 \bbl@trace{Option 'main'}
1124 \ifx\bbl@opt@main\@nnil
1125   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1126   \let\bbl@tempc\@empty
1127   \bbl@for\bbl@tempb\bbl@tempa{%
1128     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1129     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
1130   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1131   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1132   \ifx\bbl@tempb\bbl@tempc \else

```

```

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

1158 \ifx\bbl@main@language\@undefined
1159 \bbl@info{%
1160     You haven't specified a language. I'll use 'nil'\%
1161     as the main language. Reported}
1162 \bbl@load@language{nil}
1163 \fi
1164 \</package>
1165 \<core>

```

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

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

8.1 Tools

```

1166 \ifx\ldf@quit\@undefined\else
1167 \endinput\fi % Same line!
1168 \<<Make sure ProvidesFile is defined>>

```

```
1169 \ProvidesFile{babel.def}[\langle date \rangle \langle version \rangle] Babel common definitions]
```

The file `babel.def` expects some definitions made in the $\text{\LaTeX} 2_{\epsilon}$ style file. So, In $\text{\LaTeX} 2.09$ 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).

```
1170 \ifx\AtBeginDocument\@undefined % TODO. change test.
1171   \langle Emulate LaTeX \rangle
1172   \def\language{english}%
1173   \let\bblopt@shorthands\@nnil
1174   \def\bblopt@ifshorthand#1#2#3{#2}%
1175   \let\bblopt@language@opts\@empty
1176   \ifx\babeloptionstrings\@undefined
1177     \let\bblopt@strings\@nnil
1178   \else
1179     \let\bblopt@strings\babeloptionstrings
1180   \fi
1181   \def\BabelStringsDefault{generic}
1182   \def\bblopt@tempa{normal}
1183   \ifx\babeloptionmath\bblopt@tempa
1184     \def\bblopt@mathnormal{\noexpand\textormath}
1185   \fi
1186   \def\AfterBabelLanguage#1#2{}
1187   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1188   \let\bblopt@afterlang\relax
1189   \def\bblopt@safe{BR}
1190   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1191   \ifx\bblopt@trace\@undefined\def\bblopt@trace#1{}\fi
1192   \expandafter\newif\csname ifbblopt@single\endcsname
1193   \chardef\bblopt@bidimode\z@
1194 \fi
```

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

```
1195 \ifx\bblopt@trace\@undefined
1196   \let\LdfInit\endinput
1197   \def\ProvidesLanguage#1{\endinput}
1198 \endinput\fi % Same line!
```

And continue.

9 Multiple languages

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

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

```
1199 \langle Define core switching macros \rangle
```

`\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.

```
1200 \def\bblopt@version{\langle version \rangle}
1201 \def\bblopt@date{\langle date \rangle}
1202 \def\adddialect#1#2{%
1203   \global\chardef#1#2\relax
1204   \bblopt@usehooks{adddialect}{#1}{#2}%
1205   \begingroup
```

```

1206 \count@#1\relax
1207 \def\bbl@elt##1##2##3##4{%
1208 \ifnum\count@=##2\relax
1209 \bbl@info{\string#1 = using hyphenrules for ##1\\%
1210 (\string\language\the\count@)}%
1211 \def\bbl@elt####1####2####3####4{%
1212 \fi}%
1213 \bbl@cs{languages}%
1214 \endgroup}

```

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

```

1215 \def\bbl@fixname#1{%
1216 \begingroup
1217 \def\bbl@tempe{l}%
1218 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1219 \bbl@tempd
1220 {\lowercase\expandafter{\bbl@tempd}%
1221 {\uppercase\expandafter{\bbl@tempd}%
1222 \@empty
1223 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1224 \uppercase\expandafter{\bbl@tempd}}}%
1225 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1226 \lowercase\expandafter{\bbl@tempd}}}%
1227 \@empty
1228 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1229 \bbl@tempd
1230 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1231 \def\bbl@iflanguage#1{%
1232 \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

1233 \def\bbl@bcpcase#1#2#3#4\@@#5{%
1234 \ifx\@empty#3%
1235 \uppercase{\def#5{#1#2}}%
1236 \else
1237 \uppercase{\def#5{#1}}%
1238 \lowercase{\edef#5{#5#2#3#4}}%
1239 \fi}
1240 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
1241 \let\bbl@bcp\relax
1242 \lowercase{\def\bbl@tempa{#1}}%
1243 \ifx\@empty#2%
1244 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1245 \else\ifx\@empty#3%
1246 \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1247 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%

```

```

1248     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1249     {}}%
1250     \ifx\bbl@bcp\relax
1251       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1252     \fi
1253   \else
1254     \bbl@bcp#2\@empty\@empty\@empty\bbl@tempb
1255     \bbl@bcp#3\@empty\@empty\@empty\bbl@tempc
1256     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1257       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1258     {}}%
1259     \ifx\bbl@bcp\relax
1260       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1261       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1262     {}%
1263   \fi
1264   \ifx\bbl@bcp\relax
1265     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1266     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1267   {}%
1268 \fi
1269 \ifx\bbl@bcp\relax
1270   \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1271 \fi
1272 \fi\fi}
1273 \let\bbl@initoload\relax
1274 \def\bbl@provide@locale{%
1275   \ifx\babelprovide\undefined
1276     \bbl@error{For a language to be defined on the fly 'base'\\%
1277               is not enough, and the whole package must be\\%
1278               loaded. Either delete the 'base' option or\\%
1279               request the languages explicitly}%
1280     {See the manual for further details.}%
1281   \fi
1282 % TODO. Option to search if loaded, with \LocaleForEach
1283 \let\bbl@auxname\language % Still necessary. TODO
1284 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1285 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1286 \ifbbl@bcpallowed
1287   \expandafter\ifx\csname date\language\endcsname\relax
1288     \expandafter
1289     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@
1290     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1291       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1292       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1293       \expandafter\ifx\csname date\language\endcsname\relax
1294         \let\bbl@initoload\bbl@bcp
1295         \bbl@exp{\@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1296         \let\bbl@initoload\relax
1297       \fi
1298       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1299     \fi
1300   \fi
1301 \fi
1302 \expandafter\ifx\csname date\language\endcsname\relax
1303   \IfFileExists{babel-\language.tex}%
1304   {\bbl@exp{\@babelprovide[\bbl@autoload@options]{\language}}}%
1305   {}%
1306 \fi}

```

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

```
1307 \def\iflanguage#1{%
1308   \bbl@iflanguage{#1}{%
1309     \ifnum\csname l@#1\endcsname=\language
1310       \expandafter\@firstoftwo
1311     \else
1312       \expandafter\@secondoftwo
1313     \fi}}
```

9.1 Selecting the language

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

```
1314 \let\bbl@select@type\z@
1315 \edef\selectlanguage{%
1316   \noexpand\protect
1317   \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`.

```
1318 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1319 \let\xstring\string
```

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

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

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

```
1320 \def\bbl@language@stack{}
```

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

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

`\bbl@pop@language`

```
1321 \def\bbl@push@language{%
1322   \ifx\language\@undefined\else
1323     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1324   \fi}
```

Retrieving information from the stack is a little bit less simple, as we need to remove the element from the stack while storing it in the macro `\language`. For this we first define a helper function.

`\bbl@pop@lang` This macro stores its first element (which is delimited by the ‘+’-sign) in `\language` and stores the rest of the string in `\bbl@language@stack`.

```
1325 \def\bbl@pop@lang#1+#2\@@{%
1326   \edef\language{#1}%
1327   \xdef\bbl@language@stack{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before `\bbl@pop@lang` is executed TeX first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a ‘+’-sign (zero language names won’t occur as this macro will only be called after something has been pushed on the stack).

```
1328 \let\bbl@ifrestoring\@secondoftwo
1329 \def\bbl@pop@language{%
1330   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1331   \let\bbl@ifrestoring\@firstoftwo
1332   \expandafter\bbl@set@language\expandafter{\language}%
1333   \let\bbl@ifrestoring\@secondoftwo}
```

Once the name of the previous language is retrieved from the stack, it is fed to `\bbl@set@language` to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of `\localeid`. This means `\l@...` will be reserved for hyphenation patterns (so that two locales can share the same rules).

```
1334 \chardef\localeid\z@
1335 \def\bbl@id@last{0} % No real need for a new counter
1336 \def\bbl@id@assign{%
1337   \bbl@ifunset{bbl@id@\language}%
1338   {\count@bbl@id@last\relax
1339     \advance\count@\@ne
1340     \bbl@csarg\chardef{id@\language}\count@
1341     \edef\bbl@id@last{\the\count@}%
1342     \ifcase\bbl@engine\or
1343       \directlua{
1344         Babel = Babel or {}
1345         Babel.locale_props = Babel.locale_props or {}
1346         Babel.locale_props[\bbl@id@last] = {}
1347         Babel.locale_props[\bbl@id@last].name = '\language'
1348       }%
1349     \fi}%
1350   }%
1351   \chardef\localeid\bbl@c{id@}}
```

The unprotected part of `\selectlanguage`.

```
1352 \expandafter\def\csname selectlanguage \endcsname#1{%
1353   \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\tw\fi
1354   \bbl@push@language
1355   \aftergroup\bbl@pop@language
1356   \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.

```

1357 \def\BabelContentsFiles{toc,lof,lot}
1358 \def\bbl@set@language#1{% from selectlanguage, pop@
1359 % The old buggy way. Preserved for compatibility.
1360 \edef\language{%
1361   \ifnum\escapechar=\expandafter`\string#1\@empty
1362   \else\string#1\@empty\fi}%
1363 \ifcat\relax\noexpand#1%
1364   \expandafter\ifx\csname date\language\endcsname\relax
1365   \edef\language{#1}%
1366   \let\locale\language
1367 \else
1368   \bbl@info{Using '\string\language' instead of 'language' is\\%
1369             deprecated. If what you want is to use a\\%
1370             macro containing the actual locale, make\\%
1371             sure it does not not match any language.\\%
1372             Reported}%
1373 %             I'll\\%
1374 %             try to fix '\string\locale', but I cannot promise\\%
1375 %             anything. Reported}%
1376   \ifx\scantokens\undefined
1377     \def\locale{??}%
1378   \else
1379     \scantokens\expandafter{\expandafter
1380       \def\expandafter\locale\expandafter{\language}}%
1381   \fi
1382 \fi
1383 \else
1384   \def\locale{#1}% This one has the correct catcodes
1385 \fi
1386 \select@language{\language}%
1387 % write to aux
1388 \expandafter\ifx\csname date\language\endcsname\relax\else
1389   \if@files
1390     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1391       \protected@write\@auxout{\string\babel@aux{\bbl@auxname}}}%
1392     \fi
1393     \bbl@usehooks{write}}}%
1394   \fi
1395 \fi}
1396 %
1397 \newif\ifbbl@bcpallowed
1398 \bbl@bcpallowedfalse
1399 \def\select@language#1{% from set@, babel@aux
1400 % set hymap
1401 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
1402 % set name
1403 \edef\language{#1}%
1404 \bbl@fixname\language
1405 % TODO. name@map must be here?
1406 \bbl@provide@locale
1407 \bbl@iflanguage\language{%
1408   \expandafter\ifx\csname date\language\endcsname\relax

```

```

1409 \bbl@error
1410 {Unknown language '\language'. Either you have\\%
1411 misspelled its name, it has not been installed,\\%
1412 or you requested it in a previous run. Fix its name,\\%
1413 install it or just rerun the file, respectively. In\\%
1414 some cases, you may need to remove the aux file}%
1415 {You may proceed, but expect wrong results}%
1416 \else
1417 % set type
1418 \let\bbl@select@type\z@
1419 \expandafter\bbl@switch\expandafter{\language}%
1420 \fi}}
1421 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1422 \select@language{#1}%
1423 \bbl@foreach\BabelContentsFiles{%
1424 \@writefile{##1}{\babel@toc{#1}{#2}}}% %% TODO - ok in plain?
1425 \def\babel@toc#1#2{%
1426 \select@language{#1}}

```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring \TeX in a certain pre-defined state.

The name of the language is stored in the control sequence `\language`. Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\<lang>hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\<lang>hyphenmins` will be used.

```

1427 \newif\ifbbl@usedategroup
1428 \def\bbl@switch#1{% from select@, foreign@
1429 % make sure there is info for the language if so requested
1430 \bbl@ensureinfo{#1}%
1431 % restore
1432 \originalTeX
1433 \expandafter\def\expandafter\originalTeX\expandafter{%
1434 \csname noextras#1\endcsname
1435 \let\originalTeX\@empty
1436 \babel@beginsave}%
1437 \bbl@usehooks{afterreset}}}%
1438 \languageshorthands{none}%
1439 % set the locale id
1440 \bbl@id@assign
1441 % switch captions, date
1442 % No text is supposed to be added here, so we remove any
1443 % spurious spaces.
1444 \bbl@bsphack
1445 \ifcase\bbl@select@type
1446 \csname captions#1\endcsname\relax
1447 \csname date#1\endcsname\relax
1448 \else
1449 \bbl@xin@{,captions,},{, \bbl@select@opts,}%
1450 \ifin@

```

```

1451     \csname captions#1\endcsname\relax
1452     \fi
1453     \bbl@xin@{,date,}{, \bbl@select@opts,}%
1454     \ifin@ % if \foreign... within \<lang>date
1455     \csname date#1\endcsname\relax
1456     \fi
1457     \fi
1458     \bbl@esphack
1459     % switch extras
1460     \bbl@usehooks{beforeextras}{}%
1461     \csname extras#1\endcsname\relax
1462     \bbl@usehooks{afterextras}{}%
1463     % > babel-ensure
1464     % > babel-sh-<short>
1465     % > babel-bidi
1466     % > babel-fontspec
1467     % hyphenation - case mapping
1468     \ifcase\bbl@opt@hyphenmap\or
1469     \def\BabelLower##1##2{\lccode##1=##2\relax}%
1470     \ifnum\bbl@hymapsel>4\else
1471     \csname\language\name @bbl@hyphenmap\endcsname
1472     \fi
1473     \chardef\bbl@opt@hyphenmap\z@
1474     \else
1475     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1476     \csname\language\name @bbl@hyphenmap\endcsname
1477     \fi
1478     \fi
1479     \let\bbl@hymapsel\@cclv
1480     % hyphenation - select patterns
1481     \bbl@patterns{#1}%
1482     % hyphenation - allow stretching with babelnohyphens
1483     \ifnum\language=\l@babelnohyphens
1484     \babel@savevariable\emergencystretch
1485     \emergencystretch\maxdimen
1486     \babel@savevariable\hbadness
1487     \hbadness\@M
1488     \fi
1489     % hyphenation - mins
1490     \babel@savevariable\lefthyphenmin
1491     \babel@savevariable\righthyphenmin
1492     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1493     \set@hyphenmins\tw@\thr@\relax
1494     \else
1495     \expandafter\expandafter\expandafter\set@hyphenmins
1496     \csname #1hyphenmins\endcsname\relax
1497     \fi}

```

otherlanguage The other language 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.

```

1498 \long\def\otherlanguage#1{%
1499 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
1500 \csname selectlanguage \endcsname{#1}%
1501 \ignorespaces}

```

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

```
1502 \long\def\endotherlanguage{%
1503   \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`.

```
1504 \expandafter\def\csname otherlanguage*\endcsname{%
1505   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1506 \def\bbl@otherlanguage@s[#1]#2{%
1507   \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
1508   \def\bbl@select@opts{#1}%
1509   \foreign@language{#2}}
```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```
1510 \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`.

```
1511 \providecommand\bbl@beforeforeign{}
1512 \edef\foreignlanguage{%
1513   \noexpand\protect
1514   \expandafter\noexpand\csname foreignlanguage \endcsname}
1515 \expandafter\def\csname foreignlanguage \endcsname{%
1516   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1517 \providecommand\bbl@foreign@x[3][]{%
1518   \begingroup
1519     \def\bbl@select@opts{#1}%
1520     \let\BabelText\@firstofone
1521     \bbl@beforeforeign
1522     \foreign@language{#2}%
1523     \bbl@usehooks{foreign}{}}%
```

```

1524 \BabelText{#3}% Now in horizontal mode!
1525 \endgroup}
1526 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1527 \begingroup
1528 {\par}%
1529 \let\BabelText\@firstofone
1530 \foreign@language{#1}%
1531 \bbl@usehooks{foreign*}{}%
1532 \bbl@dirparastext
1533 \BabelText{#2}% Still in vertical mode!
1534 {\par}%
1535 \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`.

```

1536 \def\foreign@language#1{%
1537 % set name
1538 \edef\language#1}%
1539 \ifbbl@usedategroup
1540 \bbl@add\bbl@select@opts{,date,}%
1541 \bbl@usedategroupfalse
1542 \fi
1543 \bbl@fixname\language
1544 % TODO. name@map here?
1545 \bbl@provide@locale
1546 \bbl@iflanguage\language{%
1547 \expandafter\ifx\csname date\language\endcsname\relax
1548 \bbl@warning % TODO - why a warning, not an error?
1549 {Unknown language `#1'. Either you have\\%
1550 misspelled its name, it has not been installed,\\%
1551 or you requested it in a previous run. Fix its name,\\%
1552 install it or just rerun the file, respectively. In\\%
1553 some cases, you may need to remove the aux file.\\%
1554 I'll proceed, but expect wrong results.\\%
1555 Reported}%
1556 \fi
1557 % set type
1558 \let\bbl@select@type\@ne
1559 \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.

```

1560 \let\bbl@hyphlist\@empty
1561 \let\bbl@hyphenation@relax
1562 \let\bbl@pttnlist\@empty
1563 \let\bbl@patterns@relax
1564 \let\bbl@hymapsel=\@cclv
1565 \def\bbl@patterns#1{%
1566 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax

```

```

1567 \csname l@#1\endcsname
1568 \edef\bbl@tempa{#1}%
1569 \else
1570 \csname l@#1:\f@encoding\endcsname
1571 \edef\bbl@tempa{#1:\f@encoding}%
1572 \fi
1573 \@expandtwoargs\bbl@usehooks{patterns}{#{1}}{\bbl@tempa}}%
1574 % > luatex
1575 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1576 \begingroup
1577 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1578 \ifin@else
1579 \@expandtwoargs\bbl@usehooks{hyphenation}{#{1}}{\bbl@tempa}}%
1580 \hyphenation{%
1581 \bbl@hyphenation@
1582 \@ifundefined{bbl@hyphenation@#1}%
1583 \empty
1584 {\space\csname bbl@hyphenation@#1\endcsname}}%
1585 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1586 \fi
1587 \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*.

```

1588 \def\hyphenrules#1{%
1589 \edef\bbl@tempf{#1}%
1590 \bbl@fixname\bbl@tempf
1591 \bbl@iflanguage\bbl@tempf{%
1592 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1593 \ifx\languageshorthands@undefined\else
1594 \languageshorthands{none}%
1595 \fi
1596 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1597 \set@hyphenmins\tw@\thr@@\relax
1598 \else
1599 \expandafter\expandafter\expandafter\set@hyphenmins
1600 \csname\bbl@tempf hyphenmins\endcsname\relax
1601 \fi}}
1602 \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.

```

1603 \def\providehyphenmins#1#2{%
1604 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1605 \@namedef{#1hyphenmins}{#2}%
1606 \fi}

```

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

```

1607 \def\set@hyphenmins#1#2{%
1608 \lefthyphenmin#1\relax
1609 \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.

```

1610 \ifx\ProvidesFile\@undefined
1611   \def\ProvidesLanguage#1[#2 #3 #4]{%
1612     \wlog{Language: #1 #4 #3 <#2>}%
1613   }
1614 \else
1615   \def\ProvidesLanguage#1{%
1616     \begingroup
1617       \catcode\ 10 %
1618       \@makeother\/%
1619       \@ifnextchar[%]
1620         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}%
1621   \def\@provideslanguage#1[#2]{%
1622     \wlog{Language: #1 #2}%
1623     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1624   \endgroup}%
1625 \fi

```

`\originalTeX` The macro `\originalTeX` should be known to \TeX at this moment. As it has to be expandable we `\let` it to `\@empty` instead of `\relax`.

```

1626 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi

```

Because this part of the code can be included in a format, we make sure that the macro which initializes the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

1627 \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’:

```

1628 \providecommand\setlocale{%
1629   \bbl@error
1630   {Not yet available}%
1631   {Find an armchair, sit down and wait}}
1632 \let\uselocale\setlocale
1633 \let\locale\setlocale
1634 \let\selectlocale\setlocale
1635 \let\localename\setlocale
1636 \let\textlocale\setlocale
1637 \let\textlanguage\setlocale
1638 \let\languagetext\setlocale

```

9.2 Errors

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

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

```

1639 \edef\bbl@nulllanguage{\string\language=0}

```

```

1640 \ifx\PackageError\@undefined % TODO. Move to Plain
1641 \def\bbl@error#1#2{%
1642   \begingroup
1643     \newlinechar=`^^J
1644     \def\{^^J(babel) }%
1645     \errhelp{#2}\errmessage{\{#1}%
1646   \endgroup}
1647 \def\bbl@warning#1{%
1648   \begingroup
1649     \newlinechar=`^^J
1650     \def\{^^J(babel) }%
1651     \message{\{#1}%
1652   \endgroup}
1653 \let\bbl@infowarn\bbl@warning
1654 \def\bbl@info#1{%
1655   \begingroup
1656     \newlinechar=`^^J
1657     \def\{^^J}%
1658     \wlog{#1}%
1659   \endgroup}
1660 \fi
1661 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1662 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1663   \global\@namedef{#2}{\textbf{?#1?}}%
1664   \@nameuse{#2}%
1665   \bbl@warning{%
1666     \@backslashchar#2 not set. Please, define it\\%
1667     after the language has been loaded (typically\\%
1668     in the preamble) with something like:\\%
1669     \string\renewcommand\@backslashchar#2{..}\\%
1670     Reported}}
1671 \def\bbl@tentative{\protect\bbl@tentative@i}
1672 \def\bbl@tentative@i#1{%
1673   \bbl@warning{%
1674     Some functions for '#1' are tentative.\\%
1675     They might not work as expected and their behavior\\%
1676     could change in the future.\\%
1677     Reported}}
1678 \def\@nolanerr#1{%
1679   \bbl@error
1680     {You haven't defined the language #1\space yet.\\%
1681     Perhaps you misspelled it or your installation\\%
1682     is not complete}%
1683     {Your command will be ignored, type <return> to proceed}}
1684 \def\@nopatterns#1{%
1685   \bbl@warning
1686     {No hyphenation patterns were preloaded for\\%
1687     the language `#1' into the format.\\%
1688     Please, configure your TeX system to add them and\\%
1689     rebuild the format. Now I will use the patterns\\%
1690     preloaded for \bbl@nulllanguage\space instead}}
1691 \let\bbl@usehooks\@gobbletwo
1692 \ifx\bbl@onlyswitch\@empty\endinput\fi
1693 % Here ended switch.def

Here ended switch.def.

1694 \ifx\directlua\@undefined\else
1695   \ifx\bbl@luapatterns\@undefined
1696     \input luababel.def

```



```

1697 \fi
1698 \fi
1699 <<Basic macros>>
1700 \bbl@trace{Compatibility with language.def}
1701 \ifx\bbl@languages\@undefined
1702 \ifx\directlua\@undefined
1703 \openin1 = language.def % TODO. Remove hardcoded number
1704 \ifeof1
1705 \closein1
1706 \message{I couldn't find the file language.def}
1707 \else
1708 \closein1
1709 \begingroup
1710 \def\addlanguage#1#2#3#4#5{%
1711 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1712 \global\expandafter\let\csname l@#1\expandafter\endcsname
1713 \csname lang@#1\endcsname
1714 \fi}%
1715 \def\uselanguage#1{%
1716 \input language.def
1717 \endgroup
1718 \fi
1719 \fi
1720 \chardef\l@english\z@
1721 \fi

```

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

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

```

1722 \def\addto#1#2{%
1723 \ifx#1\@undefined
1724 \def#1{#2}%
1725 \else
1726 \ifx#1\relax
1727 \def#1{#2}%
1728 \else
1729 {\toks@\expandafter{#1#2}%
1730 \xdef#1{\the\toks@}}%
1731 \fi
1732 \fi}

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. `TODO`. Always used with additional expansions. Move them here? Move the macro to basic?

```

1733 \def\bbl@withactive#1#2{%
1734 \begingroup
1735 \lccode`~=`#2\relax
1736 \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 \LaTeX macros completely in case their definitions change

(they have changed in the past). A macro named `\macro` will be saved new control sequences named `\org@macro`.

```
1737 \def\bbl@redefine#1{%
1738   \edef\bbl@tempa{\bbl@stripslash#1}%
1739   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1740   \expandafter\def\csname\bbl@tempa\endcsname}
1741 \@onlypreamble\bbl@redefine
```

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

```
1742 \def\bbl@redefine@long#1{%
1743   \edef\bbl@tempa{\bbl@stripslash#1}%
1744   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1745   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1746 \@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_`.

```
1747 \def\bbl@redefineroobust#1{%
1748   \edef\bbl@tempa{\bbl@stripslash#1}%
1749   \bbl@ifunset{\bbl@tempa\space}%
1750   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1751     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1752   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1753   \@namedef{\bbl@tempa\space}}
1754 \@onlypreamble\bbl@redefineroobust
```

9.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```
1755 \bbl@trace{Hooks}
1756 \newcommand\AddBabelHook[3][{}]{%
1757   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1758   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1759   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1760   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1761     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1762     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1763   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1764 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1765 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1766 \def\bbl@usehooks#1#2{%
1767   \def\bbl@elth##1{%
1768     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1}{#2}}%
1769     \bbl@cs{ev@#1@}%
1770     \ifx\language\@undefined\else % Test required for Plain (?)
1771       \def\bbl@elth##1{%
1772         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}{#2}}%
1773         \bbl@cl{ev@#1}%
1774       \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).

