

Babel

Version 3.63.2517
2021/10/06

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	8
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	16
1.12	The base option	18
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	33
1.18	Dates	34
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Transforms	38
1.22	Selection based on BCP 47 tags	40
1.23	Selecting scripts	41
1.24	Selecting directions	42
1.25	Language attributes	46
1.26	Hooks	46
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	49
1.29	Tweaking some features	49
1.30	Tips, workarounds, known issues and notes	49
1.31	Current and future work	50
1.32	Tentative and experimental code	51
2	Loading languages with language.dat	51
2.1	Format	51
3	The interface between the core of babel and the language definition files	52
3.1	Guidelines for contributed languages	53
3.2	Basic macros	54
3.3	Skeleton	55
3.4	Support for active characters	56
3.5	Support for saving macro definitions	57
3.6	Support for extending macros	57
3.7	Macros common to a number of languages	57
3.8	Encoding-dependent strings	57
4	Changes	61
4.1	Changes in babel version 3.9	61

II	Source code	62
5	Identification and loading of required files	62
6	locale directory	62
7	Tools	63
7.1	Multiple languages	67
7.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	68
7.3	<code>base</code>	69
7.4	<code>key=value</code> options and other general option	69
7.5	Conditional loading of shorthands	71
7.6	Interlude for Plain	73
8	Multiple languages	73
8.1	Selecting the language	76
8.2	Errors	84
8.3	Hooks	86
8.4	Setting up language files	88
8.5	Shorthands	90
8.6	Language attributes	100
8.7	Support for saving macro definitions	102
8.8	Short tags	103
8.9	Hyphens	103
8.10	Multiencoding strings	105
8.11	Macros common to a number of languages	112
8.12	Making glyphs available	112
8.12.1	Quotation marks	112
8.12.2	Letters	113
8.12.3	Shorthands for quotation marks	114
8.12.4	Umlauts and tremas	115
8.13	Layout	116
8.14	Load engine specific macros	117
8.15	Creating and modifying languages	117
9	Adjusting the Babel behavior	138
9.1	Cross referencing macros	141
9.2	Marks	143
9.3	Preventing clashes with other packages	144
9.3.1	<code>ifthen</code>	144
9.3.2	<code>varioref</code>	145
9.3.3	<code>hhline</code>	145
9.4	Encoding and fonts	146
9.5	Basic bidi support	148
9.6	Local Language Configuration	151
9.7	Language options	151
10	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	155
11	Loading hyphenation patterns	155
12	Font handling with <code>fontspec</code>	159

13	Hooks for XeTeX and LuaTeX	164
13.1	XeTeX	164
13.2	Layout	166
13.3	LuaTeX	167
13.4	Southeast Asian scripts	173
13.5	CJK line breaking	175
13.6	Arabic justification	177
13.7	Common stuff	181
13.8	Automatic fonts and ids switching	181
13.9	Bidi	186
13.10	Layout	188
13.11	Lua: transforms	191
13.12	Lua: Auto bidi with basic and basic-r	200
14	Data for CJK	211
15	The ‘nil’ language	211
16	Support for Plain T_EX (plain.def)	211
16.1	Not renaming hyphen.tex	211
16.2	Emulating some L ^A T _E X features	212
16.3	General tools	213
16.4	Encoding related macros	217
17	Acknowledgements	220

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	6
You are loading directly a language style	8
Unknown language ‘LANG’	9
Argument of \language@active@arg” has an extra }	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	28
Package babel Info: The following fonts are not babel standard families	28

Part I

User guide

What is this document about? This user guide focuses on internationalization and localization with \LaTeX and pdf \TeX , xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-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 site](#). 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 lmrroman. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

Examples of cases where `main` is useful are the following.

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

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

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

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

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

To switch the language there are two basic macros, described below in detail:

`\selectlanguage` is used for blocks of text, while `\foreignlanguage` is for chunks of text inside paragraphs.

EXAMPLE A full bilingual document with pdf_{tex} follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDF_{TEX}

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

EXAMPLE With x_{etex} and l_{uatex}, 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 1.22 for further details.

1.3 Mostly monolingual documents

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

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

EXAMPLE A trivial document with the default font in English and Spanish, and FreeSerif in Russian is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

NOTE Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or three-letter word is a valid name for a language (eg, yi). See section 1.22 for further details.

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

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

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

```
! 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 e-Plain and pdf-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 those formats. Please, refer to [Using babel with Plain](#) for further details.

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

NOTE For “historical reasons”, a macro name is converted to a language name without the leading `\`; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a “real” name is deprecated. **New 3.43** However, if the macro name does not match any language, it will get expanded as expected.

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

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.

WARNING `\selectlanguage` should not be used inside some boxed environments (like floats or minipage) to switch the language if you need the information written to the aux be correctly synchronized. This rarely happens, but if it were the case, you must use `other language` instead.

`\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), and since it is meant for phrases only the text direction (and not the paragraph one) is set.

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

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

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

1.8 Auxiliary language selectors

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

The environment `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 `otherlanguage*` does not.

1.9 More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with the option `fontenc`.⁴ A couple of examples:

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

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things; for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionaries and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

⁴With it, encoded strings may not work as expected.

! 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` $\{\langle shorthands-list \rangle\}$
`\shorthandoff` $*\{\langle shorthands-list \rangle\}$

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.

WARNING It is worth emphasizing these macros are meant for temporary changes. Whenever possible and if there are not conflicts with other packages, shorthands must be always enabled (or disabled).

`\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 discretionaries (languages do not define shorthands consistently, and "-", "\-", "=" have different meanings). You can start with, say:

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

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

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

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

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

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

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).⁵ Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

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

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

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

```
\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:

⁵Actually, 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.

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

For your records, here is a list of shorthands, but you must double check them, as they may change.⁶

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.⁷

`\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.

⁶Thanks to Enrico Gregorio

⁷This declaration serves to nothing, but it is preserved for backward compatibility.

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 safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in $\epsilon\TeX$ 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 normal they are deactivated in math mode (default is active) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.

- config=** $\langle file \rangle$
Load $\langle file \rangle$.cfg instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).

- main=** $\langle language \rangle$
Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.

- headfoot=** `<language>`
 By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key config is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** New 3.9l Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** New 3.9l No warnings and no *infos* are written to the log file.⁸
- strings=** `generic` | `unicode` | `encoded` | `<label>` | ``
 Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional \TeX , LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
- hyphenmap=** `off` | `first` | `select` | `other` | `other*`
New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.⁹ 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.¹⁰
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.¹¹
- 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. 1.24.
- layout=** New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.

⁸You can use alternatively the package `silence`.

⁹Turned off in plain.

¹⁰Duplicated options count as several ones.

¹¹Providing `foreign` is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

1.12 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:

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

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

1.