```

1775 \def\bbl@evargs{% <- don't delete this comma
1776   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1777   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1778   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1779   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1780   beforestart=0,language=2}

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro `\bbl@e@<language>` contains `\bbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the `exclude` list. If the `fontenc` is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the `include` list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1781 \bbl@trace{Defining babelensure}
1782 \newcommand\babelensure[2][{}]{% TODO - revise test files
1783   \AddBabelHook{babel-ensure}{afterextras}{%
1784     \ifcase\bbl@select@type
1785       \bbl@cl{e}%
1786       \fi}%
1787   \begingroup
1788     \let\bbl@ens@include\@empty
1789     \let\bbl@ens@exclude\@empty
1790     \def\bbl@ens@fontenc{\relax}%
1791     \def\bbl@tempb##1{%
1792       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1793     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1794     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1795     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1796     \def\bbl@tempc{\bbl@ensure}%
1797     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1798       \expandafter{\bbl@ens@include}}%
1799     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1800       \expandafter{\bbl@ens@exclude}}%
1801     \toks@\expandafter{\bbl@tempc}%
1802     \bbl@exp{%
1803   \endgroup
1804   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1805 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1806   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1807     \ifx##1\undefined % 3.32 - Don't assume the macro exists
1808       \edef##1{\noexpand\bbl@nocaption
1809         {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1810     \fi
1811     \ifx##1\@empty\else
1812       \in@{##1}{#2}%
1813       \ifin\@else
1814         \bbl@ifunset{\bbl@ensure@\language}%
1815         {\bbl@exp{%
1816           \\\DeclareRobustCommand\bbl@ensure@\language>[1]{%

```

```

1817         \\foreignlanguage{\language}%
1818         {\ifx\relax#3\else
1819         \\fontencoding{#3}\\selectfont
1820         \fi
1821         #####1}}}%
1822     }%
1823     \toks@\expandafter{##1}%
1824     \edef##1{%
1825         \bbl@csarg\noexpand{ensure@\language}%
1826         {\the\toks@}}%
1827     \fi
1828     \expandafter\bbl@tempb
1829     \fi}%
1830 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1831 \def\bbl@tempa##1{% elt for include list
1832     \ifx##1\@empty\else
1833         \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1834         \ifin\@else
1835             \bbl@tempb##1\@empty
1836         \fi
1837         \expandafter\bbl@tempa
1838     \fi}%
1839 \bbl@tempa#1\@empty}
1840 \def\bbl@captionslist{%
1841 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1842 \contentsname\listfigurename\listtablename\indexname\figurename
1843 \tablename\partname\encname\ccname\headtoname\pagename\seename
1844 \alsoname\proofname\glossaryname}

```

9.4 Setting up language files

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

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

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

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

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

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

```

1845 \bbl@trace{Macros for setting language files up}
1846 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1847     \let\bbl@screset\@empty
1848     \let\BabelStrings\bbl@opt@string
1849     \let\BabelOptions\@empty
1850     \let\BabelLanguages\relax

```

```

1851 \ifx\originalTeX\undefined
1852   \let\originalTeX\empty
1853 \else
1854   \originalTeX
1855 \fi}
1856 \def\LdfInit#1#2{%
1857   \chardef\atcatcode=\catcode`\@
1858   \catcode`\@=11\relax
1859   \chardef\eqcatcode=\catcode`\=
1860   \catcode`\==12\relax
1861   \expandafter\if\expandafter\@backslashchar
1862     \expandafter\@car\string#2\@nil
1863   \ifx#2\@undefined\else
1864     \ldf@quit{#1}%
1865   \fi
1866 \else
1867   \expandafter\ifx\csname#2\endcsname\relax\else
1868     \ldf@quit{#1}%
1869   \fi
1870 \fi
1871 \bbl@ldfinit}

```

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

```

1872 \def\ldf@quit#1{%
1873   \expandafter\main@language\expandafter{#1}%
1874   \catcode`\@=\atcatcode \let\atcatcode\relax
1875   \catcode`\==\eqcatcode \let\eqcatcode\relax
1876   \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.

```

1877 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1878   \bbl@afterlang
1879   \let\bbl@afterlang\relax
1880   \let\BabelModifiers\relax
1881   \let\bbl@screset\relax}%
1882 \def\ldf@finish#1{%
1883   \ifx\loadlocalcfg\undefined\else % For LaTeX 209
1884     \loadlocalcfg{#1}%
1885   \fi
1886   \bbl@afterldf{#1}%
1887   \expandafter\main@language\expandafter{#1}%
1888   \catcode`\@=\atcatcode \let\atcatcode\relax
1889   \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 .

```

1890 \@onlypreamble\LdfInit
1891 \@onlypreamble\ldf@quit
1892 \@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.

```

1893 \def\main@language#1{%
1894   \def\bbl@main@language{#1}%
1895   \let\language\main@language % TODO. Set localename
1896   \bbl@id@assign
1897   \bbl@patterns{\language}%

```

We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the `\AtBeginDocument` is executed. Languages do not set `\pagedir`, so we set here for the whole document to the `main \bodydir`.

```

1898 \def\bbl@beforestart{%
1899   \bbl@usehooks{beforestart}{}%
1900   \global\let\bbl@beforestart\relax}
1901 \AtBeginDocument{%
1902   \@nameuse{bbl@beforestart}%
1903   \if@files
1904     \providecommand\babel@aux[2]{%
1905       \immediate\write\@mainaux{%
1906         \string\providecommand\string\babel@aux[2]{}%
1907         \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1908       }%
1909     \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1910     \ifbbl@single % must go after the line above.
1911       \renewcommand\selectlanguage[1]{%
1912         \renewcommand\foreignlanguage[2]{#2}%
1913         \global\let\babel@aux\@gobbletwo % Also as flag
1914       }%
1915     \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1916 \def\select@language@x#1{%
1917   \ifcase\bbl@select@type
1918     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1919   \else
1920     \select@language{#1}%
1921   \fi}

```

9.5 Shorthands

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

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

```

1922 \bbl@trace{Shorhands}
1923 \def\bbl@add@special#1{% 1:a macro like "\", \?, etc.
1924   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1925   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1926   \if\nfss@catcodes\undefined\else % TODO - same for above
1927     \begingroup
1928       \catcode`#1\active
1929       \nfss@catcodes
1930       \ifnum\catcode`#1=\active
1931         \endgroup
1932       \bbl@add\nfss@catcodes{\@makeother#1}%

```

```

1933     \else
1934     \endgroup
1935     \fi
1936 \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.

```

1937 \def\bbl@remove@special#1{%
1938   \begingroup
1939   \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1940     \else\noexpand##1\noexpand##2\fi}%
1941   \def\do{\x\do}%
1942   \def\@makeother{\x\@makeother}%
1943   \edef\x{\endgroup
1944     \def\noexpand\dospecials{\dospecials}%
1945     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1946       \def\noexpand\@sanitize{\@sanitize}%
1947     \fi}%
1948   \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).

```

1949 \def\bbl@active@def#1#2#3#4{%
1950   \@namedef{#3#1}{%
1951     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1952       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1953     \else
1954       \bbl@afterfi\csname#2@sh@#1\endcsname
1955     \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.

```

1956   \long\@namedef{#3@arg#1}##1{%
1957     \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1958       \bbl@afterelse\csname#4#1\endcsname##1%
1959     \else
1960       \bbl@afterfi\csname#2@sh@#1@string##1\endcsname
1961     \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.

```
1962 \def\initiate@active@char#1{%
1963   \bbl@ifunset{active@char\string#1}%
1964   {\bbl@withactive
1965     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1966   {}}
```

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).

```
1967 \def\@initiate@active@char#1#2#3{%
1968   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1969   \ifx#1\@undefined
1970     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1971   \else
1972     \bbl@csarg\let{oridef@#2}#1%
1973     \bbl@csarg\edef{oridef@#2}{%
1974       \let\noexpand#1%
1975       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1976   \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*").

```
1977   \ifx#1#3\relax
1978     \expandafter\let\csname normal@char#2\endcsname#3%
1979   \else
1980     \bbl@info{Making #2 an active character}%
1981     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1982     \@namedef{normal@char#2}{%
1983       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1984   \else
1985     \@namedef{normal@char#2}{#3}%
1986   \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).

```
1987   \bbl@restoreactive{#2}%
1988   \AtBeginDocument{%
1989     \catcode`#2\active
1990     \if@filesw
1991       \immediate\write\@mainaux{\catcode`\string#2\active}%
1992     \fi}%
1993   \expandafter\bbl@add@special\csname#2\endcsname
1994   \catcode`#2\active
1995   \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⟩`).

```

1996 \let\bbl@tempa\@firstoftwo
1997 \if\string^#2%
1998 \def\bbl@tempa{\noexpand\textormath}%
1999 \else
2000 \ifx\bbl@mathnormal\@undefined\else
2001 \let\bbl@tempa\bbl@mathnormal
2002 \fi
2003 \fi
2004 \expandafter\edef\csname active@char#2\endcsname{%
2005 \bbl@tempa
2006 {\noexpand\if@safe@actives
2007 \noexpand\expandafter
2008 \expandafter\noexpand\csname normal@char#2\endcsname
2009 \noexpand\else
2010 \noexpand\expandafter
2011 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2012 \noexpand\fi}%
2013 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2014 \bbl@csarg\edef{doactive#2}{%
2015 \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!).

```

2016 \bbl@csarg\edef{active@#2}{%
2017 \noexpand\active@prefix\noexpand#1%
2018 \expandafter\noexpand\csname active@char#2\endcsname}%
2019 \bbl@csarg\edef{normal@#2}{%
2020 \noexpand\active@prefix\noexpand#1%
2021 \expandafter\noexpand\csname normal@char#2\endcsname}%
2022 \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.

```

2023 \bbl@active@def#2\user@group{user@active}{language@active}%
2024 \bbl@active@def#2\language@group{language@active}{system@active}%
2025 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

2026 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2027 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2028 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
2029 {\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.

```
2030 \if\string'#2%
2031 \let\prim@s\bbl@prim@s
2032 \let\active@math@prime#1%
2033 \fi
2034 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
2035 << *More package options >> ≡
2036 \DeclareOption{math=active}{}
2037 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2038 << /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.

```
2039 \@ifpackagewith{babel}{KeepShorthandsActive}%
2040 {\let\bbl@restoreactive\@gobble}%
2041 {\def\bbl@restoreactive#1{%
2042   \bbl@exp{%
2043     \\\AfterBabelLanguage\\CurrentOption
2044     {\catcode`#1=\the\catcode`#1\relax}%
2045     \\\AtEndOfPackage
2046     {\catcode`#1=\the\catcode`#1\relax}}}%
2047   \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.

```
2048 \def\bbl@sh@select#1#2{%
2049   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2050     \bbl@afterelse\bbl@scndcs
2051   \else
2052     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2053   \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.

```
2054 \begingroup
2055 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2056 {\gdef\active@prefix#1{%
2057   \ifx\protect\@typeset@protect
2058     \else
2059       \ifx\protect\@unexpandable@protect
2060         \noexpand#1%
2061       \else
2062         \protect#1%
2063       \fi
```

```

2064     \expandafter\@gobble
2065     \fi}}
2066   {\gdef\active@prefix#1{%
2067     \ifincsname
2068       \string#1%
2069       \expandafter\@gobble
2070     \else
2071       \ifx\protect\@typeset@protect
2072       \else
2073         \ifx\protect\@unexpandable@protect
2074           \noexpand#1%
2075         \else
2076           \protect#1%
2077         \fi
2078       \expandafter\expandafter\expandafter\@gobble
2079     \fi
2080   \fi}}
2081 \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⟩`.

```

2082 \newif\if@safe@actives
2083 \@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.

```

2084 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

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

```

2085 \def\bbl@activate#1{%
2086   \bbl@withactive{\expandafter\let\expandafter}#1%
2087   \csname bbl@active@\string#1\endcsname}
2088 \def\bbl@deactivate#1{%
2089   \bbl@withactive{\expandafter\let\expandafter}#1%
2090   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs 2091 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2092 \def\bbl@scndcs#1#2{\csname#2\endcsname}

```

`\declare@shorthand` The command `\declare@shorthand` is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. `~` or `"a`;
3. the code to be executed when the shorthand is encountered.

The auxiliary macro `\babel@texpdf` improves the interoperativity with `hyperref` and takes 4 arguments: (1) The \TeX code in text mode, (2) the string for `hyperref`, (3) the \TeX code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently `hyperref` doesn’t discriminate the mode). This macro may be used in `ldf` files.

```

2093 \def\babel@texpdf#1#2#3#4{%
2094   \ifx\texorpdfstring\undefined
2095     \textormath{#1}{#2}%
2096   \else
2097     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2098     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2099   \fi}
2100 %
2101 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2102 \def\@decl@short#1#2#3\@nil#4{%
2103   \def\bbl@tempa{#3}%
2104   \ifx\bbl@tempa\@empty
2105     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2106     \bbl@ifunset{#1@sh@\string#2@}{}%
2107     {\def\bbl@tempa{#4}%
2108      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2109      \else
2110        \bbl@info
2111          {Redefining #1 shorthand \string#2\\%
2112           in language \CurrentOption}%
2113      \fi}%
2114     \@namedef{#1@sh@\string#2@}{#4}%
2115   \else
2116     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2117     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2118     {\def\bbl@tempa{#4}%
2119      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2120      \else
2121        \bbl@info
2122          {Redefining #1 shorthand \string#2\string#3\\%
2123           in language \CurrentOption}%
2124      \fi}%
2125     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2126   \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.

```

2127 \def\textormath{%
2128   \ifmmode
2129     \expandafter\@secondoftwo
2130   \else
2131     \expandafter\@firstoftwo
2132   \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’.

```

2133 \def\user@group{user}
2134 \def\language@group{english} % TODO. I don't like defaults
2135 \def\system@group{system}

```

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

```

2136 \def\useshorthands{%

```

```

2137 \@ifstar\bb1@usesesh@s{\bb1@usesesh@x{}}
2138 \def\bb1@usesesh@s#1{%
2139 \bb1@usesesh@x
2140 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
2141 {#1}}
2142 \def\bb1@usesesh@x#1#2{%
2143 \bb1@ifshorthand{#2}%
2144 {\def\user@group{user}%
2145 \initiate@active@char{#2}%
2146 #1%
2147 \bb1@activate{#2}}%
2148 {\bb1@error
2149 {Cannot declare a shorthand turned off (\string#2)}
2150 {Sorry, but you cannot use shorthands which have been\\%
2151 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 `\bb1@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

2152 \def\user@language@group{user@\language@group}
2153 \def\bb1@set@user@generic#1#2{%
2154 \bb1@ifunset{user@generic@active#1}%
2155 {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
2156 \bb1@active@def#1\user@group{user@generic@active}{language@active}%
2157 \expandafter\edef\csname#2@sh#1@@\endcsname{%
2158 \expandafter\noexpand\csname normal@char#1\endcsname}%
2159 \expandafter\edef\csname#2@sh#1@\string\protect\endcsname{%
2160 \expandafter\noexpand\csname user@active#1\endcsname}}%
2161 \@empty}
2162 \newcommand\defineshorthand[3][user]{%
2163 \edef\bb1@tempa{\zap@space#1 \@empty}%
2164 \bb1@for\bb1@tempb\bb1@tempa{%
2165 \if*\expandafter\@car\bb1@tempb\@nil
2166 \edef\bb1@tempb{user\expandafter\@gobble\bb1@tempb}%
2167 \@expandtwoargs
2168 \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
2169 \fi
2170 \declare@shorthand{\bb1@tempb}{#2}{#3}}}

```

`\languageshorthands` A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

```

2171 \def\languageshorthands#1{\def\language@group{#1}}

```

`\aliasshorthand` First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with `\aliasshorthands{"}{/}` is `\active@prefix / \active@char /`, so we still need to let the latest to `\active@char`.

```

2172 \def\aliasshorthand#1#2{%
2173 \bb1@ifshorthand{#2}%
2174 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2175 \ifx\document\@notprerr
2176 \@notshorthand{#2}%
2177 \else
2178 \initiate@active@char{#2}%

```

```

2179 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2180 \csname active@char\string#1\endcsname
2181 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2182 \csname normal@char\string#1\endcsname
2183 \bbl@activate{#2}%
2184 \fi
2185 \fi}%
2186 {\bbl@error
2187 {Cannot declare a shorthand turned off (\string#2)}
2188 {Sorry, but you cannot use shorthands which have been\\%
2189 turned off in the package options}}}

```

\@notshorthand

```

2190 \def\@notshorthand#1{%
2191 \bbl@error{%
2192 The character '\string #1' should be made a shorthand character;\\%
2193 add the command \string\usesshorthands\string{#1\string} to
2194 the preamble.\\%
2195 I will ignore your instruction}%
2196 {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.

```

2197 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2198 \DeclareRobustCommand*\shorthandoff{%
2199 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2200 \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.

```

2201 \def\bbl@switch@sh#1#2{%
2202 \ifx#2\@nnil\else
2203 \bbl@ifunset{\bbl@active@\string#2}%
2204 {\bbl@error
2205 {I cannot switch '\string#2' on or off--not a shorthand}%
2206 {This character is not a shorthand. Maybe you made\\%
2207 a typing mistake? I will ignore your instruction}}}%
2208 {\ifcase#1%
2209 \catcode`#2\relax
2210 \or
2211 \catcode`#2\active
2212 \or
2213 \csname bbl@oricat@\string#2\endcsname
2214 \csname bbl@oridef@\string#2\endcsname
2215 \fi}%
2216 \bbl@afterfi\bbl@switch@sh#1%
2217 \fi}

```

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

```

2218 \def\babelshorthand{\active@prefix\babelshorthand\bb@putsh}
2219 \def\bb@putsh#1{%
2220   \bb@ifunset{bb@active@\string#1}%
2221     {\bb@putsh@i#1\@empty\@nnil}%
2222     {\csname bb@active@\string#1\endcsname}}
2223 \def\bb@putsh@i#1#2\@nnil{%
2224   \csname\language@group @sh@\string#1@%
2225     \ifx\@empty#2\else\string#2\fi\endcsname}
2226 \ifx\bb@opt@shorthands\@nnil\else
2227   \let\bb@s@initiate@active@char\initiate@active@char
2228   \def\initiate@active@char#1{%
2229     \bb@ifshorthand{#1}{\bb@s@initiate@active@char{#1}}{}}
2230   \let\bb@s@switch@sh\bb@switch@sh
2231   \def\bb@switch@sh#1#2{%
2232     \ifx#2\@nnil\else
2233       \bb@afterfi
2234       \bb@ifshorthand{#2}{\bb@s@switch@sh#1{#2}}{\bb@switch@sh#1}%
2235     \fi}
2236   \let\bb@s@activate\bb@activate
2237   \def\bb@activate#1{%
2238     \bb@ifshorthand{#1}{\bb@s@activate{#1}}{}}
2239   \let\bb@s@deactivate\bb@deactivate
2240   \def\bb@deactivate#1{%
2241     \bb@ifshorthand{#1}{\bb@s@deactivate{#1}}{}}
2242 \fi

```

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

```

2243 \newcommand\ifbabelshorthand[3]{\bb@ifunset{bb@active@\string#1}{#3}{#2}}

```

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

```

2244 \def\bb@prim@s{%
2245   \prime\futurelet\@let@token\bb@pr@m@s}
2246 \def\bb@if@primes#1#2{%
2247   \ifx#1\@let@token
2248     \expandafter\@firstoftwo
2249   \else\ifx#2\@let@token
2250     \bb@afterelse\expandafter\@firstoftwo
2251   \else
2252     \bb@afterfi\expandafter\@secondoftwo
2253   \fi\fi}
2254 \begingroup
2255   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2256   \catcode`\'=12 \catcode`\"=\active \lccode`\"='\^
2257   \lowercase{%
2258     \gdef\bb@pr@m@s{%
2259       \bb@if@primes"%
2260         \pr@@@s
2261         {\bb@if@primes*\^{\pr@@@t\egroup}}}
2262 \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).

```
2263 \initiate@active@char{~}
2264 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2265 \bbl@activate{~}
```

\OT1dpos The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.

```
2266 \expandafter\def\csname OT1dpos\endcsname{127}
2267 \expandafter\def\csname T1dpos\endcsname{4}
```

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

```
2268 \ifx\f@encoding\undefined
2269 \def\f@encoding{OT1}
2270 \fi
```

9.6 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```
2271 \bbl@trace{Language attributes}
2272 \newcommand\languageattribute[2]{%
2273 \def\bbl@tempc{#1}%
2274 \bbl@fixname\bbl@tempc
2275 \bbl@iflanguage\bbl@tempc{%
2276 \bbl@vforeach{#2}{%
```

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

```
2277 \ifx\bbl@known@attribs\undefined
2278 \in@false
2279 \else
2280 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2281 \fi
2282 \ifin@
2283 \bbl@warning{%
2284 You have more than once selected the attribute '##1'\%
2285 for language #1. Reported}%
2286 \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.

```
2287 \bbl@exp{%
2288 \\\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2289 \edef\bbl@tempa{\bbl@tempc-##1}%
2290 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2291 {\csname\bbl@tempc @attr##1\endcsname}%
2292 {\@attrerr{\bbl@tempc}{##1}}%
2293 \fi}}
2294 \@onlypreamble\languageattribute
```


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

```
2295 \newcommand*{\@attrerr}[2]{%
2296   \bbl@error
2297   {The attribute #2 is unknown for language #1.}%
2298   {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}`.

```
2299 \def\bbl@declare@ttribute#1#2#3{%
2300   \bbl@xin@{,#2,},{,\BabelModifiers,}%
2301   \ifin@
2302     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2303   \fi
2304   \bbl@add@list\bbl@attributes{#1-#2}%
2305   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret T_EX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```
2306 \def\bbl@ifattributeset#1#2#3#4{%
2307   \ifx\bbl@known@attribs\undefined
2308     \in@false
2309   \else
2310     \bbl@xin@{,#1-#2,},{,\bbl@known@attribs,}%
2311   \fi
2312   \ifin@
2313     \bbl@afterelse#3%
2314   \else
2315     \bbl@afterfi#4%
2316   \fi
2317 }
```

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

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of `\bbl@tempa` is changed. Finally we execute `\bbl@tempa`.

```
2318 \def\bbl@ifknown@trib#1#2{%
2319   \let\bbl@tempa\@secondoftwo
2320   \bbl@loopx\bbl@tempb{#2}{%
2321     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2322   \ifin@
2323     \let\bbl@tempa\@firstoftwo
2324   \else
```

```

2325 \fi}%
2326 \bbl@tempa
2327 }

```

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

```

2328 \def\bbl@clear@ttribs{%
2329 \ifx\bbl@attributes\undefined\else
2330 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2331 \expandafter\bbl@clear@ttrib\bbl@tempa.
2332 }%
2333 \let\bbl@attributes\undefined
2334 \fi}
2335 \def\bbl@clear@ttrib#1-#2.{%
2336 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2337 \AtBeginDocument{\bbl@clear@ttribs}

```

9.7 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.
`\babel@beginsave`

```

2338 \bbl@trace{Macros for saving definitions}
2339 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2340 \newcount\babel@savecnt
2341 \babel@beginsave

```

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

```

2342 \def\babel@save#1{%
2343 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2344 \toks@\expandafter{\originalTeX\let#1=}
2345 \bbl@exp{%
2346 \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2347 \advance\babel@savecnt\@ne}
2348 \def\babel@savevariable#1{%
2349 \toks@\expandafter{\originalTeX #1=}
2350 \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.

```

2351 \def\bbl@frenchspacing{%

```

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

```

2352 \ifnum\the\sfcodes\.\.=\@m
2353 \let\bbl@nonfrenchspacing\relax
2354 \else
2355 \frenchspacing
2356 \let\bbl@nonfrenchspacing\nonfrenchspacing
2357 \fi}
2358 \let\bbl@nonfrenchspacing\nonfrenchspacing
2359 %
2360 \let\bbl@elt\relax
2361 \edef\bbl@fs@chars{%
2362 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2363 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2364 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}

```

9.8 Short tags

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

```

2365 \bbl@trace{Short tags}
2366 \def\babeltags#1{%
2367 \edef\bbl@tempa{\zap@space#1 \@empty}%
2368 \def\bbl@tempb##1=##2\@{#}%
2369 \edef\bbl@tempc{%
2370 \noexpand\newcommand
2371 \expandafter\noexpand\csname ##1\endcsname{%
2372 \noexpand\protect
2373 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2374 \noexpand\newcommand
2375 \expandafter\noexpand\csname text##1\endcsname{%
2376 \noexpand\foreignlanguage{##2}}}%
2377 \bbl@tempc}%
2378 \bbl@for\bbl@tempa\bbl@tempa{%
2379 \expandafter\bbl@tempb\bbl@tempa\@{#}%

```

9.9 Hyphens

`\babelhyphenation` This macro saves hyphenation exceptions. Two macros are used to store them: `\bbl@hyphenation@` for the global ones and `\bbl@hyphenation<lang>` for language ones. See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```

2380 \bbl@trace{Hyphens}
2381 \@onlypreamble\babelhyphenation
2382 \AtEndOfPackage{%
2383 \newcommand\babelhyphenation[2][\@empty]{%
2384 \ifx\bbl@hyphenation@\relax
2385 \let\bbl@hyphenation@\@empty
2386 \fi
2387 \ifx\bbl@hyphlist\@empty\else
2388 \bbl@warning{%
2389 You must not intermingle \string\selectlanguage\space and\%
2390 \string\babelhyphenation\space or some exceptions will not\%
2391 be taken into account. Reported}%
2392 \fi
2393 \ifx\@empty#1%
2394 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2395 \else

```

```

2396 \bbl@vforeach{#1}{%
2397 \def\bbl@tempa{##1}%
2398 \bbl@fixname\bbl@tempa
2399 \bbl@iflanguage\bbl@tempa{%
2400 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2401 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2402 \@empty
2403 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2404 #2}}}%
2405 \fi}}

```

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

```

2406 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
2407 \def\bbl@t@one{T1}
2408 \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`.

```

2409 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2410 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2411 \def\bbl@hyphen{%
2412 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2413 \def\bbl@hyphen@i#1#2{%
2414 \bbl@ifunset{bbl@hy@#1#2\@empty}%
2415 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2416 {\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.

```

2417 \def\bbl@usehyphen#1{%
2418 \leavevmode
2419 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2420 \nobreak\hskip\zskip}
2421 \def\bbl@@usehyphen#1{%
2422 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2423 \def\bbl@hyphenchar{%
2424 \ifnum\hyphenchar\font=\m@ne
2425 \babellnullhyphen
2426 \else
2427 \char\hyphenchar\font
2428 \fi}

```

Finally, we define the hyphen “types”. Their names will not change, so you may use them in `ldf`’s. After a space, the `\mbox` in `\bbl@hy@nobreak` is redundant.

```

2429 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2430 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}}{}}
2431 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}

```

³² $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

2432 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2433 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}}
2434 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2435 \def\bbl@hy@repeat{%
2436   \bbl@usehyphen{%
2437     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}}
2438 \def\bbl@hy@@repeat{%
2439   \bbl@usehyphen{%
2440     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}}
2441 \def\bbl@hy@empty{\hskip\z@skip}
2442 \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.

```

2443 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

9.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by `luatex` and `xetex`. The code is organized here with pseudo-guards, so we start with the basic commands.

Tools But first, a couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```

2444 \bbl@trace{Multiencoding strings}
2445 \def\bbl@tglobal#1{\global\let#1#1}
2446 \def\bbl@recatcode#1{% TODO. Used only once?
2447   \@tempcnta="7F
2448   \def\bbl@tempa{%
2449     \ifnum\@tempcnta>"FF\else
2450       \catcode\@tempcnta=#1\relax
2451       \advance\@tempcnta\@ne
2452       \expandafter\bbl@tempa
2453     \fi}%
2454   \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).

```

2455 \@ifpackagewith{babel}{nocase}%
2456   {\let\bbl@patchuclc\relax}%
2457   {\def\bbl@patchuclc{%
2458     \global\let\bbl@patchuclc\relax
2459     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}}%
2460     \gdef\bbl@uclc##1{%
2461       \let\bbl@encoded\bbl@encoded@uclc
2462       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2463       {##1}%

```

```

2464      {\let\bbl@tempa##1\relax % Used by LANG@bbl@uc1c
2465       \csname\language @bbl@uc1c\endcsname}%
2466      {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2467      \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2468      \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}%
2469 <<(*More package options)>> ≡
2470 \DeclareOption{nocase}{}
2471 <</More package options>>

```

The following package options control the behavior of \SetString.

```

2472 <<(*More package options)>> ≡
2473 \let\bbl@opt@strings\@nnil % accept strings=value
2474 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2475 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2476 \def\BabelStringsDefault{generic}
2477 <</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.

```

2478 \@onlypreamble\StartBabelCommands
2479 \def\StartBabelCommands{%
2480   \begingroup
2481   \bbl@recatcode{11}%
2482   <<Macros local to BabelCommands>>
2483   \def\bbl@provstring##1##2{%
2484     \providecommand##1{##2}%
2485     \bbl@tglobal##1}%
2486   \global\let\bbl@scafter\@empty
2487   \let\StartBabelCommands\bbl@startcmds
2488   \ifx\BabelLanguages\relax
2489     \let\BabelLanguages\CurrentOption
2490   \fi
2491   \begingroup
2492   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2493   \StartBabelCommands}
2494 \def\bbl@startcmds{%
2495   \ifx\bbl@screset\@nnil\else
2496     \bbl@usehooks{stopcommands}{}%
2497   \fi
2498   \endgroup
2499   \begingroup
2500   \@ifstar
2501     {\ifx\bbl@opt@strings\@nnil
2502       \let\bbl@opt@strings\BabelStringsDefault
2503       \fi
2504       \bbl@startcmds@i}%
2505     \bbl@startcmds@i}
2506 \def\bbl@startcmds@i#1#2{%
2507   \edef\bbl@L{\zap@space#1 \@empty}%
2508   \edef\bbl@G{\zap@space#2 \@empty}%
2509   \bbl@startcmds@ii}
2510 \let\bbl@startcmds\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled

blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

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

```

2511 \newcommand\bb1@startcmds@ii[1][\@empty]{%
2512   \let\SetString\@gobbletwo
2513   \let\bb1@stringdef\@gobbletwo
2514   \let\AfterBabelCommands\@gobble
2515   \ifx\@empty#1%
2516     \def\bb1@sc@label{generic}%
2517     \def\bb1@encstring##1##2{%
2518       \ProvideTextCommandDefault##1{##2}%
2519       \bb1@tglobal##1%
2520       \expandafter\bb1@tglobal\csname\string?\string##1\endcsname}%
2521     \let\bb1@sctest\in@true
2522   \else
2523     \let\bb1@sc@charset\space % <- zapped below
2524     \let\bb1@sc@fontenc\space % <- " "
2525     \def\bb1@tempa##1=##2\@nil{%
2526       \bb1@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2527     \bb1@foreach{label=#1}{\bb1@tempa##1\@nil}%
2528     \def\bb1@tempa##1 ##2{% space -> comma
2529       ##1%
2530       \ifx\@empty##2\else\ifx,##1,\else,\fi\bb1@afterfi\bb1@tempa##2\fi}%
2531     \edef\bb1@sc@fontenc{\expandafter\bb1@tempa\bb1@sc@fontenc\@empty}%
2532     \edef\bb1@sc@label{\expandafter\zap@space\bb1@sc@label\@empty}%
2533     \edef\bb1@sc@charset{\expandafter\zap@space\bb1@sc@charset\@empty}%
2534     \def\bb1@encstring##1##2{%
2535       \bb1@foreach\bb1@sc@fontenc{%
2536         \bb1@ifunset{T@####1}%
2537         {}%
2538         {\ProvideTextCommand##1{####1}{##2}%
2539         \bb1@tglobal##1%
2540         \expandafter
2541         \bb1@tglobal\csname####1\string##1\endcsname}}}%
2542     \def\bb1@sctest{%
2543       \bb1@xin@{\bb1@opt@strings,}{,\bb1@sc@label,\bb1@sc@fontenc,}}%
2544   \fi
2545   \ifx\bb1@opt@strings\@nnil % ie, no strings key -> defaults
2546   \else\ifx\bb1@opt@strings\relax % ie, strings=encoded
2547     \let\AfterBabelCommands\bb1@aftercmds
2548     \let\SetString\bb1@setstring
2549     \let\bb1@stringdef\bb1@encstring
2550   \else % ie, strings=value
2551     \bb1@sctest
2552   \fin@
2553     \let\AfterBabelCommands\bb1@aftercmds
2554     \let\SetString\bb1@setstring
2555     \let\bb1@stringdef\bb1@provstring
2556   \fi\fi\fi
2557   \bb1@scswitch
2558   \ifx\bb1@G\@empty
2559     \def\SetString##1##2{%
2560       \bb1@error{Missing group for string \string##1}%
2561       {You must assign strings to some category, typically\\%
2562       captions or extras, but you set none}}%

```

```

2563 \fi
2564 \ifx\@empty#1%
2565 \bbl@usehooks{defaultcommands}{}%
2566 \else
2567 \@expandtwoargs
2568 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2569 \fi}

```

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

The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside babel) or `\date \langle language \rangle` is defined (after babel has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in ldfs), and the second one skips undefined languages (after babel has been loaded) .

```

2570 \def\bbl@forlang#1#2{%
2571 \bbl@for#1\bbl@L{%
2572 \bbl@xin@{,#1,},{,\BabelLanguages,}%
2573 \ifin@#2\relax\fi}}
2574 \def\bbl@scswitch{%
2575 \bbl@forlang\bbl@tempa{%
2576 \ifx\bbl@G\@empty\else
2577 \ifx\SetString\@gobbletwo\else
2578 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2579 \bbl@xin@{,\bbl@GL,},{,\bbl@screset,}%
2580 \ifin@\else
2581 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2582 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2583 \fi
2584 \fi
2585 \fi}}
2586 \AtEndOfPackage{%
2587 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
2588 \let\bbl@scswitch\relax}
2589 \@onlypreamble\EndBabelCommands
2590 \def\EndBabelCommands{%
2591 \bbl@usehooks{stopcommands}{}%
2592 \endgroup
2593 \endgroup
2594 \bbl@scafter}
2595 \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.

```

2596 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2597 \bbl@forlang\bbl@tempa{%
2598 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2599 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2600 {\bbl@exp{%
2601 \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%

```



```

2602     {}%
2603     \def\BabelString{#2}%
2604     \bbl@usehooks{stringprocess}{}%
2605     \expandafter\bbl@stringdef
2606     \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`.

```

2607 \ifx\bbl@opt@strings\relax
2608   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2609   \bbl@patchuclc
2610   \let\bbl@encoded\relax
2611   \def\bbl@encoded@uclc#1{%
2612     \@inmathwarn#1%
2613     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2614       \expandafter\ifx\csname ?\string#1\endcsname\relax
2615         \TextSymbolUnavailable#1%
2616       \else
2617         \csname ?\string#1\endcsname
2618       \fi
2619     \else
2620       \csname\cf@encoding\string#1\endcsname
2621     \fi}
2622 \else
2623   \def\bbl@scset#1#2{\def#1{#2}}
2624 \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.

```

2625 <<*Macros local to BabelCommands>> ≡
2626 \def\SetStringLoop##1##2{%
2627   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2628   \count@\z@
2629   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2630     \advance\count@\@ne
2631     \toks@\expandafter{\bbl@tempa}%
2632     \bbl@exp{%
2633       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2634       \count@=\the\count@\relax}}}%
2635 <</Macros local to BabelCommands>>

```

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

```

2636 \def\bbl@aftercmds#1{%
2637   \toks@\expandafter{\bbl@scafter#1}%
2638   \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.

```

2639 <<*Macros local to BabelCommands>> ≡
2640 \newcommand\SetCase[3][{}%
2641   \bbl@patchuclc

```

```

2642 \bbl@forlang\bbl@tempa{%
2643 \expandafter\bbl@encstring
2644 \csname\bbl@tempa @bbl@ucl\endcsname{\bbl@tempa##1}%
2645 \expandafter\bbl@encstring
2646 \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2647 \expandafter\bbl@encstring
2648 \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2649 <</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.

```

2650 <<*Macros local to BabelCommands>> ≡
2651 \newcommand\SetHyphenMap[1]{%
2652 \bbl@forlang\bbl@tempa{%
2653 \expandafter\bbl@stringdef
2654 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2655 <</Macros local to BabelCommands>>

```

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

```

2656 \newcommand\BabelLower[2]{% one to one.
2657 \ifnum\lccode#1=#2\else
2658 \babel@savevariable{\lccode#1}%
2659 \lccode#1=#2\relax
2660 \fi}
2661 \newcommand\BabelLowerMM[4]{% many-to-many
2662 \@tempcnta=#1\relax
2663 \@tempcntb=#4\relax
2664 \def\bbl@tempa{%
2665 \ifnum\@tempcnta>#2\else
2666 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2667 \advance\@tempcnta#3\relax
2668 \advance\@tempcntb#3\relax
2669 \expandafter\bbl@tempa
2670 \fi}%
2671 \bbl@tempa}
2672 \newcommand\BabelLowerMO[4]{% many-to-one
2673 \@tempcnta=#1\relax
2674 \def\bbl@tempa{%
2675 \ifnum\@tempcnta>#2\else
2676 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2677 \advance\@tempcnta#3
2678 \expandafter\bbl@tempa
2679 \fi}%
2680 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2681 <<*More package options>> ≡
2682 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2683 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2684 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2685 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2686 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2687 <</More package options>>

```

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

```

2688 \AtEndOfPackage{%
2689 \ifx\bbl@opt@hyphenmap\undefined
2690 \bbl@xin@{,}{\bbl@language@opts}%

```

```

2691 \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2692 \fi}

```

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

```

2693 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2694 \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2695 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2696 \edef\bbl@tempa{#1}%
2697 \edef\bbl@tempd{%
2698 \expandafter\expandafter\expandafter
2699 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2700 \bbl@xin@
2701 {\expandafter\string\csname #2name\endcsname}%
2702 {\bbl@tempd}%
2703 \ifin@ % Renew caption
2704 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2705 \ifin@
2706 \bbl@exp{%
2707 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2708 {\\\bbl@scset\<#2name>\<#1#2name>}%
2709 {}}%
2710 \else % Old way converts to new way
2711 \bbl@ifunset{#1#2name}%
2712 {\bbl@exp{%
2713 \\\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2714 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2715 {\def\<#2name>\<#1#2name>}}%
2716 {}}}%
2717 {}%
2718 \fi
2719 \else
2720 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2721 \ifin@ % New way
2722 \bbl@exp{%
2723 \\\bbl@add\<captions#1>\\\bbl@scset\<#2name>\<#1#2name>}%
2724 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2725 {\\\bbl@scset\<#2name>\<#1#2name>}%
2726 {}}%
2727 \else % Old way, but defined in the new way
2728 \bbl@exp{%
2729 \\\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2730 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2731 {\def\<#2name>\<#1#2name>}}%
2732 {}}%
2733 \fi%
2734 \fi
2735 \@namedef{#1#2name}{#3}%
2736 \toks@ \expandafter{\bbl@captionslist}%
2737 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2738 \ifin@ \else
2739 \bbl@exp{\\\bbl@add\\bbl@captionslist{\<#2name>}}%
2740 \bbl@tglobal\bbl@captionslist
2741 \fi}
2742 % \def\bbl@setcaption@s#1#2#3{} % Not yet implemented

```

9.11 Macros common to a number of languages

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

```
2743 \bbl@trace{Macros related to glyphs}
2744 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2745   \dimen\z@ht\z@ \advance\dimen\z@ -ht\tw@%
2746   \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.

```
2747 \def\save@sf@q#1{\leavevmode
2748   \begingroup
2749   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2750   \endgroup}
```

9.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through `T1enc.def`.

9.12.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```
2751 \ProvideTextCommand{\quotedblbase}{OT1}{%
2752   \save@sf@q{\set@low@box{\textquotedblright\}%
2753     \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2754 \ProvideTextCommandDefault{\quotedblbase}{%
2755   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2756 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2757   \save@sf@q{\set@low@box{\textquoteright\}%
2758     \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2759 \ProvideTextCommandDefault{\quotesinglbase}{%
2760   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2761 \ProvideTextCommand{\guillemetleft}{OT1}{%
2762   \ifmmode
2763     \ll
2764   \else
2765     \save@sf@q{\nobreak
2766       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2767   \fi}
2768 \ProvideTextCommand{\guillemetright}{OT1}{%
2769   \ifmmode
2770     \gg
```

```

2771 \else
2772   \save@sf@q{\nobreak
2773     \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2774   \fi}
2775 \ProvideTextCommand{\guillemotleft}{OT1}{%
2776   \ifmmode
2777     \ll
2778   \else
2779     \save@sf@q{\nobreak
2780       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2781     \fi}
2782 \ProvideTextCommand{\guillemotright}{OT1}{%
2783   \ifmmode
2784     \gg
2785   \else
2786     \save@sf@q{\nobreak
2787       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2788     \fi}

```

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

```

2789 \ProvideTextCommandDefault{\guillemetleft}{%
2790   \UseTextSymbol{OT1}{\guillemetleft}}
2791 \ProvideTextCommandDefault{\guillemetright}{%
2792   \UseTextSymbol{OT1}{\guillemetright}}
2793 \ProvideTextCommandDefault{\guillemotleft}{%
2794   \UseTextSymbol{OT1}{\guillemotleft}}
2795 \ProvideTextCommandDefault{\guillemotright}{%
2796   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2797 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2798   \ifmmode
2799     <%
2800   \else
2801     \save@sf@q{\nobreak
2802       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2803     \fi}
2804 \ProvideTextCommand{\guilsinglright}{OT1}{%
2805   \ifmmode
2806     >%
2807   \else
2808     \save@sf@q{\nobreak
2809       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2810     \fi}

```

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

```

2811 \ProvideTextCommandDefault{\guilsinglleft}{%
2812   \UseTextSymbol{OT1}{\guilsinglleft}}
2813 \ProvideTextCommandDefault{\guilsinglright}{%
2814   \UseTextSymbol{OT1}{\guilsinglright}}

```

9.12.2 Letters

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

```

2815 \DeclareTextCommand{\ij}{OT1}{%

```

```

2816 i\kern-0.02em\bbl@allowhyphens j}
2817 \DeclareTextCommand{\IJ}{OT1}{%
2818 I\kern-0.02em\bbl@allowhyphens J}
2819 \DeclareTextCommand{\ij}{T1}{\char188}
2820 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2821 \ProvideTextCommandDefault{\ij}{%
2822 \UseTextSymbol{OT1}{\ij}}
2823 \ProvideTextCommandDefault{\IJ}{%
2824 \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).

```

2825 \def\crrtic@{\hrule height0.1ex width0.3em}
2826 \def\crttic@{\hrule height0.1ex width0.33em}
2827 \def\ddj@{%
2828 \setbox0\hbox{d}\dimen@=\ht0
2829 \advance\dimen@1ex
2830 \dimen@.45\dimen@
2831 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2832 \advance\dimen@ii.5ex
2833 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2834 \def\DDJ@{%
2835 \setbox0\hbox{D}\dimen@=.55\ht0
2836 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2837 \advance\dimen@ii.15ex % correction for the dash position
2838 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2839 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2840 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2841 %
2842 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2843 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2844 \ProvideTextCommandDefault{\dj}{%
2845 \UseTextSymbol{OT1}{\dj}}
2846 \ProvideTextCommandDefault{\DJ}{%
2847 \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.

```

2848 \DeclareTextCommand{\SS}{OT1}{SS}
2849 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

9.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with `\ProvideTextCommandDefault`, but this is very likely not required because their definitions are based on encoding-dependent macros.

`\glq` The ‘german’ single quotes.

```
\grq 2850 \ProvideTextCommandDefault{\glq}{%
2851 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2852 \ProvideTextCommand{\grq}{T1}{%
2853 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2854 \ProvideTextCommand{\grq}{TU}{%
2855 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2856 \ProvideTextCommand{\grq}{OT1}{%
2857 \save@sf@q{\kern-.0125em
2858 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2859 \kern.07em\relax}}
2860 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2861 \ProvideTextCommandDefault{\glqq}{%
2862 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2863 \ProvideTextCommand{\grqq}{T1}{%
2864 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2865 \ProvideTextCommand{\grqq}{TU}{%
2866 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2867 \ProvideTextCommand{\grqq}{OT1}{%
2868 \save@sf@q{\kern-.07em
2869 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2870 \kern.07em\relax}}
2871 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2872 \ProvideTextCommandDefault{\flq}{%
2873 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2874 \ProvideTextCommandDefault{\frq}{%
2875 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2876 \ProvideTextCommandDefault{\flqq}{%
2877 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2878 \ProvideTextCommandDefault{\frqq}{%
2879 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

9.12.4 Umlauts and tremas

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

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

```
2880 \def\umlauthigh{%
2881 \def\bb1@umlauta##1{\leavevmode\bggroup%
2882 \expandafter\accent\csname\f@encoding dqpos\endcsname
```

```

2883      ##1\bb1@allowhyphens\egroup}%
2884 \let\bb1@umlaut\bb1@umlauta}
2885 \def\umlautlow{%
2886 \def\bb1@umlauta{\protect\lower@umlaut}}
2887 \def\umlautelow{%
2888 \def\bb1@umlaute{\protect\lower@umlaut}}
2889 \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.

```

2890 \expandafter\ifx\csname U@D\endcsname\relax
2891 \csname newdimen\endcsname\U@D
2892 \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.

```

2893 \def\lower@umlaut#1{%
2894 \leavevmode\bgroup
2895 \U@D 1ex%
2896 {\setbox\z@\hbox{%
2897 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2898 \dimen@ -.45ex\advance\dimen@\ht\z@
2899 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2900 \expandafter\accent\csname\fontencoding dqpos\endcsname
2901 \fontdimen5\font\U@D #1%
2902 \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bb1@umlauta` or `\bb1@umlaute` to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package `fontenc` with option `OT1` is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but `babel` sets them for *all* languages – you may want to redefine `\bb1@umlauta` and/or `\bb1@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2903 \AtBeginDocument{%
2904 \DeclareTextCompositeCommand{\}{OT1}{a}{\bb1@umlauta{a}}%
2905 \DeclareTextCompositeCommand{\}{OT1}{e}{\bb1@umlaute{e}}%
2906 \DeclareTextCompositeCommand{\}{OT1}{i}{\bb1@umlaute{i}}%
2907 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bb1@umlaute{i}}%
2908 \DeclareTextCompositeCommand{\}{OT1}{o}{\bb1@umlauta{o}}%
2909 \DeclareTextCompositeCommand{\}{OT1}{u}{\bb1@umlauta{u}}%
2910 \DeclareTextCompositeCommand{\}{OT1}{A}{\bb1@umlauta{A}}%
2911 \DeclareTextCompositeCommand{\}{OT1}{E}{\bb1@umlaute{E}}%
2912 \DeclareTextCompositeCommand{\}{OT1}{I}{\bb1@umlaute{I}}%
2913 \DeclareTextCompositeCommand{\}{OT1}{O}{\bb1@umlauta{O}}%
2914 \DeclareTextCompositeCommand{\}{OT1}{U}{\bb1@umlauta{U}}%

```

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


```

2915 \ifx\l@english\@undefined
2916   \chardef\l@english\z@
2917 \fi
2918 % The following is used to cancel rules in ini files (see Amharic).
2919 \ifx\l@babelnohyphens\@undefined
2920   \newlanguage\l@babelnohyphens
2921 \fi

```

9.13 Layout

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

```

2922 \bbl@trace{Bidi layout}
2923 \providecommand\IfBabelLayout[3]{#3}%
2924 \newcommand\BabelPatchSection[1]{%
2925   \@ifundefined{#1}{}{%
2926     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2927     \@namedef{#1}{%
2928       \ifstar{\bbl@presec@#1}%
2929       {\@dblarg{\bbl@presec@#1}}}%
2930 \def\bbl@presec@#1[#2]#3{%
2931   \bbl@exp{%
2932     \\\select@language@x{\bbl@main@language}%
2933     \\\bbl@cs{sspre@#1}%
2934     \\\bbl@cs{ss@#1}%
2935     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2936     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2937     \\\select@language@x{\language}}}%
2938 \def\bbl@presec@#1#2{%
2939   \bbl@exp{%
2940     \\\select@language@x{\bbl@main@language}%
2941     \\\bbl@cs{sspre@#1}%
2942     \\\bbl@cs{ss@#1}*%
2943     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2944     \\\select@language@x{\language}}}%
2945 \IfBabelLayout{sectioning}%
2946   {\BabelPatchSection{part}%
2947    \BabelPatchSection{chapter}%
2948    \BabelPatchSection{section}%
2949    \BabelPatchSection{subsection}%
2950    \BabelPatchSection{subsubsection}%
2951    \BabelPatchSection{paragraph}%
2952    \BabelPatchSection{subparagraph}%
2953    \def\babel@toc#1{%
2954      \select@language@x{\bbl@main@language}}}%
2955 \IfBabelLayout{captions}%
2956   {\BabelPatchSection{caption}}}%

```

9.14 Load engine specific macros

```

2957 \bbl@trace{Input engine specific macros}
2958 \ifcase\bbl@engine
2959   \input txtbabel.def
2960 \or
2961   \input luababel.def
2962 \or
2963   \input xebabel.def
2964 \fi

```

9.15 Creating and modifying languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```
2965 \bbl@trace{Creating languages and reading ini files}
2966 \newcommand\babelprovide[2][{}]{%
2967   \let\bbl@savelangname\language
2968   \edef\bbl@savelocaleid{\the\localeid}%
2969   % Set name and locale id
2970   \edef\language{#2}%
2971   % \global\@namedef{\bbl@lcname#2}{#2}%
2972   \bbl@id@assign
2973   \let\bbl@KVP@captions\@nil
2974   \let\bbl@KVP@date\@nil
2975   \let\bbl@KVP@import\@nil
2976   \let\bbl@KVP@main\@nil
2977   \let\bbl@KVP@script\@nil
2978   \let\bbl@KVP@language\@nil
2979   \let\bbl@KVP@hyphenrules\@nil
2980   \let\bbl@KVP@mapfont\@nil
2981   \let\bbl@KVP@maparabic\@nil
2982   \let\bbl@KVP@mapdigits\@nil
2983   \let\bbl@KVP@intraspace\@nil
2984   \let\bbl@KVP@intrapenalty\@nil
2985   \let\bbl@KVP@onchar\@nil
2986   \let\bbl@KVP@alph\@nil
2987   \let\bbl@KVP@Alph\@nil
2988   \let\bbl@KVP@labels\@nil
2989   \bbl@csarg\let{KVP@labels*}\@nil
2990   \bbl@forkv{#1}{% TODO - error handling
2991     \in@{/}{##1}%
2992     \ifin@
2993       \bbl@renewinikey##1\@{##2}%
2994     \else
2995       \bbl@csarg\def{KVP@##1}{##2}%
2996     \fi}%
2997   % == import, captions ==
2998   \ifx\bbl@KVP@import\@nil\else
2999     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
3000     {\ifx\bbl@initoload\relax
3001       \begingroup
3002         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
3003         \bbl@input@texini{##2}%
3004       \endgroup
3005     \else
3006       \xdef\bbl@KVP@import{\bbl@initoload}%
3007     \fi}%
3008   {}%
3009 \fi
3010 \ifx\bbl@KVP@captions\@nil
3011   \let\bbl@KVP@captions\bbl@KVP@import
3012 \fi
3013 % Load ini
3014 \bbl@ifunset{date#2}%
3015   {\bbl@provide@new{#2}}%
3016   {\bbl@ifblank{#1}%
3017     {\bbl@error
```

```

3018         {If you want to modify `#2' you must tell how in\\%
3019         the optional argument. See the manual for the\\%
3020         available options.}%
3021         {Use this macro as documented}}%
3022         {\bbl@provide@renew{#2}}}%
3023 % Post tasks
3024 \bbl@ifunset{\bbl@extracaps@#2}%
3025     {\bbl@exp{\babelensure[exclude=\\today]{#2}}}%
3026     {\toks@\\expandafter\\expandafter\\expandafter
3027     {\csname bbl@extracaps@#2\endcsname}%
3028     \bbl@exp{\babelensure[exclude=\\today,include=\\the\toks@]{#2}}}%
3029 \bbl@ifunset{\bbl@ensure@\\language}%
3030     {\bbl@exp{%
3031         \\DeclareRobustCommand<\bbl@ensure@\\language>[1]{%
3032         \\foreignlanguage{\\language}%
3033         {###1}}}%
3034     }%
3035 \bbl@exp{%
3036     \\bbl@tglobal\\<\bbl@ensure@\\language>%
3037     \\bbl@tglobal\\<\bbl@ensure@\\language\\space>%
3038 % At this point all parameters are defined if 'import'. Now we
3039 % execute some code depending on them. But what about if nothing was
3040 % imported? We just load the very basic parameters.
3041 \bbl@load@basic{#2}%
3042 % == script, language ==
3043 % Override the values from ini or defines them
3044 \ifx\bbl@KVP@script\\nil\\else
3045     \bbl@csarg\\edef{sname@#2}{\bbl@KVP@script}%
3046 \fi
3047 \ifx\bbl@KVP@language\\nil\\else
3048     \bbl@csarg\\edef{lname@#2}{\bbl@KVP@language}%
3049 \fi
3050 % == onchar ==
3051 \ifx\bbl@KVP@onchar\\nil\\else
3052     \bbl@luahyphenate
3053     \directlua{
3054         if Babel.locale_mapped == nil then
3055             Babel.locale_mapped = true
3056             Babel.linebreaking.add_before(Babel.locale_map)
3057             Babel.loc_to_scr = {}
3058             Babel.chr_to_loc = Babel.chr_to_loc or {}
3059         end}%
3060 \bbl@xin@{ ids }{ \bbl@KVP@onchar\\space}%
3061 \ifin@
3062     \ifx\bbl@starthyphens\\undefined % Needed if no explicit selection
3063         \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
3064     \fi
3065     \bbl@exp{\bbl@add\bbl@starthyphens
3066     {\bbl@patterns@lua{\\language}}}%
3067 % TODO - error/warning if no script
3068     \directlua{
3069         if Babel.script_blocks['\bbl@cl{sbcp}'] then
3070             Babel.loc_to_scr[\the\\localeid] =
3071             Babel.script_blocks['\bbl@cl{sbcp}']
3072             Babel.locale_props[\the\\localeid].lc = \the\\localeid\\space
3073             Babel.locale_props[\the\\localeid].lg = \the\\nameuse{1@\\language}\\space
3074         end
3075     }%
3076 \fi

```

```

3077 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3078 \ifin@
3079 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
3080 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
3081 \directlua{
3082   if Babel.script_blocks['\bbl@cl{sbc}'] then
3083     Babel.loc_to_scr[\the\localeid] =
3084       Babel.script_blocks['\bbl@cl{sbc}']
3085   end}%
3086 \ifx\bbl@mapselect\undefined
3087   \AtBeginDocument{%
3088     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3089     {\selectfont}}%
3090   \def\bbl@mapselect{%
3091     \let\bbl@mapselect\relax
3092     \edef\bbl@prefontid{\fontid\font}}%
3093   \def\bbl@mapdir##1{%
3094     {\def\language{##1}%
3095      \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3096      \bbl@switchfont
3097      \directlua{
3098        Babel.locale_props[\the\csname bbl@id@##1\endcsname]
3099          [\bbl@prefontid] = \fontid\font\space}}}%
3100   \fi
3101   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3102   \fi
3103   % TODO - catch non-valid values
3104 \fi
3105 % == mapfont ==
3106 % For bidi texts, to switch the font based on direction
3107 \ifx\bbl@KVP@mapfont\@nil\else
3108   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
3109   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\the
3110     mapfont. Use 'direction'.%
3111     {See the manual for details.}}}%
3112   \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
3113   \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
3114   \ifx\bbl@mapselect\undefined
3115     \AtBeginDocument{%
3116       \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3117       {\selectfont}}%
3118     \def\bbl@mapselect{%
3119       \let\bbl@mapselect\relax
3120       \edef\bbl@prefontid{\fontid\font}}%
3121     \def\bbl@mapdir##1{%
3122       {\def\language{##1}%
3123        \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3124        \bbl@switchfont
3125        \directlua{Babel.fontmap
3126          [\the\csname bbl@wdir@##1\endcsname]
3127            [\bbl@prefontid]=\fontid\font}}}%
3128     \fi
3129     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3130   \fi
3131   % == Line breaking: intraspace, intrapenalty ==
3132   % For CJK, East Asian, Southeast Asian, if interspace in ini
3133   \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3134     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3135   \fi

```

```

3136 \bbl@provide@intraspace
3137 % == Line breaking: hyphenate.other.locale ==
3138 \bbl@ifunset{bbl@hyotl@languagename}{}%
3139 {\bbl@csarg\bbl@replace{hyotl@languagename}{ }{,}%
3140 \bbl@startcommands*{languagename}{}%
3141 \bbl@csarg\bbl@foreach{hyotl@languagename}{%
3142 \ifcase\bbl@engine
3143 \ifnum##1<257
3144 \SetHyphenMap{BabelLower{##1}{##1}}%
3145 \fi
3146 \else
3147 \SetHyphenMap{BabelLower{##1}{##1}}%
3148 \fi}%
3149 \bbl@endcommands}%
3150 % == Line breaking: hyphenate.other.script ==
3151 \bbl@ifunset{bbl@hyots@languagename}{}%
3152 {\bbl@csarg\bbl@replace{hyots@languagename}{ }{,}%
3153 \bbl@csarg\bbl@foreach{hyots@languagename}{%
3154 \ifcase\bbl@engine
3155 \ifnum##1<257
3156 \global\lccode##1=##1\relax
3157 \fi
3158 \else
3159 \global\lccode##1=##1\relax
3160 \fi}}%
3161 % == Counters: maparabic ==
3162 % Native digits, if provided in ini (TeX level, xe and lua)
3163 \ifcase\bbl@engine\else
3164 \bbl@ifunset{bbl@dgnat@languagename}{}%
3165 {\expandafter\ifx\csname bbl@dgnat@languagename\endcsname\@empty\else
3166 \expandafter\expandafter\expandafter
3167 \bbl@setdigits\csname bbl@dgnat@languagename\endcsname
3168 \ifx\bbl@KVP@maparabic\@nil\else
3169 \ifx\bbl@latinarabic\@undefined
3170 \expandafter\let\expandafter\@arabic
3171 \csname bbl@counter@languagename\endcsname
3172 \else % ie, if layout=counters, which redefines \@arabic
3173 \expandafter\let\expandafter\bbl@latinarabic
3174 \csname bbl@counter@languagename\endcsname
3175 \fi
3176 \fi
3177 \fi}%
3178 \fi
3179 % == Counters: mapdigits ==
3180 % Native digits (lua level).
3181 \ifodd\bbl@engine
3182 \ifx\bbl@KVP@mapdigits\@nil\else
3183 \bbl@ifunset{bbl@dgnat@languagename}{}%
3184 {\RequirePackage{luatexbase}%
3185 \bbl@activate@preotf
3186 \directlua{
3187 Babel = Babel or {} %% -> presets in luababel
3188 Babel.digits_mapped = true
3189 Babel.digits = Babel.digits or {}
3190 Babel.digits[\the\localeid] =
3191 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3192 if not Babel.numbers then
3193 function Babel.numbers(head)
3194 local LOCALE = luatexbase.registernumber'bbl@attr@locale'

```

```

3195         local GLYPH = node.id'glyph'
3196         local inmath = false
3197         for item in node.traverse(head) do
3198             if not inmath and item.id == GLYPH then
3199                 local temp = node.get_attribute(item, LOCALE)
3200                 if Babel.digits[temp] then
3201                     local chr = item.char
3202                     if chr > 47 and chr < 58 then
3203                         item.char = Babel.digits[temp][chr-47]
3204                     end
3205                 end
3206             elseif item.id == node.id'math' then
3207                 inmath = (item.subtype == 0)
3208             end
3209         end
3210         return head
3211     end
3212 end
3213 }}%
3214 \fi
3215 \fi
3216 % == Counters: alph, Alph ==
3217 % What if extras<lang> contains a \babel@save\@alph? It won't be
3218 % restored correctly when exiting the language, so we ignore
3219 % this change with the \bbl@alph@saved trick.
3220 \ifx\bbl@KVP@alph\@nil\else
3221     \toks@\expandafter\expandafter\expandafter{%
3222         \csname extras\language\endcsname}%
3223     \bbl@exp{%
3224         \def\<extras\language>{%
3225             \let\\bbl@alph@saved\\@alph
3226             \the\toks@
3227             \let\\@alph\\bbl@alph@saved
3228             \\babel@save\\@alph
3229             \let\\@alph<bbl@cntr@\bbl@KVP@alph @\language>}}%
3230 \fi
3231 \ifx\bbl@KVP@Alph\@nil\else
3232     \toks@\expandafter\expandafter\expandafter{%
3233         \csname extras\language\endcsname}%
3234     \bbl@exp{%
3235         \def\<extras\language>{%
3236             \let\\bbl@Alph@saved\\@Alph
3237             \the\toks@
3238             \let\\@Alph\\bbl@Alph@saved
3239             \\babel@save\\@Alph
3240             \let\\@Alph<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3241 \fi
3242 % == require.babel in ini ==
3243 % To load or reload the babel-*.tex, if require.babel in ini
3244 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3245     \bbl@ifunset{bbl@rtex@\language}%
3246     {\expandafter\ifx\csname bbl@rtex@\language\endcsname\@empty\else
3247         \let\BabelBeforeIni\@gobbletwo
3248         \chardef\atcatcode=\catcode`\@
3249         \catcode`\@=11\relax
3250         \bbl@input@texini{\bbl@cs{rtex@\language}}%
3251         \catcode`\@=\atcatcode
3252         \let\atcatcode\relax
3253     \fi}%

```

```

3254 \fi
3255 % == main ==
3256 \ifx\bbbl@KVP@main\@nil % Restore only if not 'main'
3257 \let\language\bbbl@savelangname
3258 \chardef\localeid\bbbl@savelocaleid\relax
3259 \fi}

```

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

```

3260 \def\bbbl@provide@new#1{%
3261 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3262 \namedef{extras#1}{}%
3263 \namedef{noextras#1}{}%
3264 \bbbl@startcommands*{#1}{captions}%
3265 \ifx\bbbl@KVP@captions\@nil % and also if import, implicit
3266 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
3267 \ifx##1\@empty\else
3268 \bbbl@exp{%
3269 \SetString\##1{%
3270 \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
3271 \expandafter\bbbl@tempb
3272 \fi}%
3273 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
3274 \else
3275 \ifx\bbbl@initoload\relax
3276 \bbbl@read@ini{\bbbl@KVP@captions}0% Here letters cat = 11
3277 \else
3278 \bbbl@read@ini{\bbbl@initoload}0% Here all letters cat = 11
3279 \fi
3280 \bbbl@after@ini
3281 \bbbl@savestrings
3282 \fi
3283 \StartBabelCommands*{#1}{date}%
3284 \ifx\bbbl@KVP@import\@nil
3285 \bbbl@exp{%
3286 \SetString\today{\bbbl@nocaption{today}{#1today}}}%
3287 \else
3288 \bbbl@savetoday
3289 \bbbl@savedate
3290 \fi
3291 \bbbl@endcommands
3292 \bbbl@load@basic{#1}%
3293 % == hyphenmins == (only if new)
3294 \bbbl@exp{%
3295 \gdef\<#1hyphenmins>{%
3296 {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
3297 {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}}}%
3298 % == hyphenrules ==
3299 \bbbl@provide@hyphens{#1}%
3300 % == frenchspacing == (only if new)
3301 \bbbl@ifunset{\bbbl@frspc@#1}{}%
3302 {\edef\bbbl@tempa{\bbbl@cl{frspc}}%
3303 \edef\bbbl@tempa{\expandafter\@car\bbbl@tempa\@nil}%
3304 \if u\bbbl@tempa % do nothing
3305 \else\if n\bbbl@tempa % non french
3306 \expandafter\bbbl@add\csname extras#1\endcsname{%
3307 \let\bbbl@elt\bbbl@fs@elt@i
3308 \bbbl@fs@chars}%
3309 \else\if y\bbbl@tempa % french
3310 \expandafter\bbbl@add\csname extras#1\endcsname{%

```

```

3311         \let\bbl@elt\bbl@fs@elt@ii
3312         \bbl@fs@chars}%
3313     \fi\fi\fi}%
3314 %
3315 \ifx\bbl@KVP@main\@nil\else
3316     \expandafter\main@language\expandafter{#1}%
3317 \fi}
3318 % A couple of macros used above, to avoid hashes #####...
3319 \def\bbl@fs@elt@i#1#2#3{%
3320     \ifnum\sfcode`#1=#2\relax
3321         \babel@savevariable{\sfcode`#1}%
3322         \sfcode`#1=#3\relax
3323     \fi}%
3324 \def\bbl@fs@elt@ii#1#2#3{%
3325     \ifnum\sfcode`#1=#3\relax
3326         \babel@savevariable{\sfcode`#1}%
3327         \sfcode`#1=#2\relax
3328     \fi}%
3329 %
3330 \def\bbl@provide@renew#1{%
3331     \ifx\bbl@KVP@captions\@nil\else
3332         \StartBabelCommands*{#1}{captions}%
3333         \bbl@read@ini{\bbl@KVP@captions}0%   Here all letters cat = 11
3334         \bbl@after@ini
3335         \bbl@savestrings
3336         \EndBabelCommands
3337     \fi
3338     \ifx\bbl@KVP@import\@nil\else
3339         \StartBabelCommands*{#1}{date}%
3340         \bbl@savetoday
3341         \bbl@savestate
3342         \EndBabelCommands
3343     \fi
3344     % == hyphenrules ==
3345     \bbl@provide@hyphens{#1}}
3346 % Load the basic parameters (ids, typography, counters, and a few
3347 % more), while captions and dates are left out. But it may happen some
3348 % data has been loaded before automatically, so we first discard the
3349 % saved values.
3350 \def\bbl@linebreak@export{%
3351     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3352     \bbl@exportkey{hyphr}{typography.hyphenrules}{h}%
3353     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3354     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3355     \bbl@exportkey{prehc}{typography.prehyphenchar}{h}%
3356     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{h}%
3357     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{h}%
3358     \bbl@exportkey{intsp}{typography.intraspace}{h}%
3359     \bbl@exportkey{chrng}{characters.ranges}{h}}
3360 \def\bbl@load@basic#1{%
3361     \bbl@ifunset{bbl@inidata@\language}\@nil{
3362         \getlocaleproperty\bbl@tempa{\language}{identification/load.level}%
3363         \ifcase\bbl@tempa\else
3364             \bbl@csarg\let{lname@\language}\relax
3365         \fi}%
3366     \bbl@ifunset{bbl@lname@#1}%
3367     {\def\BabelBeforeIni##1##2{%
3368         \begingroup
3369         \let\bbl@ini@captions@aux\@gobbletwo

```



```

3370 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3371 \bbl@read@ini{##1}0%
3372 \bbl@linebreak@export
3373 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3374 \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3375 \ifx\bbl@initoload\relax\endinput\fi
3376 \endgroup}%
3377 \begingroup % boxed, to avoid extra spaces:
3378 \ifx\bbl@initoload\relax
3379 \bbl@input@texini{#1}%
3380 \else
3381 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3382 \fi
3383 \endgroup}%
3384 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3385 \def\bbl@provide@hyphens#1{%
3386 \let\bbl@tempa\relax
3387 \ifx\bbl@KVP@hyphenrules\@nil\else
3388 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3389 \bbl@foreach\bbl@KVP@hyphenrules{%
3390 \ifx\bbl@tempa\relax % if not yet found
3391 \bbl@ifsamestring{##1}{+}%
3392 {\bbl@exp{\addlanguage<l@##1>}}}%
3393 }%
3394 \bbl@ifunset{l@##1}%
3395 }%
3396 {\bbl@exp{\let\bbl@tempa<l@##1>}}}%
3397 \fi}%
3398 \fi
3399 \ifx\bbl@tempa\relax % if no opt or no language in opt found
3400 \ifx\bbl@KVP@import\@nil
3401 \ifx\bbl@initoload\relax\else
3402 \bbl@exp{% and hyphenrules is not empty
3403 \bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3404 }%
3405 {\let\bbl@tempa<l@bbl@cl{hyphr}>}}}%
3406 \fi
3407 \else % if importing
3408 \bbl@exp{% and hyphenrules is not empty
3409 \bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3410 }%
3411 {\let\bbl@tempa<l@bbl@cl{hyphr}>}}}%
3412 \fi
3413 \fi
3414 \bbl@ifunset{\bbl@tempa}% ie, relax or undefined
3415 {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
3416 {\bbl@exp{\adddialect<l@#1>\language}}}%
3417 }% so, l@<lang> is ok - nothing to do
3418 {\bbl@exp{\adddialect<l@#1>\bbl@tempa}}}% found in opt list or ini
3419

```

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.

```

3420 \ifx\bbl@readstream\@undefined
3421 \csname newread\endcsname\bbl@readstream
3422 \fi
3423 \def\bbl@input@texini#1{%

```

```

3424 \bbl@bsphack
3425 \bbl@exp{%
3426 \catcode`\\%=14 \catcode`\\|=0
3427 \catcode`\\={1 \catcode`\\}=2
3428 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
3429 \catcode`\\%=the\catcode`\%\relax
3430 \catcode`\\|=the\catcode`\|\relax
3431 \catcode`\\={the\catcode`\{\relax
3432 \catcode`\\}=the\catcode`\}\relax}%
3433 \bbl@esphack}
3434 \def\bbl@inipreread#1=#2\@@{%
3435 \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3436 \bbl@trim\toks@{#2}%
3437 % Move trims here ??
3438 \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
3439 {\bbl@exp{%
3440 \\\g@addto@macro\\bbl@inidata{%
3441 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3442 \expandafter\bbl@inireader\bbl@tempa=#2\@@}%
3443 }%
3444 \def\bbl@fetch@ini#1#2{%
3445 \bbl@exp{\def\\bbl@inidata{%
3446 \\\bbl@elt{identification}{tag.ini}{#1}%
3447 \\\bbl@elt{identification}{load.level}{#2}}}%
3448 \openin\bbl@readstream=babel-#1.ini
3449 \ifeof\bbl@readstream
3450 \bbl@error
3451 {There is no ini file for the requested language\\%
3452 (#1). Perhaps you misspelled it or your installation\\%
3453 is not complete.}%
3454 {Fix the name or reinstall babel.}%
3455 \else
3456 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3457 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3458 \bbl@info{Importing
3459 \ifcase#2 \or font and identification \or basic \fi
3460 data for \language\name\\%
3461 from babel-#1.ini. Reported}%
3462 \loop
3463 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3464 \endlinechar\m@ne
3465 \read\bbl@readstream to \bbl@line
3466 \endlinechar`^^M
3467 \ifx\bbl@line\empty\else
3468 \expandafter\bbl@inline\bbl@line\bbl@inline
3469 \fi
3470 \repeat
3471 \fi}
3472 \def\bbl@read@ini#1#2{%
3473 \bbl@csarg\xdef{lini@\language\name}{#1}%
3474 \let\bbl@section\@empty
3475 \let\bbl@savestrings\@empty
3476 \let\bbl@savetoday\@empty
3477 \let\bbl@savestate\@empty
3478 \let\bbl@inireader\bbl@iniskip
3479 \bbl@fetch@ini{#1}{#2}%
3480 \bbl@foreach\bbl@renewlist{%
3481 \bbl@ifunset{bbl@renew@##1}{\bbl@inisec[##1]\@}}}%
3482 \global\let\bbl@renewlist\@empty

```

```

3483 % Ends last section. See \bbl@inisec
3484 \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
3485 \bbl@cs{renew@\bbl@section}%
3486 \global\bbl@csarg\let{renew@\bbl@section}\relax
3487 \bbl@cs{secpost@\bbl@section}%
3488 \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3489 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
3490 \bbl@to\global\bbl@ini@loaded}
3491 \def\bbl@iniline#1\bbl@iniline{%
3492 \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@@}% ]

```

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

```

3493 \def\bbl@iniskip#1\@@{%          if starts with ;
3494 \def\bbl@inisec[#1]#2\@@{%      if starts with opening bracket
3495 \def\bbl@elt##1##2{%
3496   \expandafter\toks@\expandafter{%
3497     \expandafter{\bbl@section}{##1}{##2}}%
3498   \bbl@exp{%
3499     \g@addto@macro\bbl@inidata{\bbl@elt\the\toks@}}%
3500   \bbl@inireader##1=##2\@@}%
3501 \bbl@cs{renew@\bbl@section}%
3502 \global\bbl@csarg\let{renew@\bbl@section}\relax
3503 \bbl@cs{secpost@\bbl@section}%
3504 % The previous code belongs to the previous section.
3505 % -----
3506 % Now start the current one.
3507 \in@{=date.}{#1}%
3508 \ifin@
3509   \lowercase{\def\bbl@tempa{= #1=}}%
3510   \bbl@replace\bbl@tempa{=date.gregorian.}{}%
3511   \bbl@replace\bbl@tempa{=date.}{}%
3512   \in@{.licr=}{#1=}%
3513   \ifin@
3514     \ifcase\bbl@engine
3515       \bbl@replace\bbl@tempa{.licr=}{}%
3516     \else
3517       \let\bbl@tempa\relax
3518     \fi
3519   \fi
3520   \ifx\bbl@tempa\relax\else
3521     \bbl@replace\bbl@tempa{=}{}%
3522     \bbl@exp{%
3523       \def<\bbl@inikv@#1>####1=####2\@@{%
3524         \bbl@inidate####1...\relax{####2}{\bbl@tempa}}%
3525       \fi
3526     \fi
3527   \def\bbl@section{#1}%
3528   \def\bbl@elt##1##2{%
3529     \@namedef{\bbl@KVP@#1/#1}{}}%
3530   \bbl@cs{renew@#1}%
3531   \bbl@cs{secpre@#1}% pre-section `hook'
3532   \bbl@ifunset{\bbl@inikv@#1}%
3533     {\let\bbl@inireader\bbl@iniskip}%
3534     {\bbl@exp{\let\bbl@inireader<\bbl@inikv@#1>}}
3535 \let\bbl@renewlist\@empty
3536 \def\bbl@renewinikey#1/#2\@@#3{%

```

```

3537 \bbl@ifunset{\bbl@renew@#1}%
3538   {\bbl@add@list\bbl@renewlist{#1}}%
3539   }%
3540 \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>.

```

3541 \def\bbl@inikv#1=#2\@{%      key=value
3542   \bbl@trim@def\bbl@tempa{#1}%
3543   \bbl@trim\toks@{#2}%
3544   \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.

```

3545 \def\bbl@exportkey#1#2#3{%
3546   \bbl@ifunset{\bbl@kv@#2}%
3547   {\bbl@csarg\gdef{#1@\language}\{#3}}%
3548   {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
3549     \bbl@csarg\gdef{#1@\language}\{#3}}%
3550   \else
3551     \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3552     \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3553 \def\bbl@iniwarning#1{%
3554   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3555   {\bbl@warning%
3556     From babel-\bbl@cs{lini@\language}.ini:\%
3557     \bbl@cs{@kv@identification.warning#1}\%
3558     Reported }}%
3559 %
3560 \let\bbl@inikv@identification\bbl@inikv
3561 \def\bbl@secpost@identification{%
3562   \bbl@iniwarning}%
3563   \ifcase\bbl@engine
3564     \bbl@iniwarning{.pdflatex}%
3565   \or
3566     \bbl@iniwarning{.lualatex}%
3567   \or
3568     \bbl@iniwarning{.xelatex}%
3569   \fi%
3570   \bbl@exportkey{elname}{identification.name.english}{}%
3571   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3572     {\csname\bbl@elname@\language\endcsname}}%
3573   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
3574   \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
3575   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3576   \bbl@exportkey{esname}{identification.script.name}{}%
3577   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3578     {\csname\bbl@esname@\language\endcsname}}%
3579   \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3580   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3581   \ifbbl@bcptoname
3582     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\language}%
3583   \fi}

```

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

```

3584 \let\bbl@inikv@typography\bbl@inikv
3585 \let\bbl@inikv@characters\bbl@inikv
3586 \let\bbl@inikv@numbers\bbl@inikv

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localenumeral, and another one preserving the trailing .1 for the ‘units’.

```

3587 \def\bbl@inikv@counters#1=#2\@@{%
3588   \bbl@ifsamestring{#1}{digits}%
3589   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3590     decimal digits}%
3591     {Use another name.}}%
3592   }%
3593 \def\bbl@tempc{#1}%
3594 \bbl@trim@def{\bbl@tempb*}{#2}%
3595 \in@{.1$}{#1$}%
3596 \ifin@
3597   \bbl@replace\bbl@tempc{.1}{}%
3598   \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}\bbl@tempc @\language}%
3599   \noexpand\bbl@alphanumeric{\bbl@tempc}%
3600 \fi
3601 \in@{.F.}{#1}%
3602 \ifin@else\in@{.S.}{#1}\fi
3603 \ifin@
3604   \bbl@csarg\protected@xdef{cntr@#1@\language}\bbl@tempb*}%
3605 \else
3606   \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3607   \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3608   \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3609 \fi}
3610 \def\bbl@after@ini{%
3611   \bbl@linebreak@export
3612   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3613   \bbl@exportkey{rqtex}{identification.require.babel}{}%
3614   \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3615   \bbl@toglobal\bbl@savetoday
3616   \bbl@toglobal\bbl@savestate}

```

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

```

3617 \ifcase\bbl@engine
3618   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3619     \bbl@ini@captions@aux{#1}{#2}}
3620 \else
3621   \def\bbl@inikv@captions#1=#2\@@{%
3622     \bbl@ini@captions@aux{#1}{#2}}
3623 \fi

```

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

```

3624 \def\bbl@ini@captions@aux#1#2{%
3625   \bbl@trim@def\bbl@tempa{#1}%
3626   \bbl@xin@{.template}{\bbl@tempa}%
3627   \ifin@
3628     \bbl@replace\bbl@tempa{.template}{}%
3629     \def\bbl@toreplace{#2}%
3630     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3631     \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3632     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%

```

```

3633 \bbl@replace\bbl@toreplace{}}{name\endcsname{}}}%
3634 \bbl@replace\bbl@toreplace{}}{\endcsname{}}}%
3635 \bbl@xin@{,\bbl@tempa,}{,chapter,}%
3636 \ifin@
3637 \bbl@patchchapter
3638 \global\bbl@csarg\let{chapfmt@\language}\bbl@toreplace
3639 \fi
3640 \bbl@xin@{,\bbl@tempa,}{,appendix,}%
3641 \ifin@
3642 \bbl@patchchapter
3643 \global\bbl@csarg\let{appxfmt@\language}\bbl@toreplace
3644 \fi
3645 \bbl@xin@{,\bbl@tempa,}{,part,}%
3646 \ifin@
3647 \bbl@patchpart
3648 \global\bbl@csarg\let{partfmt@\language}\bbl@toreplace
3649 \fi
3650 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3651 \ifin@
3652 \toks@\expandafter{\bbl@toreplace}%
3653 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3654 \fi
3655 \else
3656 \bbl@ifblank{#2}%
3657 {\bbl@exp%
3658 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3659 {\bbl@trim\toks@{#2}}%
3660 \bbl@exp%
3661 \bbl@add\bbl@savestrings%
3662 \SetString\<\bbl@tempa name>{\the\toks@}}%
3663 \toks@\expandafter{\bbl@captionslist}%
3664 \bbl@exp{\in{\<\bbl@tempa name>}{\the\toks@}}%
3665 \ifin@else
3666 \bbl@exp%
3667 \bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3668 \bbl@to\global\<\bbl@extracaps@\language>}%
3669 \fi
3670 \fi}

```

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

```

3671 \def\bbl@list@the{%
3672 part,chapter,section,subsection,subsubsection,paragraph,%
3673 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3674 table,page,footnote,mpfootnote,mpfn}
3675 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3676 \bbl@ifunset{bbl@map@#1@\language}%
3677 {\nameuse{#1}}%
3678 {\nameuse{bbl@map@#1@\language}}}
3679 \def\bbl@inikv@labels#1=#2\@@{%
3680 \in@{.map}{#1}%
3681 \ifin@
3682 \ifx\bbl@KVP@labels\@nil\else
3683 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3684 \ifin@
3685 \def\bbl@tempc{#1}%
3686 \bbl@replace\bbl@tempc{.map}{}%
3687 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3688 \bbl@exp%

```

```

3689 \gdef\<bbl@map@bbl@tempc @\language>%
3690 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3691 \bbl@foreach\bbl@list@the{%
3692 \bbl@ifunset{the##1}{}%
3693 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3694 \bbl@exp{%
3695 \\bbl@sreplace\<the##1>%
3696 {\<bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}%
3697 \\bbl@sreplace\<the##1>%
3698 {\<\empty @\bbl@tempc>\<c##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3699 \expandafter\ifx\cname the##1\endcsname\bbl@tempd\else
3700 \toks@ \expandafter\expandafter\expandafter{%
3701 \cname the##1\endcsname}%
3702 \expandafter\xdef\cname the##1\endcsname{{\the\toks@}}%
3703 \fi}}%
3704 \fi
3705 \fi
3706 %
3707 \else
3708 %
3709 % The following code is still under study. You can test it and make
3710 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3711 % language dependent.
3712 \in@{enumerate.}{#1}%
3713 \ifin@
3714 \def\bbl@tempa{#1}%
3715 \bbl@replace\bbl@tempa{enumerate.}{}%
3716 \def\bbl@toreplace{#2}%
3717 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3718 \bbl@replace\bbl@toreplace{[ ]}{\cname the}%
3719 \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}%
3720 \toks@ \expandafter{\bbl@toreplace}%
3721 \bbl@exp{%
3722 \\bbl@add\<extras\language>{%
3723 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3724 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3725 \\bbl@toglobal\<extras\language>}%
3726 \fi
3727 \fi}

```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```

3728 \def\bbl@chapttype{chap}
3729 \ifx\@makechapterhead\@undefined
3730 \let\bbl@patchchapter\relax
3731 \else\ifx\thechapter\@undefined
3732 \let\bbl@patchchapter\relax
3733 \else\ifx\ps@headings\@undefined
3734 \let\bbl@patchchapter\relax
3735 \else
3736 \def\bbl@patchchapter{%
3737 \global\let\bbl@patchchapter\relax
3738 \bbl@add\appendix{\def\bbl@chapttype{appx}}% Not harmful, I hope
3739 \bbl@toglobal\appendix
3740 \bbl@sreplace\ps@headings
3741 {\@chapapp\ thechapter}%
3742 {\bbl@chapterformat}%

```

```

3743 \bbl@toglobal\ps@headings
3744 \bbl@sreplace\chaptermark
3745 {\@chapapp\ thechapter}%
3746 {\bbl@chapterformat}%
3747 \bbl@toglobal\chaptermark
3748 \bbl@sreplace\makechapterhead
3749 {\@chapapp\space\thechapter}%
3750 {\bbl@chapterformat}%
3751 \bbl@toglobal\@makechapterhead
3752 \gdef\bbl@chapterformat{%
3753 \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3754 {\@chapapp\space\thechapter}
3755 {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}}
3756 \fi\fi\fi
3757 \ifx\@part\undefined
3758 \let\bbl@patchpart\relax
3759 \else
3760 \def\bbl@patchpart{%
3761 \global\let\bbl@patchpart\relax
3762 \bbl@sreplace\@part
3763 {\partname\nobreakspace\thepart}%
3764 {\bbl@partformat}%
3765 \bbl@toglobal\@part
3766 \gdef\bbl@partformat{%
3767 \bbl@ifunset{\bbl@partfmt@\language}%
3768 {\partname\nobreakspace\thepart}
3769 {\@nameuse{\bbl@partfmt@\language}}}}
3770 \fi

```

Date. TODO. Document

```

3771 % Arguments are _not_ protected.
3772 \let\bbl@calendar\@empty
3773 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3774 \def\bbl@localedate#1#2#3#4{%
3775 \begin{group}
3776 \ifx\@empty#1\@empty\else
3777 \let\bbl@ld@calendar\@empty
3778 \let\bbl@ld@variant\@empty
3779 \edef\bbl@tempa{\zap@space#1 \@empty}%
3780 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3781 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3782 \edef\bbl@calendar{%
3783 \bbl@ld@calendar
3784 \ifx\bbl@ld@variant\@empty\else
3785 .\bbl@ld@variant
3786 \fi}%
3787 \bbl@replace\bbl@calendar{\gregorian}{}}%
3788 \fi
3789 \bbl@cased
3790 {\@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3791 \end{group}
3792 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3793 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3794 \bbl@trim@def\bbl@tempa{#1.#2}%
3795 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3796 {\bbl@trim@def\bbl@tempa{#3}%
3797 \bbl@trim\toks@{#5}%
3798 \@temptokena\expandafter{\bbl@savestate}%
3799 \bbl@exp{% Reverse order - in ini last wins

```



```

3800 \def\\bbl@savestate{%
3801   \\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}%
3802   \the\temptokena}}}%
3803 {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3804   {\lowercase{\def\bbl@tempb{#6}}}%
3805   \bbl@trim@def\bbl@toreplace{#5}%
3806   \bbl@TG@date
3807   \bbl@ifunset{\bbl@date@\language @}%
3808   {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3809   % TODO. Move to a better place.
3810   \bbl@exp{%
3811     \gdef\<\language date>\protect\<\language date >}%
3812     \gdef\<\language date >####1####2####3{%
3813       \\bbl@usedategrouptue
3814       \<bbl@ensure@\language>{%
3815         \\localedate{####1}{####2}{####3}}}%
3816       \\bbl@add\\bbl@savetoday{%
3817         \\SetString\\today{%
3818           \<\language date>%
3819           {\the\year}{\the\month}{\the\day}}}%
3820       }%
3821     \ifx\bbl@tempb\@empty\else
3822       \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3823     \fi}%
3824   }}%

```

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.

```

3825 \let\bbl@calendar\@empty
3826 \newcommand\BabelDateSpace{\nobreakspace}
3827 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3828 \newcommand\BabelDated[1]{\number#1}
3829 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3830 \newcommand\BabelDateM[1]{\number#1}
3831 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3832 \newcommand\BabelDateMMM[1]{%
3833   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3834 \newcommand\BabelDatey[1]{\number#1}%
3835 \newcommand\BabelDateyy[1]{%
3836   \ifnum#1<10 0\number#1 %
3837   \else\ifnum#1<100 \number#1 %
3838   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3839   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3840   \else
3841     \bbl@error
3842     {Currently two-digit years are restricted to the\
3843     range 0-9999.}%
3844     {There is little you can do. Sorry.}%
3845   \fi\fi\fi\fi}}
3846 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3847 \def\bbl@replace@finish@iii#1{%
3848   \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
3849 \def\bbl@TG@date{%
3850   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3851   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3852   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3853   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3854   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%

```

```

3855 \bbl@replace\bbl@toreplace{[MM]}\BabelDateMM{####2}%
3856 \bbl@replace\bbl@toreplace{[MMMM]}\BabelDateMMMM{####2}%
3857 \bbl@replace\bbl@toreplace{[y]}\BabelDatey{####1}%
3858 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{####1}%
3859 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}%
3860 \bbl@replace\bbl@toreplace{[y]}\bbl@datecncr[####1|}%
3861 \bbl@replace\bbl@toreplace{[m]}\bbl@datecncr[####2|}%
3862 \bbl@replace\bbl@toreplace{[d]}\bbl@datecncr[####3|}%
3863 % Note after \bbl@replace \toks@ contains the resulting string.
3864 % TODO - Using this implicit behavior doesn't seem a good idea.
3865 \bbl@replace@finish@iii\bbl@toreplace}
3866 \def\bbl@datecncr{\expandafter\bbl@xdatecncr\expandafter}
3867 \def\bbl@xdatecncr[#1|#2]{\localenumeral{#2}{#1}}

```

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

```

3868 \def\bbl@provide@lsys#1{%
3869   \bbl@ifunset{bbl@lname@#1}%
3870     {\bbl@ini@basic{#1}}%
3871     {}%
3872   \bbl@csarg\let{lsys@#1}\@empty
3873   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3874   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3875   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3876   \bbl@ifunset{bbl@lname@#1}{}%
3877     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3878   \ifcase\bbl@engine\or\or
3879     \bbl@ifunset{bbl@prehc@#1}{}%
3880     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3881      {}%
3882      {\ifx\bbl@xenohyph\@undefined
3883       \let\bbl@xenohyph\bbl@xenohyph@d
3884       \ifx\AtBeginDocument\@notprerr
3885         \expandafter\@secondoftwo % to execute right now
3886         \fi
3887       \AtBeginDocument{%
3888         \expandafter\bbl@add
3889         \csname selectfont \endcsname{\bbl@xenohyph}%
3890         \expandafter\selectlanguage\expandafter{\languagename}%
3891         \expandafter\bbl@tglobal\csname selectfont \endcsname}%
3892       \fi}}%
3893   \fi
3894   \bbl@csarg\bbl@tglobal{lsys@#1}}
3895 \def\bbl@xenohyph@d{%
3896   \bbl@ifset{bbl@prehc@languagename}%
3897     {\ifnum\hyphenchar\font=\defaultshyphenchar
3898      \iffontchar\font\bbl@c1{prehc}\relax
3899      \hyphenchar\font\bbl@c1{prehc}\relax
3900      \else\iffontchar\font"200B
3901       \hyphenchar\font"200B
3902      \else
3903        \bbl@warning
3904        {Neither 0 nor ZERO WIDTH SPACE are available\\%
3905         in the current font, and therefore the hyphen\\%
3906         will be printed. Try changing the fontspec's\\%
3907         'HyphenChar' to another value, but be aware\\%
3908         this setting is not safe (see the manual)}%
3909        \hyphenchar\font\defaultshyphenchar
3910      \fi\fi

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

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 T_EX. Non-digits characters are kept. The first macro is the generic “localized” command.

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

154

```

3957 \<ifcase>###1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3958 \else
3959 \toks@\expandafter{\the\toks@\or #1}%
3960 \expandafter\bb1@buildifcase
3961 \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).

```

3962 \newcommand\localenumeral[2]{\bb1@cs{cntr@#1@\language}\{#2}}
3963 \def\bb1@localecntr#1#2{\localenumeral{#2}{#1}}
3964 \newcommand\localecounter[2]{%
3965 \expandafter\bb1@localecntr
3966 \expandafter{\number\csname c@#2\endcsname}\{#1}}
3967 \def\bb1@alphnumeral#1#2{%
3968 \expandafter\bb1@alphnumeral@i\number#2 76543210\@{#1}}
3969 \def\bb1@alphnumeral@i#1#2#3#4#5#6#7#8\@#9{%
3970 \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
3971 \bb1@alphnumeral@ii{#9}000000#1\or
3972 \bb1@alphnumeral@ii{#9}00000#1#2\or
3973 \bb1@alphnumeral@ii{#9}0000#1#2#3\or
3974 \bb1@alphnumeral@ii{#9}000#1#2#3#4\else
3975 \bb1@alphnum@invalid{>9999}%
3976 \fi}
3977 \def\bb1@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3978 \bb1@ifunset{bb1@cntr@#1.F.\number#5#6#7#8@\language}%
3979 {\bb1@cs{cntr@#1.4@\language}\{#5}
3980 \bb1@cs{cntr@#1.3@\language}\{#6}
3981 \bb1@cs{cntr@#1.2@\language}\{#7}
3982 \bb1@cs{cntr@#1.1@\language}\{#8}
3983 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3984 \bb1@ifunset{bb1@cntr@#1.S.321@\language}\{}}%
3985 {\bb1@cs{cntr@#1.S.321@\language}\{}}%
3986 \fi}%
3987 {\bb1@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
3988 \def\bb1@alphnum@invalid#1{%
3989 \bb1@error{Alphabetic numeral too large (#1)}%
3990 {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.

```

3991 \newcommand\localeinfo[1]{%
3992 \bb1@ifunset{bb1@csname bb1@info@#1\endcsname @\language}%
3993 {\bb1@error{I've found no info for the current locale.\%
3994 The corresponding ini file has not been loaded\%
3995 Perhaps it doesn't exist}%
3996 {See the manual for details.}}%
3997 {\bb1@cs{\csname bb1@info@#1\endcsname @\language}}%
3998 % \@namedef{bb1@info@name.locale}\{lname}
3999 \@namedef{bb1@info@tag.ini}\{lini}
4000 \@namedef{bb1@info@name.english}\{elname}
4001 \@namedef{bb1@info@name.opentype}\{lname}
4002 \@namedef{bb1@info@tag.bcp47}\{tbc}
4003 \@namedef{bb1@info@language.tag.bcp47}\{lbc}
4004 \@namedef{bb1@info@tag.opentype}\{lotf}

```

```

4005 \@namedef{bbl@info@script.name}{esname}
4006 \@namedef{bbl@info@script.name.opentype}{sname}
4007 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4008 \@namedef{bbl@info@script.tag.opentype}{sotf}
4009 \let\bbl@ensureinfo\@gobble
4010 \newcommand\BabelEnsureInfo{%
4011   \ifx\InputIfFileExists\undefined\else
4012     \def\bbl@ensureinfo##1{%
4013       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}}%
4014   \fi
4015   \bbl@foreach\bbl@loaded{%
4016     \def\language{##1}%
4017     \bbl@ensureinfo{##1}}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

4018 \newcommand\getlocaleproperty{%
4019   \@ifstar\bbl@getproperty@s\bbl@getproperty@x%
4020   \def\bbl@getproperty@s#1#2#3{%
4021     \let#1\relax
4022     \def\bbl@elt##1##2##3{%
4023       \bbl@ifsamestring{##1/##2}{#3}%
4024       {\providecommand#1{##3}%
4025       \def\bbl@elt####1####2####3{}}}%
4026     {}}%
4027     \bbl@cs{inidata@#2}}%
4028   \def\bbl@getproperty@x#1#2#3{%
4029     \bbl@getproperty@s{#1}{#2}{#3}%
4030     \ifx#1\relax
4031       \bbl@error
4032         {Unknown key for locale '#2':\%
4033         #3\%
4034         \string#1 will be set to \relax}%
4035       {Perhaps you misspelled it.}%
4036     \fi}
4037   \let\bbl@ini@loaded\@empty
4038   \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel behavior

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

```

4039 \newcommand\babeladjust[1]{% TODO. Error handling.
4040   \bbl@forkv{#1}{%
4041     \bbl@ifunset{bbl@ADJ@##1@##2}%
4042     {\bbl@cs{ADJ@##1}{##2}}%
4043     {\bbl@cs{ADJ@##1@##2}}}
4044 %
4045 \def\bbl@adjust@lua#1#2{%
4046   \ifvmode
4047     \ifnum\currentgrouplevel=\z@
4048       \directlua{ Babel.#2 }%
4049       \expandafter\expandafter\expandafter\@gobble
4050     \fi
4051   \fi
4052   {\bbl@error % The error is gobbled if everything went ok.
4053     {Currently, #1 related features can be adjusted only\%

```

```

4054         in the main vertical list.}%
4055         {Maybe things change in the future, but this is what it is.}}
4056 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4057   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4058 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4059   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4060 \@namedef{bbl@ADJ@bidi.text@on}{%
4061   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4062 \@namedef{bbl@ADJ@bidi.text@off}{%
4063   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4064 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4065   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4066 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4067   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4068 %
4069 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4070   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4071 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4072   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4073 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4074   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4075 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4076   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4077 %
4078 \def\bbl@adjust@layout#1{%
4079   \ifvmode
4080     #1%
4081     \expandafter\@gobble
4082   \fi
4083   {\bbl@error   % The error is gobbled if everything went ok.
4084     {Currently, layout related features can be adjusted only\\%
4085       in vertical mode.}%
4086     {Maybe things change in the future, but this is what it is.}}
4087 \@namedef{bbl@ADJ@layout.tabular@on}{%
4088   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4089 \@namedef{bbl@ADJ@layout.tabular@off}{%
4090   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4091 \@namedef{bbl@ADJ@layout.lists@on}{%
4092   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4093 \@namedef{bbl@ADJ@layout.lists@off}{%
4094   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4095 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4096   \bbl@activateposthyphen}
4097 %
4098 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4099   \bbl@bcpallowedtrue}
4100 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4101   \bbl@bcpallowedfalse}
4102 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4103   \def\bbl@bcp@prefix{#1}}
4104 \def\bbl@bcp@prefix{bcp47-}
4105 \@namedef{bbl@ADJ@autoload.options}#1{%
4106   \def\bbl@autoload@options{#1}}
4107 \let\bbl@autoload@bcptoptions\@empty
4108 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4109   \def\bbl@autoload@bcptoptions{#1}}
4110 \newif\ifbbl@bcptoname
4111 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4112   \bbl@bcptonametrue

```

```

4113 \BabelEnsureInfo}
4114 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4115 \bbl@bcptonamefalse}
4116% TODO: use babel name, override
4117%
4118% As the final task, load the code for lua.
4119%
4120 \ifx\directlua\@undefined\else
4121 \ifx\bbl@luapatterns\@undefined
4122 \input luababel.def
4123 \fi
4124 \fi
4125 \</core>

A proxy file for switch.def

4126 \<*kernel>
4127 \let\bbl@onlyswitch\@empty
4128 \input babel.def
4129 \let\bbl@onlyswitch\@undefined
4130 \</kernel>
4131 \<*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by \LaTeX because it should instruct \TeX to read hyphenation patterns. To this end the `docstrip` option `patterns` can be used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

To make sure that \LaTeX 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```

4132 \<<Make sure ProvidesFile is defined>>
4133 \ProvidesFile{hyphen.cfg}[\<<date>>] [\<<version>>] Babel hyphens]
4134 \xdef\bbl@format{\jobname}
4135 \def\bbl@version{\<<version>>}
4136 \def\bbl@date{\<<date>>}
4137 \ifx\AtBeginDocument\@undefined
4138 \def\@empty{}
4139 \let\orig@dump\dump
4140 \def\dump{%
4141 \ifx\@ztryfc\@undefined
4142 \else
4143 \toks0=\expandafter{\@preamblecmds}%
4144 \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4145 \def\@begindocumenthook{}%
4146 \fi
4147 \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4148 \fi
4149 \<<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.

```
4150 \def\process@line#1#2 #3 #4 {%
4151   \ifx=#1%
4152     \process@synonym{#2}%
4153   \else
4154     \process@language{#1#2}{#3}{#4}%
4155   \fi
4156   \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.

```
4157 \toks@{}
4158 \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.

```
4159 \def\process@synonym#1{%
4160   \ifnum\last@language=\m@ne
4161     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4162   \else
4163     \expandafter\chardef\csname l@#1\endcsname\last@language
4164     \wlog{\string\l@#1=\string\language\the\last@language}%
4165     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4166       \csname\language\hyphenmins\endcsname
4167     \let\bbl@elt\relax
4168     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4169   \fi}
```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\(lang)hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not

empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```

4170 \def\process@language#1#2#3{%
4171   \expandafter\addlanguage\csname l@#1\endcsname
4172   \expandafter\language\csname l@#1\endcsname
4173   \edef\language#1{%
4174     \bbl@hook@everylanguage{#1}%
4175     % > luatex
4176     \bbl@get@enc#1::\@@@
4177   \begingroup
4178     \lefthyphenmin\m@ne
4179     \bbl@hook@loadpatterns{#2}%
4180     % > luatex
4181     \ifnum\lefthyphenmin=\m@ne
4182     \else
4183       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4184         \the\lefthyphenmin\the\righthyphenmin}%
4185     \fi
4186   \endgroup
4187   \def\bbl@tempa{#3}%
4188   \ifx\bbl@tempa\@empty\else
4189     \bbl@hook@loadexceptions{#3}%
4190     % > luatex
4191   \fi
4192   \let\bbl@elt\relax
4193   \edef\bbl@languages{%
4194     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4195   \ifnum\the\language=\z@
4196     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4197       \set@hyphenmins\tw@\thr@@\relax
4198     \else
4199       \expandafter\expandafter\expandafter\set@hyphenmins
4200       \csname #1hyphenmins\endcsname
4201     \fi
4202     \the\toks@
4203     \toks@{}%
4204   \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.

```

4205 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. loadkernel currently loads nothing, but define some basic macros instead.

```

4206 \def\bbl@hook@everylanguage#1{}
4207 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4208 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4209 \def\bbl@hook@loadkernel#1{%
4210   \def\addlanguage{\csname newlanguage\endcsname}%
4211   \def\adddialect##1##2{%
4212     \global\chardef##1##2\relax

```

```

4213 \wlog{\string##1 = a dialect from \string\language##2}%
4214 \def\iflanguage##1{%
4215 \expandafter\ifx\csname l@##1\endcsname\relax
4216 \@nolanerr{##1}%
4217 \else
4218 \ifnum\csname l@##1\endcsname=\language
4219 \expandafter\expandafter\expandafter\@firstoftwo
4220 \else
4221 \expandafter\expandafter\expandafter\@secondoftwo
4222 \fi
4223 \fi}%
4224 \def\providehyphenmins##1##2{%
4225 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4226 \@namedef{##1hyphenmins}{##2}%
4227 \fi}%
4228 \def\set@hyphenmins##1##2{%
4229 \leftthyphenmin##1\relax
4230 \rightthyphenmin##2\relax}%
4231 \def\selectlanguage{%
4232 \errhelp{Selecting a language requires a package supporting it}%
4233 \errmessage{Not loaded}}%
4234 \let\foreignlanguage\selectlanguage
4235 \let\otherlanguage\selectlanguage
4236 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4237 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4238 \def\setlocale{%
4239 \errhelp{Find an armchair, sit down and wait}%
4240 \errmessage{Not yet available}}%
4241 \let\uselocale\setlocale
4242 \let\locale\setlocale
4243 \let\selectlocale\setlocale
4244 \let\localename\setlocale
4245 \let\textlocale\setlocale
4246 \let\textlanguage\setlocale
4247 \let\languagetext\setlocale}
4248 \begingroup
4249 \def\AddBabelHook#1#2{%
4250 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4251 \def\next{\toks1}%
4252 \else
4253 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4254 \fi
4255 \next}
4256 \ifx\directlua\undefined
4257 \ifx\XeTeXinputencoding\undefined\else
4258 \input xebabel.def
4259 \fi
4260 \else
4261 \input luababel.def
4262 \fi
4263 \openin1 = babel-\bbl@format.cfg
4264 \ifeof1
4265 \else
4266 \input babel-\bbl@format.cfg\relax
4267 \fi
4268 \closein1
4269 \endgroup
4270 \bbl@hook@loadkernel{switch.def}

```

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

```
4271 \openin1 = language.dat
```

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

```
4272 \def\languagename{english}%
4273 \ifeof1
4274 \message{I couldn't find the file language.dat,\space
4275         I will try the file hyphen.tex}
4276 \input hyphen.tex\relax
4277 \chardef\l@english\z@
4278 \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`.

```
4279 \last@language\m@ne
```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
4280 \loop
4281 \endlinechar\m@ne
4282 \read1 to \bbl@line
4283 \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.

```
4284 \if T\ifeof1F\fi T\relax
4285 \ifx\bbl@line\@empty\else
4286 \edef\bbl@line{\bbl@line\space\space\space}%
4287 \expandafter\process@line\bbl@line\relax
4288 \fi
4289 \repeat
```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns, and close the configuration file.

```
4290 \begingroup
4291 \def\bbl@elt#1#2#3#4{%
4292 \global\language=#2\relax
4293 \gdef\languagename{#1}%
4294 \def\bbl@elt##1##2##3##4{}}%
4295 \bbl@languages
4296 \endgroup
4297 \fi
4298 \closein1
```

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

```
4299 \if/\the\toks@/\else
4300 \errhelp{language.dat loads no language, only synonyms}
4301 \errmessage{Orphan language synonym}
4302 \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.

```

4303 \let\bbl@line\@undefined
4304 \let\process@line\@undefined
4305 \let\process@synonym\@undefined
4306 \let\process@language\@undefined
4307 \let\bbl@get@enc\@undefined
4308 \let\bbl@hyph@enc\@undefined
4309 \let\bbl@tempa\@undefined
4310 \let\bbl@hook@loadkernel\@undefined
4311 \let\bbl@hook@everylanguage\@undefined
4312 \let\bbl@hook@loadpatterns\@undefined
4313 \let\bbl@hook@loadexceptions\@undefined
4314 \let\bbl@hook@loadpatterns\@undefined

```

Here the code for iniT_EX ends.

12 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].

```

4315 <<(*More package options)>> ≡
4316 \chardef\bbl@bidimode\z@
4317 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4318 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4319 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4320 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4321 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4322 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4323 <</More package options>>

```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. bbl@font replaces hardcoded font names inside \..family by the corresponding macro \..default.

At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading message, which is replaced by a more explanatory one.

```

4324 <<(*Font selection)>> ≡
4325 \bbl@trace{Font handling with fontspec}
4326 \ifx\ExplSyntaxOn\@undefined\else
4327   \ExplSyntaxOn
4328   \catcode`\ =10
4329   \def\bbl@loadfontspec{%
4330     \usepackage{fontspec}%
4331     \expandafter
4332     \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4333       Font '\l_fontspec_fontname_tl' is using the\\%
4334       default features for language '##1'.\\%
4335       That's usually fine, because many languages\\%
4336       require no specific features, but if the output is\\%
4337       not as expected, consider selecting another font.}
4338     \expandafter
4339     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%

```

```

4340     Font '\l_fontspec_fontname_tl' is using the\\%
4341     default features for script '##2'.\\%
4342     That's not always wrong, but if the output is\\%
4343     not as expected, consider selecting another font.}}
4344 \ExplSyntaxOff
4345 \fi
4346 \@onlypreamble\babelfont
4347 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4348 \bbl@foreach{#1}{\%
4349 \expandafter\ifx\csname date##1\endcsname\relax
4350 \IfFileExists{babel-##1.tex}%
4351 {\babelprovide{##1}}%
4352 }%
4353 \fi}%
4354 \edef\bbl@tempa{#1}%
4355 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4356 \ifx\fontspec@undefined
4357 \bbl@loadfontspec
4358 \fi
4359 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4360 \bbl@bblfont}
4361 \newcommand\bbl@bblfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
4362 \bbl@ifunset{\bbl@tempb family}%
4363 {\bbl@providefam{\bbl@tempb}}%
4364 {\bbl@exp{%
4365 \\\bbl@sreplace\<\bbl@tempb family >%
4366 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4367 % For the default font, just in case:
4368 \bbl@ifunset{\bbl@lsys\language\name}{\bbl@provide@lsys{\language\name}}}%
4369 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4370 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4371 \bbl@exp{%
4372 \let\bbl@\bbl@tempb dflt@\language\name>\<\bbl@\bbl@tempb dflt@>%
4373 \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\language\name>%
4374 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4375 {\bbl@foreach\bbl@tempa{\% ie bbl@rmdflt@lang / *scrt
4376 \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:

```

4377 \def\bbl@providefam#1{\%
4378 \bbl@exp{%
4379 \\\newcommand\<#1default>{\% Just define it
4380 \\\bbl@add@list\\\bbl@font@fams{#1}%
4381 \\\DeclareRobustCommand\<#1family>{\%
4382 \\\not@math@alphabet\<#1family>\relax
4383 \\\fontfamily\<#1default>\selectfont}%
4384 \\\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.

```

4385 \def\bbl@nostdfont#1{\%
4386 \bbl@ifunset{\bbl@WFF@\f@family}%
4387 {\bbl@csarg\gdef{WFF@\f@family}}{\% Flag, to avoid dupl warns
4388 \bbl@infowarn{The current font is not a babel standard family:\\%
4389 #1%
4390 \fontname\font\\%
4391 There is nothing intrinsically wrong with this warning, and\\%
4392 you can ignore it altogether if you do not need these\\%
4393 families. But if they are used in the document, you should be\\%

```

```

4394     aware 'babel' will no set Script and Language for them, so\\%
4395     you may consider defining a new family with \string\babelfont.\\%
4396     See the manual for further details about \string\babelfont.\\%
4397     Reported}}
4398     {}}%
4399 \gdef\bbl@switchfont{%
4400   \bbl@ifunset\bbl@lsys@\language\name\{\bbl@provide@lsys\{\language\name\}\}%
4401   \bbl@exp{%   eg Arabic -> arabic
4402     \lowercase{\edef\\bbl@tempa{\bbl@cl\{sname\}}}%
4403   \bbl@foreach\bbl@font@fams{%
4404     \bbl@ifunset\bbl@##1dflt@\language\name\%      (1) language?
4405     {\bbl@ifunset\bbl@##1dflt@*\bbl@tempa\}%      (2) from script?
4406     {\bbl@ifunset\bbl@##1dflt@\}%                  2=F - (3) from generic?
4407     {}%                                              123=F - nothing!
4408     {\bbl@exp{%                                     3=T - from generic
4409       \global\let<\bbl@##1dflt@\language\name>%
4410         \<\bbl@##1dflt@>}}}%
4411     {\bbl@exp{%                                     2=T - from script
4412       \global\let<\bbl@##1dflt@\language\name>%
4413         \<\bbl@##1dflt@*\bbl@tempa>}}}%
4414     {}%                                              1=T - language, already defined
4415   \def\bbl@tempa{\bbl@nostdfont{}}}%
4416   \bbl@foreach\bbl@font@fams{%   don't gather with prev for
4417     \bbl@ifunset\bbl@##1dflt@\language\name\%
4418     {\bbl@cs{famrst@##1}%
4419     \global\bbl@csarg\let{famrst@##1}\relax}%
4420     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4421       \\bbl@add\\originalTeX{%
4422         \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4423         \<##1default>\<##1family>{##1}}}%
4424         \\bbl@font@set<\bbl@##1dflt@\language\name>% the main part!
4425         \<##1default>\<##1family>}}}%
4426   \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.

```

4427 \ifx\family\undefined\else   % if latex
4428 \ifcase\bbl@engine           % if pdftex
4429   \let\bbl@ckeckstdfonts\relax
4430 \else
4431   \def\bbl@ckeckstdfonts{%
4432     \begingroup
4433     \global\let\bbl@ckeckstdfonts\relax
4434     \let\bbl@tempa\@empty
4435     \bbl@foreach\bbl@font@fams{%
4436       \bbl@ifunset\bbl@##1dflt@\%
4437       {\nameuse{##1family}%
4438       \bbl@csarg\gdef{WFF@\family\}% Flag
4439       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \family\\}%
4440         \space\space\fontname\font\\}%
4441       \bbl@csarg\xdef{##1dflt@}{\family}%
4442       \expandafter\xdef\csname ##1default\endcsname{\family}%
4443       }%
4444     \ifx\bbl@tempa\@empty\else
4445       \bbl@infowarn{The following font families will use the default\\%
4446         settings for all or some languages:\\%
4447         \bbl@tempa
4448         There is nothing intrinsically wrong with it, but\\%
4449         'babel' will no set Script and Language, which could\\%

```

```

4450             be relevant in some languages. If your document uses\\%
4451             these families, consider redefining them with \string\babelfont.\\%
4452             Reported}%
4453             \fi
4454         \endgroup}
4455     \fi
4456 \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.

```

4457 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4458     \bbl@xin@{<>}{#1}%
4459     \ifin@
4460         \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4461     \fi
4462     \bbl@exp{%
4463         \def\\#2{#1}%          'Unprotected' macros return prev values
4464         \\\bbl@ifsamestring{#2}{\f@family}%
4465         {\\#3%
4466             \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4467             \let\\bbl@tempa\relax}%
4468         {}}}
4469 %      TODO - next should be global?, but even local does its job. I'm
4470 %      still not sure -- must investigate:
4471 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4472     \let\bbl@tempe\bbl@mapselect
4473     \let\bbl@mapselect\relax
4474     \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4475     \let#4\@empty           %      Make sure \renewfontfamily is valid
4476     \bbl@exp{%
4477         \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4478         \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4479         {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4480         \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4481         {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4482         \\renewfontfamily\\#4%
4483         [\bbl@cs{lsys@\language\name},#2]{#3}% ie \bbl@exp{..}{#3}
4484     \begingroup
4485         #4%
4486         \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4487     \endgroup
4488     \let#4\bbl@temp@fam
4489     \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4490     \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.

```

4491 \def\bbl@font@rst#1#2#3#4{%
4492     \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.

```

4493 \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 :-).

```

4494 \newcommand\babelFSstore[2][{}]{%
4495   \bbl@ifblank{#1}%
4496   {\bbl@csarg\def{sname@#2}{Latin}}%
4497   {\bbl@csarg\def{sname@#2}{#1}}%
4498   \bbl@provide@dirs{#2}%
4499   \bbl@csarg\ifnum{wdir@#2}>\z@
4500     \let\bbl@beforeforeign\leavevmode
4501     \EnableBabelHook{babel-bidi}%
4502   \fi
4503   \bbl@foreach{#2}{%
4504     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4505     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4506     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4507 \def\bbl@FSstore#1#2#3#4{%
4508   \bbl@csarg\edef{#2default#1}{#3}%
4509   \expandafter\addto\csname extras#1\endcsname{%
4510     \let#4#3%
4511     \ifx#3\f@family
4512       \edef#3{\csname bbl@#2default#1\endcsname}%
4513       \fontfamily{#3}\selectfont
4514     \else
4515       \edef#3{\csname bbl@#2default#1\endcsname}%
4516     \fi}%
4517   \expandafter\addto\csname noextras#1\endcsname{%
4518     \ifx#3\f@family
4519       \fontfamily{#4}\selectfont
4520     \fi
4521     \let#3#4}}
4522 \let\bbl@langfeatures\@empty
4523 \def\babelFSfeatures{% make sure \fontspec is redefined once
4524   \let\bbl@ori@fontspec\fontspec
4525   \renewcommand\fontspec[1][{}]{%
4526     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4527   \let\babelFSfeatures\bbl@FSfeatures
4528   \babelFSfeatures}
4529 \def\bbl@FSfeatures#1#2{%
4530   \expandafter\addto\csname extras#1\endcsname{%
4531     \babel@save\bbl@langfeatures
4532     \edef\bbl@langfeatures{#2,}}
4533 <</Font selection>>

```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

```

4534 <<{*Footnote changes}>> ≡
4535 \bbl@trace{Bidi footnotes}
4536 \ifnum\bbl@bidimode>\z@
4537   \def\bbl@footnote#1#2#3{%
4538     \@ifnextchar[%
4539       {\bbl@footnote@o{#1}{#2}{#3}}%
4540       {\bbl@footnote@x{#1}{#2}{#3}}}
4541   \long\def\bbl@footnote@x#1#2#3#4{%
4542     \bgroup

```



```

4543 \select@language@x{\bbl@main@language}%
4544 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4545 \egroup}
4546 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4547 \bgroup
4548 \select@language@x{\bbl@main@language}%
4549 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4550 \egroup}
4551 \def\bbl@footnotetext#1#2#3{%
4552 \@ifnextchar[%
4553 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4554 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4555 \long\def\bbl@footnotetext@x#1#2#3#4{%
4556 \bgroup
4557 \select@language@x{\bbl@main@language}%
4558 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4559 \egroup}
4560 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4561 \bgroup
4562 \select@language@x{\bbl@main@language}%
4563 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4564 \egroup}
4565 \def\BabelFootnote#1#2#3#4{%
4566 \ifx\bbl@fn@footnote\undefined
4567 \let\bbl@fn@footnote\footnote
4568 \fi
4569 \ifx\bbl@fn@footnotetext\undefined
4570 \let\bbl@fn@footnotetext\footnotetext
4571 \fi
4572 \bbl@ifblank{#2}%
4573 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4574 \@namedef{\bbl@stripslash#1text}%
4575 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4576 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4577 \@namedef{\bbl@stripslash#1text}%
4578 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4579 \fi
4580 <</Footnote changes>>

```

Now, the code.

```

4581 (*xetex)
4582 \def\BabelStringsDefault{unicode}
4583 \let\xebbl@stop\relax
4584 \AddBabelHook{xetex}{encodedcommands}{%
4585 \def\bbl@tempa{#1}%
4586 \ifx\bbl@tempa\empty
4587 \XeTeXinputencoding"bytes"%
4588 \else
4589 \XeTeXinputencoding"#1"%
4590 \fi
4591 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4592 \AddBabelHook{xetex}{stopcommands}{%
4593 \xebbl@stop
4594 \let\xebbl@stop\relax}
4595 \def\bbl@intraspace#1 #2 #3\@@{%
4596 \bbl@csarg\gdef{\xeisp@language}%
4597 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4598 \def\bbl@intrapenalty#1\@@{%
4599 \bbl@csarg\gdef{\xeipn@language}%

```

```

4600     {\XeTeXlinebreakpenalty #1\relax}}
4601 \def\bbl@provide@intraspace{%
4602   \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4603   \ifin@else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4604   \ifin@
4605     \bbl@ifunset{\bbl@intsp@{\languagename}}{s}%
4606     {\expandafter\ifx\cename\bbl@intsp@{\languagename}\endcsname\empty\else
4607       \ifx\bbl@KVP@intraspace\@nil
4608         \bbl@exp{%
4609           \\bbl@intraspace\bbl@cl{intsp}\\@}%
4610         \fi
4611         \ifx\bbl@KVP@intrapenalty\@nil
4612           \bbl@intrapenalty0\@@
4613         \fi
4614       \fi
4615       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4616         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4617       \fi
4618       \ifx\bbl@KVP@intrapenalty\@nil\else
4619         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4620       \fi
4621       \bbl@exp{%
4622         \\bbl@add\<extras\languagename>{%
4623           \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4624           \<bbl@xeisp@\languagename>%
4625           \<bbl@xeipn@\languagename>}%
4626         \\bbl@tglobal\<extras\languagename>%
4627         \\bbl@add\<noextras\languagename>{%
4628           \XeTeXlinebreaklocale "en"%
4629           \\bbl@tglobal\<noextras\languagename>}%
4630       \ifx\bbl@ispace\@undefined
4631         \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4632       \ifx\AtBeginDocument\@notprerr
4633         \expandafter\@secondoftwo % to execute right now
4634       \fi
4635       \AtBeginDocument{%
4636         \expandafter\bbl@add
4637         \cselectfont \endcsname{\bbl@ispace}%
4638         \expandafter\bbl@tglobal\cselectfont \endcsname}%
4639     \fi}%
4640 \fi}
4641 \ifx\DisableBabelHook\@undefined\endinput\fi
4642 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4643 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4644 \DisableBabelHook{babel-fontspec}
4645 <<Font selection>>
4646 \input txtbabel.def
4647 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titlesp, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T_EX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdfTeX

and xetex.

```
4648 (*texxet)
4649 \providecommand\bbl@provide@intraspace{}
4650 \bbl@trace{Redefinitions for bidi layout}
4651 \def\bbl@sspre@caption{%
4652   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir\bbl@main@language}}}}
4653 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4654 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4655 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4656 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4657   \def\hangfrom#1{%
4658     \setbox\@tempboxa\hbox{#1}%
4659     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4660     \noindent\box\@tempboxa}
4661 \def\raggedright{%
4662   \let\@centercr
4663   \bbl@startskip\z@skip
4664   \@rightskip\@flushglue
4665   \bbl@endskip\@rightskip
4666   \parindent\z@
4667   \parfillskip\bbl@startskip}
4668 \def\raggedleft{%
4669   \let\@centercr
4670   \bbl@startskip\@flushglue
4671   \bbl@endskip\z@skip
4672   \parindent\z@
4673   \parfillskip\bbl@endskip}
4674 \fi
4675 \IfBabelLayout{lists}
4676   {\bbl@sreplace\list
4677     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4678     \def\bbl@listleftmargin{%
4679       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4680     \ifcase\bbl@engine
4681       \def\labelenumii{\theenumii}% pdfTeX doesn't reverse ()
4682       \def\p@enumiii{\p@enumii}\theenumii}%
4683     \fi
4684     \bbl@sreplace\@verbatim
4685       {\leftskip\@totalleftmargin}%
4686       {\bbl@startskip\textwidth
4687         \advance\bbl@startskip-\linewidth}%
4688     \bbl@sreplace\@verbatim
4689       {\rightskip\z@skip}%
4690       {\bbl@endskip\z@skip}}%
4691   {}
4692 \IfBabelLayout{contents}
4693   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4694     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4695   {}
4696 \IfBabelLayout{columns}
4697   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4698     \def\bbl@outputbox#1{%
4699       \hb@xt@\textwidth{%
4700         \hskip\columnwidth
4701         \hfil
4702         {\normalcolor\vrule \@width\columnseprule}%
4703         \hfil
4704         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4705       }
4706     }
4707   }
```

```

4705      \hskip-\textwidth
4706      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4707      \hskip\columnsep
4708      \hskip\columnwidth}}}%
4709  {}
4710  <<Footnote changes>>
4711  \IfBabelLayout{footnotes}%
4712  {\BabelFootnote\footnote\language\language{}{}}%
4713  \BabelFootnote\localfootnote\language\language{}{}}%
4714  \BabelFootnote\mainfootnote{}{}}{}
4715  {}

```

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.

```

4716 \IfBabelLayout{counters}%
4717 {\let\bbbl@latinarabic=\@arabic
4718  \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}}%
4719  \let\bbbl@asciroman=\@roman
4720  \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
4721  \let\bbbl@asciiRoman=\@Roman
4722  \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}}{}
4723 </texet>

```

13.3 LuaTeX

The loader for luatex is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with `luatex` patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, `lua(e)tex` is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4724 (*luatex)
4725 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4726 \bbl@trace{Read language.dat}
4727 \ifx\bbl@readstream\undefined
4728   \csname newread\endcsname\bbl@readstream
4729 \fi
4730 \begingroup
4731   \toks@{}
4732   \count@ \z@ % 0=start, 1=0th, 2=normal
4733   \def\bbl@process@line#1#2 #3 #4 {%
4734     \ifx=#1%
4735       \bbl@process@synonym{#2}%
4736     \else
4737       \bbl@process@language{#1#2}{#3}{#4}%
4738     \fi
4739     \ignorespaces}
4740   \def\bbl@manylang{%
4741     \ifnum\bbl@last>\@ne
4742       \bbl@info{Non-standard hyphenation setup}%
4743     \fi
4744     \let\bbl@manylang\relax}
4745   \def\bbl@process@language#1#2#3{%
4746     \ifcase\count@
4747       \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4748     \or
4749       \count@\tw@
4750     \fi
4751     \ifnum\count@=\tw@
4752       \expandafter\addlanguage\csname l@#1\endcsname
4753       \language\allocationnumber
4754       \chardef\bbl@last\allocationnumber
4755       \bbl@manylang
4756       \let\bbl@elt\relax
4757       \xdef\bbl@languages{%
4758         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4759     \fi
4760     \the\toks@
4761     \toks@{}}
4762   \def\bbl@process@synonym@aux#1#2{%
4763     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4764     \let\bbl@elt\relax
4765     \xdef\bbl@languages{%
4766       \bbl@languages\bbl@elt{#1}{#2}{}}}%
4767   \def\bbl@process@synonym#1{%
4768     \ifcase\count@
4769       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4770     \or
4771       \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4772     \else
4773       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4774     \fi}
4775   \ifx\bbl@languages\undefined % Just a (sensible?) guess
4776     \chardef\l@english\z@
4777     \chardef\l@USenglish\z@

```

```

4778 \chardef\bbl@last\z@
4779 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
4780 \gdef\bbl@languages{%
4781   \bbl@elt{english}{0}{\hyphen.tex}{}%
4782   \bbl@elt{USenglish}{0}{}{}}
4783 \else
4784   \global\let\bbl@languages@format\bbl@languages
4785   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4786     \ifnum#2>\z@\else
4787       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4788     \fi}%
4789   \xdef\bbl@languages{\bbl@languages}%
4790 \fi
4791 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{} } % Define flags
4792 \bbl@languages
4793 \openin\bbl@readstream=language.dat
4794 \ifeof\bbl@readstream
4795   \bbl@warning{I couldn't find language.dat. No additional\\%
4796     patterns loaded. Reported}%
4797 \else
4798   \loop
4799     \endlinechar\m@ne
4800     \read\bbl@readstream to \bbl@line
4801     \endlinechar`\^^M
4802     \if T\ifeof\bbl@readstream F\fi T\relax
4803     \ifx\bbl@line\empty\else
4804       \edef\bbl@line{\bbl@line\space\space\space}%
4805       \expandafter\bbl@process@line\bbl@line\relax
4806     \fi
4807   \repeat
4808 \fi
4809 \endgroup
4810 \bbl@trace{Macros for reading patterns files}
4811 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4812 \ifx\babelcatcodetablenum\undefined
4813   \ifx\newcatcodetable\undefined
4814     \def\babelcatcodetablenum{5211}
4815     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4816   \else
4817     \newcatcodetable\babelcatcodetablenum
4818     \newcatcodetable\bbl@pattcodes
4819   \fi
4820 \else
4821   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4822 \fi
4823 \def\bbl@luapatterns#1#2{%
4824   \bbl@get@enc#1::\@@@
4825   \setbox\z@\hbox\bgroup
4826   \begingroup
4827     \savecatcodetable\babelcatcodetablenum\relax
4828     \initcatcodetable\bbl@pattcodes\relax
4829     \catcodetable\bbl@pattcodes\relax
4830     \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4831     \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4832     \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4833     \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4834     \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4835     \catcode`\'=12 \catcode`\'=12 \catcode`\`=12
4836     \input #1\relax

```

```

4837 \catcodetable\babelcatcodetablenum\relax
4838 \endgroup
4839 \def\bbbl@tempa{#2}%
4840 \ifx\bbbl@tempa\@empty\else
4841 \input #2\relax
4842 \fi
4843 \egroup}%
4844 \def\bbbl@patterns@lua#1{%
4845 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4846 \csname l@#1\endcsname
4847 \edef\bbbl@tempa{#1}%
4848 \else
4849 \csname l@#1:\f@encoding\endcsname
4850 \edef\bbbl@tempa{#1:\f@encoding}%
4851 \fi\relax
4852 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4853 \@ifundefined{bbbl@hyphendata@the\language}%
4854 {\def\bbbl@elt##1##2##3##4{%
4855 \ifnum##2=\csname l@bbbl@tempa\endcsname % #2=spanish, dutch:OT1...
4856 \def\bbbl@tempb{##3}%
4857 \ifx\bbbl@tempb\@empty\else % if not a synonymous
4858 \def\bbbl@tempc{##3}{##4}%
4859 \fi
4860 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
4861 \fi}%
4862 \bbbl@languages
4863 \@ifundefined{bbbl@hyphendata@the\language}%
4864 {\bbbl@info{No hyphenation patterns were set for\%
4865 language '\bbbl@tempa'. Reported}}%
4866 {\expandafter\expandafter\expandafter\bbbl@luapatterns
4867 \csname bbl@hyphendata@the\language\endcsname}}}%
4868 \endinput\fi
4869 % Here ends \ifx\AddBabelHook\@undefined
4870 % A few lines are only read by hyphen.cfg
4871 \ifx\DisableBabelHook\@undefined
4872 \AddBabelHook{luatex}{everylanguage}{%
4873 \def\process@language##1##2##3{%
4874 \def\process@line####1####2 ####3 ####4 {}}}
4875 \AddBabelHook{luatex}{loadpatterns}{%
4876 \input #1\relax
4877 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4878 {{#1}}}%
4879 \AddBabelHook{luatex}{loadexceptions}{%
4880 \input #1\relax
4881 \def\bbbl@tempb##1##2{{##1}{##2}}%
4882 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4883 {\expandafter\expandafter\expandafter\bbbl@tempb
4884 \csname bbl@hyphendata@the\language\endcsname}}
4885 \endinput\fi
4886 % Here stops reading code for hyphen.cfg
4887 % The following is read the 2nd time it's loaded
4888 \begingroup % TODO - to a lua file
4889 \catcode`\%=12
4890 \catcode`\'=12
4891 \catcode`\%=12
4892 \catcode`\:=12
4893 \directlua{
4894 Babel = Babel or {}
4895 function Babel.bytes(line)

```

```

4896     return line:gsub("(.)",
4897         function (chr) return unicode.utf8.char(string.byte(chr)) end)
4898 end
4899 function Babel.begin_process_input()
4900     if luatexbase and luatexbase.add_to_callback then
4901         luatexbase.add_to_callback('process_input_buffer',
4902             Babel.bytes,'Babel.bytes')
4903     else
4904         Babel.callback = callback.find('process_input_buffer')
4905         callback.register('process_input_buffer',Babel.bytes)
4906     end
4907 end
4908 function Babel.end_process_input ()
4909     if luatexbase and luatexbase.remove_from_callback then
4910         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4911     else
4912         callback.register('process_input_buffer',Babel.callback)
4913     end
4914 end
4915 function Babel.addpatterns(pp, lg)
4916     local lg = lang.new(lg)
4917     local pats = lang.patterns(lg) or ''
4918     lang.clear_patterns(lg)
4919     for p in pp:gmatch('[^%s]+') do
4920         ss = ''
4921         for i in string.utfcharacters(p:gsub('%d', '')) do
4922             ss = ss .. '%d?' .. i
4923         end
4924         ss = ss:gsub('^%%d%?%.','%%%.') .. '%d?'
4925         ss = ss:gsub('%.%%d%?$', '%%%.')
4926         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4927         if n == 0 then
4928             tex.sprint(
4929                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4930                 .. p .. [[]])
4931             pats = pats .. ' ' .. p
4932         else
4933             tex.sprint(
4934                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4935                 .. p .. [[]])
4936         end
4937     end
4938     lang.patterns(lg, pats)
4939 end
4940 }
4941 \endgroup
4942 \ifx\newattribute\@undefined\else
4943     \newattribute\bbl@attr@locale
4944     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4945     \AddBabelHook{luatex}{beforeextras}{%
4946         \setattribute\bbl@attr@locale\localeid}
4947 \fi
4948 \def\BabelStringsDefault{unicode}
4949 \let\luabbl@stop\relax
4950 \AddBabelHook{luatex}{encodedcommands}{%
4951     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4952     \ifx\bbl@tempa\bbl@tempb\else
4953         \directlua{Babel.begin_process_input()}%
4954     \def\luabbl@stop{%

```



```

4955 \directlua{Babel.end_process_input()}}%
4956 \fi}%
4957 \AddBabelHook{luatex}{stopcommands}{%
4958 \luabbbl@stop
4959 \let\luabbbl@stop\relax}
4960 \AddBabelHook{luatex}{patterns}{%
4961 \@ifundefined{bbl@hyphendata@the\language}%
4962 {\def\bbl@elt##1##2##3##4{%
4963 \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4964 \def\bbl@tempb{##3}%
4965 \ifx\bbl@tempb\empty\else % if not a synonymous
4966 \def\bbl@tempc{##3}{##4}}%
4967 \fi
4968 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4969 \fi}%
4970 \bbl@languages
4971 \@ifundefined{bbl@hyphendata@the\language}%
4972 {\bbl@info{No hyphenation patterns were set for\%
4973 language '#2'. Reported}}%
4974 {\expandafter\expandafter\expandafter\bbl@luapatterns
4975 \csname bbl@hyphendata@the\language\endcsname}}}%
4976 \@ifundefined{bbl@patterns@}{}%
4977 \begingroup
4978 \bbl@xin@{\, \number\language,}{\, \bbl@pttnlist}%
4979 \ifin@ \else
4980 \ifx\bbl@patterns@\empty \else
4981 \directlua{ Babel.addpatterns(
4982 [[\bbl@patterns@]], \number\language) }%
4983 \fi
4984 \@ifundefined{bbl@patterns@#1}%
4985 \empty
4986 {\directlua{ Babel.addpatterns(
4987 [[\space\csname bbl@patterns@#1\endcsname]],
4988 \number\language) }}%
4989 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4990 \fi
4991 \endgroup}%
4992 \bbl@exp{%
4993 \bbl@ifunset{bbl@prehc@\languagename}{}%
4994 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
4995 {\prehyphenchar=\bbl@cl{prehc}\relax}}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4996 \@onlypreamble\babelpatterns
4997 \AtEndOfPackage{%
4998 \newcommand\babelpatterns[2][\empty]{%
4999 \ifx\bbl@patterns@\relax
5000 \let\bbl@patterns@\empty
5001 \fi
5002 \ifx\bbl@pttnlist\empty \else
5003 \bbl@warning{%
5004 You must not intermingle \string\selectlanguage\space and\%
5005 \string\babelpatterns\space or some patterns will not\%
5006 be taken into account. Reported}%
5007 \fi
5008 \ifx\@empty#1%

```

```

5009     \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5010   \else
5011     \edef\bbl@tempb{\zap@space#1 \@empty}%
5012     \bbl@for\bbl@tempa\bbl@tempb{%
5013       \bbl@fixname\bbl@tempa
5014       \bbl@iflanguage\bbl@tempa{%
5015         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5016           \@ifundefined{bbl@patterns@\bbl@tempa}%
5017           \@empty
5018           {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5019           #2}}}%
5020   \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. *In progress*. Replace regular (ie, implicit) discretionary by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary are not touched. See Unicode UAX 14.

```

5021% TODO - to a lua file
5022 \directlua{
5023   Babel = Babel or {}
5024   Babel.linebreaking = Babel.linebreaking or {}
5025   Babel.linebreaking.before = {}
5026   Babel.linebreaking.after = {}
5027   Babel.locale = {} % Free to use, indexed with \localeid
5028   function Babel.linebreaking.add_before(func)
5029     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5030     table.insert(Babel.linebreaking.before , func)
5031   end
5032   function Babel.linebreaking.add_after(func)
5033     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5034     table.insert(Babel.linebreaking.after, func)
5035   end
5036 }
5037 \def\bbl@intraspace#1 #2 #3\@@{%
5038   \directlua{
5039     Babel = Babel or {}
5040     Babel.intraspaces = Babel.intraspaces or {}
5041     Babel.intraspaces['\csname bbl@sbcpr@\languagename\endcsname'] = %
5042       {b = #1, p = #2, m = #3}
5043     Babel.locale_props[\the\localeid].intraspace = %
5044       {b = #1, p = #2, m = #3}
5045   }}
5046 \def\bbl@intrapenalty#1\@@{%
5047   \directlua{
5048     Babel = Babel or {}
5049     Babel.intrapenalties = Babel.intrapenalties or {}
5050     Babel.intrapenalties['\csname bbl@sbcpr@\languagename\endcsname'] = #1
5051     Babel.locale_props[\the\localeid].intrapenalty = #1
5052   }}
5053 \begingroup
5054 \catcode`\%=12
5055 \catcode`\^=14
5056 \catcode`\'=12
5057 \catcode`\~=12
5058 \gdef\bbl@seaintraspace{^
5059   \let\bbl@seaintraspace\relax

```

```

5060 \directlua{
5061   Babel = Babel or {}
5062   Babel.sea_enabled = true
5063   Babel.sea_ranges = Babel.sea_ranges or {}
5064   function Babel.set_chranges (script, chrng)
5065     local c = 0
5066     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5067       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5068       c = c + 1
5069     end
5070   end
5071   function Babel.sea_disc_to_space (head)
5072     local sea_ranges = Babel.sea_ranges
5073     local last_char = nil
5074     local quad = 655360    ^^ 10 pt = 655360 = 10 * 65536
5075     for item in node.traverse(head) do
5076       local i = item.id
5077       if i == node.id'glyph' then
5078         last_char = item
5079       elseif i == 7 and item.subtype == 3 and last_char
5080         and last_char.char > 0x0C99 then
5081         quad = font.getfont(last_char.font).size
5082         for lg, rg in pairs(sea_ranges) do
5083           if last_char.char > rg[1] and last_char.char < rg[2] then
5084             lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
5085             local intraspace = Babel.intraspaces[lg]
5086             local intrapenalty = Babel.intrapenalties[lg]
5087             local n
5088             if intrapenalty ~= 0 then
5089               n = node.new(14, 0) ^^ penalty
5090               n.penalty = intrapenalty
5091               node.insert_before(head, item, n)
5092             end
5093             n = node.new(12, 13) ^^ (glue, spaceskip)
5094             node.setglue(n, intraspace.b * quad,
5095               intraspace.p * quad,
5096               intraspace.m * quad)
5097             node.insert_before(head, item, n)
5098             node.remove(head, item)
5099           end
5100         end
5101       end
5102     end
5103   end
5104 }^^
5105 \bbl@luahyphenate}
5106 \catcode`\%=14
5107 \gdef\bbl@cjkintraspaces{%
5108   \let\bbl@cjkintraspaces\relax
5109   \directlua{
5110     Babel = Babel or {}
5111     require'babel-data-cjk.lua'
5112     Babel.cjk_enabled = true
5113     function Babel.cjk_linebreak(head)
5114       local GLYPH = node.id'glyph'
5115       local last_char = nil
5116       local quad = 655360    % 10 pt = 655360 = 10 * 65536
5117       local last_class = nil
5118       local last_lang = nil

```

```

5119
5120     for item in node.traverse(head) do
5121         if item.id == GLYPH then
5122
5123             local lang = item.lang
5124
5125             local LOCALE = node.get_attribute(item,
5126                 luatexbase.registernumber'bbl@attr@locale')
5127             local props = Babel.locale_props[LOCALE]
5128
5129             local class = Babel.cjk_class[item.char].c
5130
5131             if class == 'cp' then class = 'cl' end % ]) as CL
5132             if class == 'id' then class = 'I' end
5133
5134             local br = 0
5135             if class and last_class and Babel.cjk_breaks[last_class][class] then
5136                 br = Babel.cjk_breaks[last_class][class]
5137             end
5138
5139             if br == 1 and props.linebreak == 'c' and
5140                 lang ~= \the\l@nohyphenation\space and
5141                 last_lang ~= \the\l@nohyphenation then
5142                 local intrapenalty = props.intrapenalty
5143                 if intrapenalty ~= 0 then
5144                     local n = node.new(14, 0) % penalty
5145                     n.penalty = intrapenalty
5146                     node.insert_before(head, item, n)
5147                 end
5148                 local intraspace = props.intraspace
5149                 local n = node.new(12, 13) % (glue, spaceskip)
5150                 node.setglue(n, intraspace.b * quad,
5151                     intraspace.p * quad,
5152                     intraspace.m * quad)
5153                 node.insert_before(head, item, n)
5154             end
5155
5156             if font.getfont(item.font) then
5157                 quad = font.getfont(item.font).size
5158             end
5159             last_class = class
5160             last_lang = lang
5161         else % if penalty, glue or anything else
5162             last_class = nil
5163         end
5164     end
5165     lang.hyphenate(head)
5166 end
5167 }%
5168 \bbl@luahyphenate}
5169 \gdef\bbl@luahyphenate{%
5170 \let\bbl@luahyphenate\relax
5171 \directlua{
5172     luatexbase.add_to_callback('hyphenate',
5173     function (head, tail)
5174         if Babel.linebreaking.before then
5175             for k, func in ipairs(Babel.linebreaking.before) do
5176                 func(head)
5177             end

```

```

5178     end
5179     if Babel.cjk_enabled then
5180         Babel.cjk_linebreak(head)
5181     end
5182     lang.hyphenate(head)
5183     if Babel.linebreaking.after then
5184         for k, func in ipairs(Babel.linebreaking.after) do
5185             func(head)
5186         end
5187     end
5188     if Babel.sea_enabled then
5189         Babel.sea_disc_to_space(head)
5190     end
5191 end,
5192 'Babel.hyphenate')
5193 }
5194 }
5195 \endgroup
5196 \def\bbl@provide@intraspace{%
5197   \bbl@ifunset{bbl@intsp@languagename}{}%
5198   {\expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else
5199     \bbl@xin@{\bbl@cl{lbrk}}{c}%
5200     \ifin@           % cjk
5201     \bbl@cjk@intraspace
5202     \directlua{
5203       Babel = Babel or {}
5204       Babel.locale_props = Babel.locale_props or {}
5205       Babel.locale_props[\the\localeid].linebreak = 'c'
5206     }%
5207     \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5208     \ifx\bbl@KVP@intrapenalty\@nil
5209       \bbl@intrapenalty0\@@
5210     \fi
5211   \else           % sea
5212     \bbl@sea@intraspace
5213     \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5214     \directlua{
5215       Babel = Babel or {}
5216       Babel.sea_ranges = Babel.sea_ranges or {}
5217       Babel.set_chranges('\bbl@cl{sbc}',
5218         '\bbl@cl{chrng}')
5219     }%
5220     \ifx\bbl@KVP@intrapenalty\@nil
5221       \bbl@intrapenalty0\@@
5222     \fi
5223   \fi
5224 \fi
5225 \ifx\bbl@KVP@intrapenalty\@nil\else
5226   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5227 \fi}}

```

13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few

characters have an additional key for the width (fullwidth vs. halfwidth), not yet used.
There is a separate file, defined below.

Work in progress.

Common stuff.

```
5228 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5229 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5230 \DisableBabelHook{babel-fontspec}
5231 <<Font selection>>
```

13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5232% TODO - to a lua file
5233 \directlua{
5234 Babel.script_blocks = {
5235   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5236             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5237   ['Armn'] = {{0x0530, 0x058F}},
5238   ['Beng'] = {{0x0980, 0x09FF}},
5239   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5240   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5241   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5242             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5243   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5244   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5245             {0xAB00, 0xAB2F}},
5246   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5247   % Don't follow strictly Unicode, which places some Coptic letters in
5248   % the 'Greek and Coptic' block
5249   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5250   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5251             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5252             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5253             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5254             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5255             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5256   ['Hebr'] = {{0x0590, 0x05FF}},
5257   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5258             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5259   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5260   ['Knda'] = {{0x0C80, 0x0CFF}},
5261   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5262             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5263             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5264   ['Lao'] = {{0x0E80, 0x0EFF}},
5265   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5266             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5267             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5268   ['Mahj'] = {{0x11150, 0x1117F}},
5269   ['Mlym'] = {{0x0D00, 0x0D7F}},
```

```

5270 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5271 ['Orya'] = {{0x0B00, 0x0B7F}},
5272 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5273 ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5274 ['Taml'] = {{0x0B80, 0x0BFF}},
5275 ['Telu'] = {{0x0C00, 0x0C7F}},
5276 ['Tfng'] = {{0x2D30, 0x2D7F}},
5277 ['Thai'] = {{0x0E00, 0x0E7F}},
5278 ['Tibt'] = {{0x0F00, 0x0FFF}},
5279 ['Vaii'] = {{0xA500, 0xA63F}},
5280 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5281 }
5282
5283 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5284 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5285 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5286
5287 function Babel.locale_map(head)
5288   if not Babel.locale_mapped then return head end
5289
5290   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5291   local GLYPH = node.id('glyph')
5292   local inmath = false
5293   local toloc_save
5294   for item in node.traverse(head) do
5295     local toloc
5296     if not inmath and item.id == GLYPH then
5297       % Optimization: build a table with the chars found
5298       if Babel.chr_to_loc[item.char] then
5299         toloc = Babel.chr_to_loc[item.char]
5300       else
5301         for lc, maps in pairs(Babel.loc_to_scr) do
5302           for _, rg in pairs(maps) do
5303             if item.char >= rg[1] and item.char <= rg[2] then
5304               Babel.chr_to_loc[item.char] = lc
5305               toloc = lc
5306               break
5307             end
5308           end
5309         end
5310       end
5311       % Now, take action, but treat composite chars in a different
5312       % fashion, because they 'inherit' the previous locale. Not yet
5313       % optimized.
5314       if not toloc and
5315         (item.char >= 0x0300 and item.char <= 0x036F) or
5316         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5317         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5318         toloc = toloc_save
5319       end
5320       if toloc and toloc > -1 then
5321         if Babel.locale_props[toloc].lg then
5322           item.lang = Babel.locale_props[toloc].lg
5323           node.set_attribute(item, LOCALE, toloc)
5324         end
5325         if Babel.locale_props[toloc]['/'..item.font] then
5326           item.font = Babel.locale_props[toloc]['/'..item.font]
5327         end
5328         toloc_save = toloc

```

```

5329     end
5330     elseif not inmath and item.id == 7 then
5331         item.replace = item.replace and Babel.locale_map(item.replace)
5332         item.pre      = item.pre and Babel.locale_map(item.pre)
5333         item.post     = item.post and Babel.locale_map(item.post)
5334     elseif item.id == node.id'math' then
5335         inmath = (item.subtype == 0)
5336     end
5337 end
5338 return head
5339 end
5340 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

5341 \newcommand\babelcharproperty[1]{%
5342   \count@=#1\relax
5343   \ifvmode
5344     \expandafter\bbl@chprop
5345   \else
5346     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5347               vertical mode (preamble or between paragraphs)}%
5348     {See the manual for futher info}%
5349   \fi}
5350 \newcommand\bbl@chprop[3][\the\count@]{%
5351   \@tempcnta=#1\relax
5352   \bbl@ifunset\bbl@chprop@#2}%
5353   {\bbl@error{No property named '#2'. Allowed values are\\%
5354             direction (bc), mirror (bmg), and linebreak (lb)}%
5355     {See the manual for futher info}}%
5356   }%
5357   \loop
5358     \bbl@cs{chprop@#2}{#3}%
5359   \ifnum\count@<\@tempcnta
5360     \advance\count@\@ne
5361   \repeat}
5362 \def\bbl@chprop@direction#1{%
5363   \directlua{
5364     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5365     Babel.characters[\the\count@]['d'] = '#1'
5366   }}
5367 \let\bbl@chprop@bc\bbl@chprop@direction
5368 \def\bbl@chprop@mirror#1{%
5369   \directlua{
5370     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5371     Babel.characters[\the\count@]['m'] = '\number#1'
5372   }}
5373 \let\bbl@chprop@bmg\bbl@chprop@mirror
5374 \def\bbl@chprop@linebreak#1{%
5375   \directlua{
5376     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5377     Babel.cjk_characters[\the\count@]['c'] = '#1'
5378   }}
5379 \let\bbl@chprop@lb\bbl@chprop@linebreak
5380 \def\bbl@chprop@locale#1{%
5381   \directlua{
5382     Babel.chr_to_loc = Babel.chr_to_loc or {}
5383     Babel.chr_to_loc[\the\count@] =
5384       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space

```



```
5385   }}
```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```
5386 \begingroup % TODO - to a lua file
5387 \catcode`\#=12
5388 \catcode`\%=12
5389 \catcode`\&=14
5390 \directlua{
5391   Babel.linebreaking.replacements = {}
5392   Babel.linebreaking.replacements[0] = {}  &% pre
5393   Babel.linebreaking.replacements[1] = {}  &% post
5394
5395   &% Discretionaries contain strings as nodes
5396   function Babel.str_to_nodes(fn, matches, base)
5397     local n, head, last
5398     if fn == nil then return nil end
5399     for s in string.utfvalues(fn(matches)) do
5400       if base.id == 7 then
5401         base = base.replace
5402       end
5403       n = node.copy(base)
5404       n.char = s
5405       if not head then
5406         head = n
5407       else
5408         last.next = n
5409       end
5410       last = n
5411     end
5412     return head
5413   end
5414
5415   Babel.fetch_subtext = {}
5416
5417   Babel.fetch_subtext[1] = function(head)
5418     local word_string = ''
5419     local word_nodes = {}
5420     local lang
5421     local item = head
5422     local inmath = false
5423     local mode = 0 &%%%'word' first steps in merging with subtext
5424
5425     while item do
5426
5427       if item.id == 29
5428         and not(item.char == 124) &% ie, not |
```

```

5429         and not(item.char == 61) &% ie, not =
5430         and not inmath
5431         and (item.lang == lang or lang == nil) then
5432         lang = lang or item.lang
5433         word_string = word_string .. unicode.utf8.char(item.char)
5434         word_nodes[#word_nodes+1] = item
5435
5436     elseif item.id == 7 and item.subtype == 2
5437         and not inmath and mode == 0 then
5438         word_string = word_string .. '='
5439         word_nodes[#word_nodes+1] = item
5440
5441     elseif item.id == 7 and item.subtype == 3
5442         and not inmath and mode == 0 then
5443         word_string = word_string .. '|'
5444         word_nodes[#word_nodes+1] = item
5445
5446     elseif item.id == 11 and item.subtype == 0 then
5447         inmath = true
5448
5449     elseif mode > 0 and item.id == 12 and item.subtype == 13 then
5450         word_string = word_string .. '|'
5451         word_nodes[#word_nodes+1] = item
5452
5453     elseif word_string == '' then
5454         &% pass
5455
5456     else
5457         return word_string, word_nodes, item, lang
5458     end
5459
5460     item = item.next
5461 end
5462 end
5463
5464 &%%&
5465 &% Preliminary code for \babelprehyphenation
5466 &% TODO. Copypaste pattern. Merge
5467 Babel.fetch_subtext[0] = function(head)
5468     local word_string = ''
5469     local word_nodes = {}
5470     local lang
5471     local item = head
5472     local inmath = false
5473
5474     while item do
5475
5476         if item.id == 29 then
5477             local locale = node.get_attribute(item, Babel.attr_locale)
5478
5479             if not(item.char == 124) &% ie, not | = space
5480                 and not inmath
5481                 and (locale == lang or lang == nil) then
5482                 lang = lang or locale
5483                 word_string = word_string .. unicode.utf8.char(item.char)
5484                 word_nodes[#word_nodes+1] = item
5485             end
5486
5487             if item == node.tail(head) then

```

```

5488         item = nil
5489         return word_string, word_nodes, item, lang
5490     end
5491
5492     elseif item.id == 12 and item.subtype == 13 and not inmath then
5493         word_string = word_string .. '|'
5494         word_nodes[#word_nodes+1] = item
5495
5496         if item == node.tail(head) then
5497             item = nil
5498             return word_string, word_nodes, item, lang
5499         end
5500
5501     elseif item.id == 11 and item.subtype == 0 then
5502         inmath = true
5503
5504     elseif word_string == '' then
5505         &% pass
5506
5507     else
5508         return word_string, word_nodes, item, lang
5509     end
5510 end
5511
5512 item = item.next
5513 end
5514 end
5515
5516 function Babel.pre_hyphenate_replace(head)
5517     Babel.hyphenate_replace(head, 0)
5518 end
5519
5520 function Babel.post_hyphenate_replace(head)
5521     Babel.hyphenate_replace(head, 1)
5522 end
5523
5524 function Babel.hyphenate_replace(head, mode)
5525     local u = unicode.utf8
5526     local lbkr = Babel.linebreaking.replacements[mode]
5527
5528     local word_head = head
5529
5530     while true do &% for each subtext block
5531
5532         local w, wn, nw, lang = Babel.fetch_subtext[mode](word_head)
5533         if not lang then return head end
5534
5535         if not lbkr[lang] then
5536             break
5537         end
5538
5539         &% For each saved (pre|post)hyphenation
5540         for k=1, #lbkr[lang] do
5541             local p = lbkr[lang][k].pattern
5542             local r = lbkr[lang][k].replace
5543
5544             &% This variable is set in some cases below to the first *byte*
5545             &% after the match, either as found by u.match (faster) or the
5546             &% computed position based on sc if w has changed.

```

```

5547     local last_match = 0
5548
5549     %% For every match.
5550     while true do
5551         local new  %% used when inserting and removing nodes
5552         local refetch = false
5553
5554         local matches = { u.match(w, p, last_match) }
5555         if #matches < 2 then break end
5556
5557         %% Get and remove empty captures (with ()), which return a
5558         %% number with the position), and keep actual captures
5559         %% (from (...)), if any, in matches.
5560         local first = table.remove(matches, 1)
5561         local last  = table.remove(matches, #matches)
5562         local save_last = last  %% with A()BC()D, points to D
5563
5564         %% Fix offsets, from bytes to unicode. Explained above.
5565         first = u.len(w:sub(1, first-1)) + 1
5566         last  = u.len(w:sub(1, last-1))  %% now last points to C
5567
5568         if Babel.debug then
5569             print(p)
5570             print('', 'sc', 'first', 'last', 'last_m', 'w')
5571         end
5572
5573         %% This loop traverses the matched substring and takes the
5574         %% corresponding action stored in the replacement list.
5575         %% sc = the position in substr nodes / string
5576         %% rc = the replacement table index
5577         local sc = first-1
5578         local rc = 0
5579         while rc < last-first+1 do %% for each replacement
5580             sc = sc + 1
5581             rc = rc + 1
5582             local crep = r[rc]
5583             local char_node = wn[sc]
5584             local char_base = char_node
5585             local end_replacement = false
5586
5587             if crep and crep.data then
5588                 char_base = wn[crep.data+first-1]
5589             end
5590
5591             if Babel.debug then
5592                 print('*', sc, first, last, last_match, w)
5593             end
5594
5595             if crep and next(crep) == nil then %% {}
5596                 last_match = save_last
5597
5598             elseif crep == nil then %% remove
5599                 node.remove(head, char_node)
5600                 table.remove(wn, sc)
5601                 w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5602                 last_match = utf8.offset(w, sc)
5603                 sc = sc - 1  %% Nothing has been inserted
5604
5605             elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then

```

```

5606         local d = node.new(7, 0)    %% (disc, discretionary)
5607         d.pre      = Babel.str_to_nodes(crep.pre, matches, char_base)
5608         d.post      = Babel.str_to_nodes(crep.post, matches, char_base)
5609         d.replace    = Babel.str_to_nodes(crep.no, matches, char_base)
5610         d.attr = char_base.attr
5611         if crep.pre == nil then    %% TeXbook p96
5612             d.penalty = crep.penalty or tex.hyphenpenalty
5613         else
5614             d.penalty = crep.penalty or tex.exhyphenpenalty
5615         end
5616         head, new = node.insert_before(head, char_node, d)
5617         end_replacement = true
5618
5619     elseif crep and crep.penalty then
5620         local d = node.new(14, 0)    %% (penalty, userpenalty)
5621         d.attr = char_base.attr
5622         d.penalty = crep.penalty
5623         head, new = node.insert_before(head, char_node, d)
5624         end_replacement = true
5625
5626     elseif crep and crep.string then
5627         local str = crep.string(matches)
5628         if str == '' then    %% Gather with nil
5629             refetch = true
5630             if sc == 1 then
5631                 word_head = char_node.next
5632             end
5633             head, new = node.remove(head, char_node)
5634         elseif char_node.id == 29 and u.len(str) == 1 then
5635             char_node.char = string.utfvalue(str)
5636             w = u.sub(w, 1, sc-1) .. str .. u.sub(w, sc+1)
5637             last_match = utf8.offset(w, sc+1)
5638         else
5639             refetch = true
5640             local n
5641             for s in string.utfvalues(str) do
5642                 if char_node.id == 7 then
5643                     log('Automatic hyphens cannot be replaced, just removed.')
5644                 else
5645                     n = node.copy(char_base)
5646                 end
5647                 n.char = s
5648                 if sc == 1 then
5649                     head, new = node.insert_before(head, char_node, n)
5650                     word_head = new
5651                 else
5652                     node.insert_before(head, char_node, n)
5653                 end
5654             end
5655             node.remove(head, char_node)
5656             end    %% string length
5657         end    %% if char and char.string (ie replacement cases)
5658
5659     %% Shared by disc and penalty. With them, the inserted item
5660     %% does NOT go to w because it's neither = nor | nor a
5661     %% char.
5662     if end_replacement then
5663         if sc == 1 then
5664             word_head = new

```

```

5665         end
5666         if crep.insert then
5667             last_match = save_last
5668         else
5669             node.remove(head, char_node)
5670             table.remove(wn, sc)
5671             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5672             last_match = utf8.offset(w, sc)
5673         end
5674     end
5675 end   %% for each replacement
5676
5677 if Babel.debug then
5678     print('/', sc, first, last, last_match, w)
5679 end
5680
5681 %% TODO. refetch must be eventually unnecessary
5682 if refetch then
5683     w, wn, nw, lang = Babel.fetch_subtext[mode](word_head)
5684 end
5685
5686     end   %% for match
5687 end   %% for patterns
5688 word_head = nw
5689 end   %% for words
5690 return head
5691 end
5692
5693 %% This table stores capture maps, numbered consecutively
5694 Babel.capture_maps = {}
5695
5696 %% The following functions belong to the next macro
5697 function Babel.capture_func(key, cap)
5698     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
5699     ret = ret:gsub('{{[0-9]}|([^\]|+)|(-)}', Babel.capture_func_map)
5700     ret = ret:gsub("%[%[%]%.%", '')
5701     ret = ret:gsub("%.%[%[%]%", '')
5702     return key .. [[=function(m) return ]] .. ret .. [[ end]]
5703 end
5704
5705 function Babel.capt_map(from, mapno)
5706     return Babel.capture_maps[mapno][from] or from
5707 end
5708
5709 %% Handle the {n|abc|ABC} syntax in captures
5710 function Babel.capture_func_map(capno, from, to)
5711     local froms = {}
5712     for s in string.utfcharacters(from) do
5713         table.insert(froms, s)
5714     end
5715     local cnt = 1
5716     table.insert(Babel.capture_maps, {})
5717     local mlen = table.getn(Babel.capture_maps)
5718     for s in string.utfcharacters(to) do
5719         Babel.capture_maps[mlen][froms[cnt]] = s
5720         cnt = cnt + 1
5721     end
5722     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5723         (mlen) .. ").." .. "["

```

```

5724 end
5725 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ - 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).

```

5726 \catcode`\#=6
5727 \gdef\babelposthyphenation#1#2#3{&%
5728   \bbl@activateposthyphen
5729   \begingroup
5730     \def\babeltempa{\bbl@add@list\babeltempb}&%
5731     \let\babeltempb\@empty
5732     \bbl@foreach{#3}{&%
5733       \bbl@ifsamestring{##1}{remove}&%
5734       {\bbl@add@list\babeltempb{nil}}&%
5735       {\directlua{
5736         local rep = [[#1]]
5737         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5738         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5739         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5740         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5741         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5742         tex.print([[\\string\babeltempa{}}] .. rep .. [[]]})
5743       }}&%
5744     \directlua{
5745       local lbkr = Babel.linebreaking.replacements[1]
5746       local u = unicode.utf8
5747       &% Convert pattern:
5748       local patt = string.gsub(==[#2]==, '%s', '')
5749       if not u.find(patt, '()', nil, true) then
5750         patt = '()' .. patt .. '()'
5751       end
5752       patt = string.gsub(patt, '%(%)%^\s', '^()')
5753       patt = string.gsub(patt, '%$$(%)', '()$')
5754       patt = u.gsub(patt, '{(.)}',
5755         function (n)
5756           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5757         end)
5758       lbkr[\\the\\csname l@#1\\endcsname] = lbkr[\\the\\csname l@#1\\endcsname] or {}
5759       table.insert(lbkr[\\the\\csname l@#1\\endcsname],
5760         { pattern = patt, replace = { \babeltempb } })
5761     }&%
5762   \endgroup}
5763 % TODO. Working !!! Copypaste pattern.
5764 \gdef\babelprehyphenation#1#2#3{&%
5765   \bbl@activateprehyphen
5766   \begingroup
5767     \def\babeltempa{\bbl@add@list\babeltempb}&%
5768     \let\babeltempb\@empty
5769     \bbl@foreach{#3}{&%
5770       \bbl@ifsamestring{##1}{remove}&%

```

```

5771     {\bbl@add@list\babeltempb{nil}}&%
5772     {\directlua{
5773         local rep = {[#1]}
5774         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5775         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5776         tex.print([[string\babeltempa{}}] .. rep .. [{}]])
5777     }}&%
5778     \directlua{
5779         local lbkr = Babel.linebreaking.replacements[0]
5780         local u = unicode.utf8
5781         &% Convert pattern:
5782         local patt = string.gsub([==[#2]==], '%s', '')
5783         if not u.find(patt, '()', nil, true) then
5784             patt = '()' .. patt .. '()'
5785         end
5786         &% patt = string.gsub(patt, '%(%)%', '^()')
5787         &% patt = string.gsub(patt, '([^\%])%$%()', '%1()$')
5788         patt = u.gsub(patt, '{(.)}',
5789             function (n)
5790                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5791             end)
5792         lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5793         table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5794             { pattern = patt, replace = { \babeltempb } })
5795     }&%
5796     \endgroup}
5797 \endgroup
5798 \def\bbl@activateposthyphen{%
5799     \let\bbl@activateposthyphen\relax
5800     \directlua{
5801         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5802     }}
5803 % TODO. Working !!!
5804 \def\bbl@activateprehyphen{%
5805     \let\bbl@activateprehyphen\relax
5806     \directlua{
5807         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5808     }}

```

13.7 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5809 \bbl@trace{Redefinitions for bidi layout}
5810 \ifx\@eqnnum\undefined\else
5811     \ifx\bbl@attr@dir\undefined\else
5812         \edef\@eqnnum{%

```



```

5813      \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5814      \unexpanded\expandafter{\@eqnnum}}}}
5815  \fi
5816 \fi
5817 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5818 \ifnum\bbl@bidimode>\z@
5819   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5820     \bbl@exp{%
5821       \mathdir\the\bodydir
5822       #1%           Once entered in math, set boxes to restore values
5823       \<ifmmode>%
5824       \everyvbox{%
5825         \the\everyvbox
5826         \bodydir\the\bodydir
5827         \mathdir\the\mathdir
5828         \everyhbox{\the\everyhbox}%
5829         \everyvbox{\the\everyvbox}}%
5830       \everyhbox{%
5831         \the\everyhbox
5832         \bodydir\the\bodydir
5833         \mathdir\the\mathdir
5834         \everyhbox{\the\everyhbox}%
5835         \everyvbox{\the\everyvbox}}%
5836       \<fi>}}}%
5837   \def\@hangfrom#1{%
5838     \setbox\@tempboxa\hbox{#1}}}%
5839   \hangindent\wd\@tempboxa
5840   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5841     \shapemode\@ne
5842   \fi
5843   \noindent\box\@tempboxa}
5844 \fi
5845 \IfBabelLayout{tabular}
5846   {\let\bbl@OL@tabular\@tabular
5847    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5848    \let\bbl@NL@tabular\@tabular
5849    \AtBeginDocument{%
5850      \ifx\bbl@NL@tabular\@tabular\else
5851        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5852        \let\bbl@NL@tabular\@tabular
5853      \fi}}
5854   {}
5855 \IfBabelLayout{lists}
5856   {\let\bbl@OL@list\list
5857    \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5858    \let\bbl@NL@list\list
5859    \def\bbl@listparshape#1#2#3{%
5860      \parshape #1 #2 #3 %
5861      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5862        \shapemode\tw@
5863      \fi}}
5864   {}
5865 \IfBabelLayout{graphics}
5866   {\let\bbl@pictresetdir\relax
5867    \def\bbl@pictsetdir{%
5868      \ifcase\bbl@thetextdir
5869        \let\bbl@pictresetdir\relax
5870      \else
5871        \textdir TLT\relax

```

```

5872 \def\bbl@pictresetdir{\textdir TRT\relax}%
5873 \fi}%
5874 \let\bbl@OL@@picture\@picture
5875 \let\bbl@OL@put\put
5876 \bbl@sreplace\@picture{\hskip-}\{\bbl@pictsetdir\hskip-}%
5877 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5878 \killglue
5879 \raise#2\unitlength
5880 \hb@xt@z@{\kern#1\unitlength{\bbl@pictresetdir#3}\hss}}%
5881 \AtBeginDocument
5882 {\ifx\tikz@atbegin@node\undefined\else
5883 \let\bbl@OL@pgfpicture\pgfpicture
5884 \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
5885 {\bbl@pictsetdir\pgfpicturetrue}%
5886 \bbl@addpgfsys@beginpicture{\bbl@pictsetdir}%
5887 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5888 \fi}}
5889 {}

```

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.

```

5890 \IfBabelLayout{counters}%
5891 {\let\bbl@OL@@textsuperscript\textsuperscript
5892 \bbl@sreplace\textsuperscript{\m@th}\{\m@th\mathdir\pagedir}%
5893 \let\bbl@latin@arabic=\@arabic
5894 \let\bbl@OL@@arabic\@arabic
5895 \def\@arabic#1{\babelsublr{\bbl@latin@arabic#1}}%
5896 \@ifpackagewith{babel}{bidi=default}%
5897 {\let\bbl@asciroman=\@roman
5898 \let\bbl@OL@@roman\@roman
5899 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5900 \let\bbl@asciRoman=\@Roman
5901 \let\bbl@OL@@roman\@Roman
5902 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
5903 \let\bbl@OL@labelenumii\labelenumii
5904 \def\labelenumii{}\theenumii}%
5905 \let\bbl@OL@p@enumiii\p@enumiii
5906 \def\p@enumiii{\p@enumii}\theenumii{}\{\}\{\}
5907 <<Footnote changes>>
5908 \IfBabelLayout{footnotes}%
5909 {\let\bbl@OL@footnote\footnote
5910 \BabelFootnote\footnote\languagename{}\{\}%
5911 \BabelFootnote\localfootnote\languagename{}\{\}%
5912 \BabelFootnote\mainfootnote{}\{\}\{\}
5913 {}

```

Some \LaTeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5914 \IfBabelLayout{extras}%
5915 {\let\bbl@OL@underline\underline
5916 \bbl@sreplace\underline{\$@@underline}\{\bbl@nextfake\$@@underline}%
5917 \let\bbl@OL@LaTeX2e\LaTeX2e
5918 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5919 \if b\expandafter\@car\@series\@nil\boldmath\fi
5920 \babelsublr{%
5921 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
5922 {}
5923 </luatex>

```

13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (`<l>`, `<r>` or `<al>`).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```
5924 (*basic-r)
5925 Babel = Babel or {}
5926
5927 Babel.bidi_enabled = true
5928
5929 require('babel-data-bidi.lua')
5930
5931 local characters = Babel.characters
5932 local ranges = Babel.ranges
5933
5934 local DIR = node.id("dir")
5935
5936 local function dir_mark(head, from, to, outer)
5937   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
```

```

5938 local d = node.new(DIR)
5939 d.dir = '+' .. dir
5940 node.insert_before(head, from, d)
5941 d = node.new(DIR)
5942 d.dir = '-' .. dir
5943 node.insert_after(head, to, d)
5944 end
5945
5946 function Babel.bidi(head, ispar)
5947   local first_n, last_n      -- first and last char with nums
5948   local last_es              -- an auxiliary 'last' used with nums
5949   local first_d, last_d      -- first and last char in L/R block
5950   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):

```

5951 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5952 local strong_lr = (strong == 'l') and 'l' or 'r'
5953 local outer = strong
5954
5955 local new_dir = false
5956 local first_dir = false
5957 local inmath = false
5958
5959 local last_lr
5960
5961 local type_n = ''
5962
5963 for item in node.traverse(head) do
5964
5965   -- three cases: glyph, dir, otherwise
5966   if item.id == node.id'glyph'
5967     or (item.id == 7 and item.subtype == 2) then
5968
5969     local itemchar
5970     if item.id == 7 and item.subtype == 2 then
5971       itemchar = item.replace.char
5972     else
5973       itemchar = item.char
5974     end
5975     local chardata = characters[itemchar]
5976     dir = chardata and chardata.d or nil
5977     if not dir then
5978       for nn, et in ipairs(ranges) do
5979         if itemchar < et[1] then
5980           break
5981         elseif itemchar <= et[2] then
5982           dir = et[3]
5983           break
5984         end
5985       end
5986     end
5987     dir = dir or 'l'
5988     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).

```

5989     if new_dir then
5990         attr_dir = 0
5991         for at in node.traverse(item.attr) do
5992             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5993                 attr_dir = at.value % 3
5994             end
5995         end
5996         if attr_dir == 1 then
5997             strong = 'r'
5998         elseif attr_dir == 2 then
5999             strong = 'al'
6000         else
6001             strong = 'l'
6002         end
6003         strong_lr = (strong == 'l') and 'l' or 'r'
6004         outer = strong_lr
6005         new_dir = false
6006     end
6007
6008     if dir == 'nsm' then dir = strong end -- W1

```

Numbers. The dual `<al>/<r>` system for R is somewhat cumbersome.

```

6009     dir_real = dir -- We need dir_real to set strong below
6010     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:

```

6011     if strong == 'al' then
6012         if dir == 'en' then dir = 'an' end -- W2
6013         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6014         strong_lr = 'r' -- W3
6015     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

6016     elseif item.id == node.id'dir' and not inmath then
6017         new_dir = true
6018         dir = nil
6019     elseif item.id == node.id'math' then
6020         inmath = (item.subtype == 0)
6021     else
6022         dir = nil -- Not a char
6023     end

```

Numbers in R mode. A sequence of `<en>`, `<et>`, `<an>`, `<es>` and `<cs>` is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the `texdir` is set. This means you cannot insert, say, a `whatsit`, but this is what I would expect (with `luacolor` you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only `<an>` is relevant if `<al>`.

```

6024     if dir == 'en' or dir == 'an' or dir == 'et' then
6025         if dir ~= 'et' then
6026             type_n = dir
6027         end
6028         first_n = first_n or item
6029         last_n = last_es or item

```

```

6030     last_es = nil
6031   elseif dir == 'es' and last_n then -- W3+W6
6032     last_es = item
6033   elseif dir == 'cs' then           -- it's right - do nothing
6034   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6035     if strong_lr == 'r' and type_n ~= '' then
6036       dir_mark(head, first_n, last_n, 'r')
6037     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6038       dir_mark(head, first_n, last_n, 'r')
6039       dir_mark(head, first_d, last_d, outer)
6040       first_d, last_d = nil, nil
6041     elseif strong_lr == 'l' and type_n ~= '' then
6042       last_d = last_n
6043     end
6044     type_n = ''
6045     first_n, last_n = nil, nil
6046   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:

```

6047   if dir == 'l' or dir == 'r' then
6048     if dir ~= outer then
6049       first_d = first_d or item
6050       last_d = item
6051     elseif first_d and dir ~= strong_lr then
6052       dir_mark(head, first_d, last_d, outer)
6053       first_d, last_d = nil, nil
6054     end
6055   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.

```

6056   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6057     item.char = characters[item.char] and
6058               characters[item.char].m or item.char
6059   elseif (dir or new_dir) and last_lr ~= item then
6060     local mir = outer .. strong_lr .. (dir or outer)
6061     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6062       for ch in node.traverse(node.next(last_lr)) do
6063         if ch == item then break end
6064         if ch.id == node.id'glyph' and characters[ch.char] then
6065           ch.char = characters[ch.char].m or ch.char
6066         end
6067       end
6068     end
6069   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).

```

6070   if dir == 'l' or dir == 'r' then
6071     last_lr = item
6072     strong = dir_real           -- Don't search back - best save now
6073     strong_lr = (strong == 'l') and 'l' or 'r'

```

```

6074     elseif new_dir then
6075         last_lr = nil
6076     end
6077 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

6078 if last_lr and outer == 'r' then
6079     for ch in node.traverse_id(node.id('glyph', node.next(last_lr))) do
6080         if characters[ch.char] then
6081             ch.char = characters[ch.char].m or ch.char
6082         end
6083     end
6084 end
6085 if first_n then
6086     dir_mark(head, first_n, last_n, outer)
6087 end
6088 if first_d then
6089     dir_mark(head, first_d, last_d, outer)
6090 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

6091 return node.prev(head) or head
6092 end
6093 </basic-r>

```

And here the Lua code for bidi=basic:

```

6094 <(*basic)
6095 Babel = Babel or {}
6096
6097 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6098
6099 Babel.fontmap = Babel.fontmap or {}
6100 Babel.fontmap[0] = {}      -- l
6101 Babel.fontmap[1] = {}      -- r
6102 Babel.fontmap[2] = {}      -- al/an
6103
6104 Babel.bidi_enabled = true
6105 Babel.mirroring_enabled = true
6106
6107 require('babel-data-bidi.lua')
6108
6109 local characters = Babel.characters
6110 local ranges = Babel.ranges
6111
6112 local DIR = node.id('dir')
6113 local GLYPH = node.id('glyph')
6114
6115 local function insert_implicit(head, state, outer)
6116     local new_state = state
6117     if state.sim and state.eim and state.sim ~= state.eim then
6118         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6119         local d = node.new(DIR)
6120         d.dir = '+' .. dir
6121         node.insert_before(head, state.sim, d)
6122         local d = node.new(DIR)
6123         d.dir = '-' .. dir
6124         node.insert_after(head, state.eim, d)
6125     end

```

```

6126 new_state.sim, new_state.eim = nil, nil
6127 return head, new_state
6128 end
6129
6130 local function insert_numeric(head, state)
6131   local new
6132   local new_state = state
6133   if state.san and state.ean and state.san ~= state.ean then
6134     local d = node.new(DIR)
6135     d.dir = '+TLT'
6136     _, new = node.insert_before(head, state.san, d)
6137     if state.san == state.sim then state.sim = new end
6138     local d = node.new(DIR)
6139     d.dir = '-TLT'
6140     _, new = node.insert_after(head, state.ean, d)
6141     if state.ean == state.eim then state.eim = new end
6142   end
6143   new_state.san, new_state.ean = nil, nil
6144   return head, new_state
6145 end
6146
6147 -- TODO - \hbox with an explicit dir can lead to wrong results
6148 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6149 -- was made to improve the situation, but the problem is the 3-dir
6150 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6151 -- well.
6152
6153 function Babel.bidi(head, ispar, hdir)
6154   local d -- d is used mainly for computations in a loop
6155   local prev_d = ''
6156   local new_d = false
6157
6158   local nodes = {}
6159   local outer_first = nil
6160   local inmath = false
6161
6162   local glue_d = nil
6163   local glue_i = nil
6164
6165   local has_en = false
6166   local first_et = nil
6167
6168   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6169
6170   local save_outer
6171   local temp = node.get_attribute(head, ATDIR)
6172   if temp then
6173     temp = temp % 3
6174     save_outer = (temp == 0 and 'l') or
6175                  (temp == 1 and 'r') or
6176                  (temp == 2 and 'al')
6177   elseif ispar then -- Or error? Shouldn't happen
6178     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6179   else -- Or error? Shouldn't happen
6180     save_outer = ('TRT' == hdir) and 'r' or 'l'
6181   end
6182   -- when the callback is called, we are just _after_ the box,
6183   -- and the textdir is that of the surrounding text
6184   -- if not ispar and hdir ~= tex.textdir then

```



```

6185 -- save_outer = ('TRT' == hdir) and 'r' or 'l'
6186 -- end
6187 local outer = save_outer
6188 local last = outer
6189 -- 'al' is only taken into account in the first, current loop
6190 if save_outer == 'al' then save_outer = 'r' end
6191
6192 local fontmap = Babel.fontmap
6193
6194 for item in node.traverse(head) do
6195
6196     -- In what follows, #node is the last (previous) node, because the
6197     -- current one is not added until we start processing the neutrals.
6198
6199     -- three cases: glyph, dir, otherwise
6200     if item.id == GLYPH
6201         or (item.id == 7 and item.subtype == 2) then
6202
6203         local d_font = nil
6204         local item_r
6205         if item.id == 7 and item.subtype == 2 then
6206             item_r = item.replace -- automatic discs have just 1 glyph
6207         else
6208             item_r = item
6209         end
6210         local chardata = characters[item_r.char]
6211         d = chardata and chardata.d or nil
6212         if not d or d == 'nsm' then
6213             for nn, et in ipairs(ranges) do
6214                 if item_r.char < et[1] then
6215                     break
6216                 elseif item_r.char <= et[2] then
6217                     if not d then d = et[3]
6218                     elseif d == 'nsm' then d_font = et[3]
6219                     end
6220                     break
6221                 end
6222             end
6223         end
6224         d = d or 'l'
6225
6226         -- A short 'pause' in bidi for mapfont
6227         d_font = d_font or d
6228         d_font = (d_font == 'l' and 0) or
6229             (d_font == 'nsm' and 0) or
6230             (d_font == 'r' and 1) or
6231             (d_font == 'al' and 2) or
6232             (d_font == 'an' and 2) or nil
6233         if d_font and fontmap and fontmap[d_font][item_r.font] then
6234             item_r.font = fontmap[d_font][item_r.font]
6235         end
6236
6237         if new_d then
6238             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6239             if inmath then
6240                 attr_d = 0
6241             else
6242                 attr_d = node.get_attribute(item, ATDIR)
6243                 attr_d = attr_d % 3

```

```

6244         end
6245         if attr_d == 1 then
6246             outer_first = 'r'
6247             last = 'r'
6248         elseif attr_d == 2 then
6249             outer_first = 'r'
6250             last = 'al'
6251         else
6252             outer_first = 'l'
6253             last = 'l'
6254         end
6255         outer = last
6256         has_en = false
6257         first_et = nil
6258         new_d = false
6259     end
6260
6261     if glue_d then
6262         if (d == 'l' and 'l' or 'r') ~= glue_d then
6263             table.insert(nodes, {glue_i, 'on', nil})
6264         end
6265         glue_d = nil
6266         glue_i = nil
6267     end
6268
6269     elseif item.id == DIR then
6270         d = nil
6271         new_d = true
6272
6273     elseif item.id == node.id'glue' and item.subtype == 13 then
6274         glue_d = d
6275         glue_i = item
6276         d = nil
6277
6278     elseif item.id == node.id'math' then
6279         inmath = (item.subtype == 0)
6280
6281     else
6282         d = nil
6283     end
6284
6285     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6286     if last == 'al' and d == 'en' then
6287         d = 'an'          -- W3
6288     elseif last == 'al' and (d == 'et' or d == 'es') then
6289         d = 'on'          -- W6
6290     end
6291
6292     -- EN + CS/ES + EN      -- W4
6293     if d == 'en' and #nodes >= 2 then
6294         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6295             and nodes[#nodes-1][2] == 'en' then
6296             nodes[#nodes][2] = 'en'
6297         end
6298     end
6299
6300     -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
6301     if d == 'an' and #nodes >= 2 then
6302         if (nodes[#nodes][2] == 'cs')

```

```

6303         and nodes[#nodes-1][2] == 'an' then
6304             nodes[#nodes][2] = 'an'
6305         end
6306     end
6307
6308     -- ET/EN          -- W5 + W7->l / W6->on
6309     if d == 'et' then
6310         first_et = first_et or (#nodes + 1)
6311     elseif d == 'en' then
6312         has_en = true
6313         first_et = first_et or (#nodes + 1)
6314     elseif first_et then      -- d may be nil here !
6315         if has_en then
6316             if last == 'l' then
6317                 temp = 'l'      -- W7
6318             else
6319                 temp = 'en'     -- W5
6320             end
6321         else
6322             temp = 'on'        -- W6
6323         end
6324         for e = first_et, #nodes do
6325             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6326         end
6327         first_et = nil
6328         has_en = false
6329     end
6330
6331     if d then
6332         if d == 'al' then
6333             d = 'r'
6334             last = 'al'
6335         elseif d == 'l' or d == 'r' then
6336             last = d
6337         end
6338         prev_d = d
6339         table.insert(nodes, {item, d, outer_first})
6340     end
6341
6342     outer_first = nil
6343
6344 end
6345
6346 -- TODO -- repeated here in case EN/ET is the last node. Find a
6347 -- better way of doing things:
6348 if first_et then      -- dir may be nil here !
6349     if has_en then
6350         if last == 'l' then
6351             temp = 'l'      -- W7
6352         else
6353             temp = 'en'     -- W5
6354         end
6355     else
6356         temp = 'on'        -- W6
6357     end
6358     for e = first_et, #nodes do
6359         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6360     end
6361 end

```

```

6362
6363 -- dummy node, to close things
6364 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6365
6366 ----- NEUTRAL -----
6367
6368 outer = save_outer
6369 last = outer
6370
6371 local first_on = nil
6372
6373 for q = 1, #nodes do
6374     local item
6375
6376     local outer_first = nodes[q][3]
6377     outer = outer_first or outer
6378     last = outer_first or last
6379
6380     local d = nodes[q][2]
6381     if d == 'an' or d == 'en' then d = 'r' end
6382     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6383
6384     if d == 'on' then
6385         first_on = first_on or q
6386     elseif first_on then
6387         if last == d then
6388             temp = d
6389         else
6390             temp = outer
6391         end
6392         for r = first_on, q - 1 do
6393             nodes[r][2] = temp
6394             item = nodes[r][1] -- MIRRORING
6395             if Babel.mirroring_enabled and item.id == GLYPH
6396                 and temp == 'r' and characters[item.char] then
6397                 local font_mode = font.fonts[item.font].properties.mode
6398                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6399                     item.char = characters[item.char].m or item.char
6400                 end
6401             end
6402         end
6403         first_on = nil
6404     end
6405
6406     if d == 'r' or d == 'l' then last = d end
6407 end
6408
6409 ----- IMPLICIT, REORDER -----
6410
6411 outer = save_outer
6412 last = outer
6413
6414 local state = {}
6415 state.has_r = false
6416
6417 for q = 1, #nodes do
6418
6419     local item = nodes[q][1]
6420

```

```

6421   outer = nodes[q][3] or outer
6422
6423   local d = nodes[q][2]
6424
6425   if d == 'nsm' then d = last end          -- W1
6426   if d == 'en' then d = 'an' end
6427   local isdir = (d == 'r' or d == 'l')
6428
6429   if outer == 'l' and d == 'an' then
6430     state.san = state.san or item
6431     state.ean = item
6432   elseif state.san then
6433     head, state = insert_numeric(head, state)
6434   end
6435
6436   if outer == 'l' then
6437     if d == 'an' or d == 'r' then        -- im -> implicit
6438       if d == 'r' then state.has_r = true end
6439       state.sim = state.sim or item
6440       state.eim = item
6441     elseif d == 'l' and state.sim and state.has_r then
6442       head, state = insert_implicit(head, state, outer)
6443     elseif d == 'l' then
6444       state.sim, state.eim, state.has_r = nil, nil, false
6445     end
6446   else
6447     if d == 'an' or d == 'l' then
6448       if nodes[q][3] then -- nil except after an explicit dir
6449         state.sim = item -- so we move sim 'inside' the group
6450       else
6451         state.sim = state.sim or item
6452       end
6453       state.eim = item
6454     elseif d == 'r' and state.sim then
6455       head, state = insert_implicit(head, state, outer)
6456     elseif d == 'r' then
6457       state.sim, state.eim = nil, nil
6458     end
6459   end
6460
6461   if isdir then
6462     last = d          -- Don't search back - best save now
6463   elseif d == 'on' and state.san then
6464     state.san = state.san or item
6465     state.ean = item
6466   end
6467
6468 end
6469
6470 return node.prev(head) or head
6471 end
6472 </basic>

```

14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

For the meaning of these codes, see the Unicode standard.

15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available. The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
6473 ⟨*nil⟩
6474 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
6475 \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.

```
6476 \ifx\l@nil\undefined
6477   \newlanguage\l@nil
6478   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
6479   \let\bbl@elt\relax
6480   \edef\bbl@languages{% Add it to the list of languages
6481     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
6482 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
6483 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 6484 \let\captionnil\@empty
6485 \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.

```
6486 \ldf@finish{nil}
6487 ⟨/nil⟩
```

16 Support for Plain T_EX (plain.def)

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

People can have a file `locallyhyphen.tex` or whatever they like, but they mustn't diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

```
6488 (*bplain | blplain)
6489 \catcode`\{=1 % left brace is begin-group character
6490 \catcode`\}=2 % right brace is end-group character
6491 \catcode`\#=6 % hash mark is macro parameter character
```

```
6492 \openin 0 hyphen.cfg
6493 \ifeof0
6494 \else
6495   \let\@input
```

```

6496 \def\input #1 {%
6497     \let\input\@
6498     \a hyphen.cfg
6499     \let\@undefined
6500 }
6501 \fi
6502 </bplain | bplain>

```

```
6503 <bplain>\a plain.tex
6504 <blplain>\a lplain.tex
```

```
6505 <bplain>\def\fmtname{babel-plain}
6506 <blplain>\def\fmtname{babel-lplain}
```

16.2 Emulating some L^AT_EX features

```

6507 <<{*Emulate LaTeX}> \equiv
6508 % == Code for plain ==
6509 \def\@empty{}
6510 \def\loadlocalcfg#1{%
6511   \openin0#1.cfg

```

```

6512 \ifeof0
6513 \closein0
6514 \else
6515 \closein0
6516 {\immediate\write16{*****}%
6517 \immediate\write16{* Local config file #1.cfg used}%
6518 \immediate\write16{*}%
6519 }
6520 \input #1.cfg\relax
6521 \fi
6522 \@endofldf}

```

16.3 General tools

A number of \LaTeX macro's that are needed later on.

```

6523 \long\def\@firstofone#1{#1}
6524 \long\def\@firstoftwo#1#2{#1}
6525 \long\def\@secondoftwo#1#2{#2}
6526 \def\@nnil{\@nil}
6527 \def\@gobbletwo#1#2{}
6528 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6529 \def\@star@or@long#1{%
6530 \@ifstar
6531 {\let\l@ngrel@x\relax#1}%
6532 {\let\l@ngrel@x\long#1}}
6533 \let\l@ngrel@x\relax
6534 \def\@car#1#2\@nil{#1}
6535 \def\@cdr#1#2\@nil{#2}
6536 \let\@typeset@protect\relax
6537 \let\protected@edef\edef
6538 \long\def\@gobble#1{}
6539 \edef\@backslashchar{\expandafter\@gobble\string\}
6540 \def\strip@prefix#1>{}
6541 \def\g@addto@macro#1#2{%
6542 \toks@\expandafter{#1#2}%
6543 \xdef#1{\the\toks@}}
6544 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6545 \def\@nameuse#1{\csname #1\endcsname}
6546 \def\@ifundefined#1{%
6547 \expandafter\ifx\csname#1\endcsname\relax
6548 \expandafter\@firstoftwo
6549 \else
6550 \expandafter\@secondoftwo
6551 \fi}
6552 \def\@expandtwoargs#1#2#3{%
6553 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6554 \def\zap@space#1 #2{%
6555 #1%
6556 \ifx#2\@empty\else\expandafter\zap@space\fi
6557 #2}
6558 \let\bbl@trace\@gobble

```

$\LaTeX 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

6559 \ifx\@preamblecmds\undefined
6560 \def\@preamblecmds{}
6561 \fi
6562 \def\@onlypreamble#1{%

```



```

6563 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6564 \@preamblecmds\do#1}}
6565 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

6566 \def\begindocument{%
6567 \@begindocumenthook
6568 \global\let\@begindocumenthook\@undefined
6569 \def\do##1{\global\let##1\@undefined}%
6570 \@preamblecmds
6571 \global\let\do\noexpand}

6572 \ifx\@begindocumenthook\@undefined
6573 \def\@begindocumenthook{}
6574 \fi
6575 \@onlypreamble\@begindocumenthook
6576 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

6577 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6578 \@onlypreamble\AtEndOfPackage
6579 \def\@endofldf{}
6580 \@onlypreamble\@endofldf
6581 \let\bbl@afterlang\@empty
6582 \chardef\bbl@opt@hyphenmap\z@

```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

6583 \catcode`\&=\z@
6584 \ifx&\if@files\@undefined
6585 \expandafter\let\csname if@files\expandafter\endcsname
6586 \csname iffalse\endcsname
6587 \fi
6588 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

6589 \def\newcommand{\@star@or@long\new@command}
6590 \def\new@command#1{%
6591 \@testopt{\@newcommand#1}0}
6592 \def\@newcommand#1[#2]{%
6593 \@ifnextchar [{\@xargdef#1[#2]}%
6594 {\@argdef#1[#2]}}
6595 \long\def\@argdef#1[#2]#3{%
6596 \@yargdef#1\@ne{#2}{#3}}
6597 \long\def\@xargdef#1[#2][#3]#4{%
6598 \expandafter\def\expandafter#1\expandafter{%
6599 \expandafter\@protected@testopt\expandafter #1%
6600 \csname\string#1\expandafter\endcsname{#3}}%
6601 \expandafter\@yargdef \csname\string#1\endcsname
6602 \tw@{#2}{#4}}
6603 \long\def\@yargdef#1#2#3{%
6604 \@tempcnta#3\relax
6605 \advance \@tempcnta \@ne
6606 \let\@hash@\relax
6607 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6608 \@tempcntb #2%

```

```

6609 \@whilenum\@tempcntb <\@tempcnta
6610 \do{%
6611   \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
6612   \advance\@tempcntb \@ne}%
6613   \let\@hash@##%
6614   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6615 \def\providecommand{\@star@or@long\provide@command}
6616 \def\provide@command#1{%
6617   \begingroup
6618   \escapechar\m@ne\xdef\@gtempa{\string#1}%
6619   \endgroup
6620   \expandafter\ifundefined\@gtempa
6621     {\def\reserved@a{\new@command#1}}%
6622     {\let\reserved@a\relax
6623     \def\reserved@a{\new@command\reserved@a}}%
6624   \reserved@a}%
6625 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6626 \def\declare@robustcommand#1{%
6627   \edef\reserved@a{\string#1}%
6628   \def\reserved@b{#1}%
6629   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6630   \edef#1{%
6631     \ifx\reserved@a\reserved@b
6632       \noexpand\x@protect
6633       \noexpand#1%
6634     \fi
6635     \noexpand\protect
6636     \expandafter\noexpand\csname
6637       \expandafter\@gobble\string#1 \endcsname
6638   }%
6639   \expandafter\new@command\csname
6640     \expandafter\@gobble\string#1 \endcsname
6641 }
6642 \def\x@protect#1{%
6643   \ifx\protect\@typeset@protect\else
6644     \@x@protect#1%
6645   \fi
6646 }
6647 \catcode`\&=\z@ % Trick to hide conditionals
6648 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

6649 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6650 \catcode`\&=4
6651 \ifx\in@\@undefined
6652   \def\in@#1#2{%
6653     \def\in@@##1##2##3\in@@{%
6654       \ifx\in@@##2\in@false\else\in@true\fi}%
6655     \in@@##1\in@\in@@}
6656 \else
6657   \let\bbl@tempa\@empty
6658 \fi
6659 \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).

```
6660 \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.

```
6661 \def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their \LaTeX 2_ε versions; just enough to make things work in plain \TeX environments.

```
6662 \ifx\@tempcnta\@undefined
6663   \csname newcount\endcsname\@tempcnta\relax
6664 \fi
6665 \ifx\@tempcntb\@undefined
6666   \csname newcount\endcsname\@tempcntb\relax
6667 \fi
```

To prevent wasting two counters in \LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
6668 \ifx\bye\@undefined
6669   \advance\count10 by -2\relax
6670 \fi
6671 \ifx\@ifnextchar\@undefined
6672   \def\@ifnextchar#1#2#3{%
6673     \let\reserved@d=#1%
6674     \def\reserved@a{#2}\def\reserved@b{#3}%
6675     \futurelet\@let@token\@ifnch}
6676   \def\@ifnch{%
6677     \ifx\@let@token\@sptoken
6678       \let\reserved@c\@xifnch
6679     \else
6680       \ifx\@let@token\reserved@d
6681         \let\reserved@c\reserved@a
6682       \else
6683         \let\reserved@c\reserved@b
6684       \fi
6685     \fi
6686     \reserved@c}
6687   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6688   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6689 \fi
6690 \def\@testopt#1#2{%
6691   \@ifnextchar[ {#1}{#1[#2]}}
6692 \def\@protected@testopt#1{%
6693   \ifx\protect\@typeset@protect
6694     \expandafter\@testopt
6695   \else
6696     \@x@protect#1%
6697   \fi}
6698 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6699   #2\relax}\fi}
6700 \long\def\@iwhilenum#1{\ifnum #1\relax\expandafter\@iwhilenum
6701   \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```
6702 \def\DeclareTextCommand{%
6703   \@dec@text@cmd\providecommand
6704 }
6705 \def\ProvideTextCommand{%
6706   \@dec@text@cmd\providecommand
6707 }
6708 \def\DeclareTextSymbol#1#2#3{%
6709   \@dec@text@cmd\chardef#1{#2}#3\relax
6710 }
6711 \def\@dec@text@cmd#1#2#3{%
6712   \expandafter\def\expandafter#2%
6713     \expandafter{%
6714       \csname#3-cmd\expandafter\endcsname
6715       \expandafter#2%
6716       \csname#3\string#2\endcsname
6717     }%
6718 %   \let\@ifdefinable\@rc@ifdefinable
6719   \expandafter#1\csname#3\string#2\endcsname
6720 }
6721 \def\@current@cmd#1{%
6722   \ifx\protect\@typeset@protect\else
6723     \noexpand#1\expandafter\@gobble
6724   \fi
6725 }
6726 \def\@changed@cmd#1#2{%
6727   \ifx\protect\@typeset@protect
6728     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6729       \expandafter\ifx\csname ?\string#1\endcsname\relax
6730         \expandafter\def\csname ?\string#1\endcsname{%
6731           \@changed@x@err{#1}%
6732         }%
6733       \fi
6734       \global\expandafter\let
6735         \csname\cf@encoding \string#1\expandafter\endcsname
6736         \csname ?\string#1\endcsname
6737     \fi
6738     \csname\cf@encoding\string#1%
6739       \expandafter\endcsname
6740   \else
6741     \noexpand#1%
6742   \fi
6743 }
6744 \def\@changed@x@err#1{%
6745   \errhelp{Your command will be ignored, type <return> to proceed}%
6746   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6747 \def\DeclareTextCommandDefault#1{%
6748   \DeclareTextCommand#1?%
6749 }
6750 \def\ProvideTextCommandDefault#1{%
6751   \ProvideTextCommand#1?%
6752 }
6753 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6754 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6755 \def\DeclareTextAccent#1#2#3{%
6756   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6757 }
```

```

6758 \def\DeclareTextCompositeCommand#1#2#3#4{%
6759   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6760   \edef\reserved@b{\string##1}%
6761   \edef\reserved@c{%
6762     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6763   \ifx\reserved@b\reserved@c
6764     \expandafter\expandafter\expandafter\ifx
6765       \expandafter\@car\reserved@a\relax\relax\@nil
6766       \@text@composite
6767     \else
6768       \edef\reserved@b##1{%
6769         \def\expandafter\noexpand
6770           \csname#2\string#1\endcsname####1{%
6771           \noexpand\@text@composite
6772           \expandafter\noexpand\csname#2\string#1\endcsname
6773           ####1\noexpand\@empty\noexpand\@text@composite
6774           {##1}%
6775         }%
6776       }%
6777       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6778     \fi
6779     \expandafter\def\csname\expandafter\string\csname
6780       #2\endcsname\string#1-\string#3\endcsname{#4}
6781   \else
6782     \errhelp{Your command will be ignored, type <return> to proceed}%
6783     \errmessage{\string\DeclareTextCompositeCommand\space used on
6784       inappropriate command \protect#1}
6785   \fi
6786 }
6787 \def\@text@composite#1#2#3\@text@composite{%
6788   \expandafter\@text@composite@x
6789     \csname\string#1-\string#2\endcsname
6790 }
6791 \def\@text@composite@x#1#2{%
6792   \ifx#1\relax
6793     #2%
6794   \else
6795     #1%
6796   \fi
6797 }
6798 %
6799 \def\@strip@args#1:#2-#3\@strip@args{#2}
6800 \def\DeclareTextComposite#1#2#3#4{%
6801   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6802   \bgroup
6803     \lcode`\@=#4%
6804     \lowercase{%
6805       \egroup
6806       \reserved@a @%
6807     }%
6808 }
6809 %
6810 \def\UseTextSymbol#1#2{#2}
6811 \def\UseTextAccent#1#2#3{}
6812 \def\@use@text@encoding#1{}
6813 \def\DeclareTextSymbolDefault#1#2{%
6814   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6815 }
6816 \def\DeclareTextAccentDefault#1#2{%

```

```

6817 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6818 }
6819 \def\cf@encoding{OT1}

```

Currently we only use the $\LaTeX 2_{\epsilon}$ method for accents for those that are known to be made active in *some* language definition file.

```

6820 \DeclareTextAccent{"}{OT1}{127}
6821 \DeclareTextAccent{'}{OT1}{19}
6822 \DeclareTextAccent{^}{OT1}{94}
6823 \DeclareTextAccent{`}{OT1}{18}
6824 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```

6825 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6826 \DeclareTextSymbol{\textquotedblright}{OT1}{93}
6827 \DeclareTextSymbol{\textquoteleft}{OT1}{94}
6828 \DeclareTextSymbol{\textquoteright}{OT1}{95}
6829 \DeclareTextSymbol{\i}{OT1}{16}
6830 \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`.

```

6831 \ifx\scriptsize\@undefined
6832 \let\scriptsize\sevenrm
6833 \fi
6834 % End of code for plain
6835 <\/Emulate LaTeX>

```

A proxy file:

```

6836 <*plain>
6837 \input babel.def
6838 </plain>

```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.

- [7] Leslie Lamport, in: $\text{T}_{\text{E}}\text{X}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German $\text{T}_{\text{E}}\text{X}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International $\text{E}_{\text{T}}\text{X}$ is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using $\text{E}_{\text{T}}\text{X}$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).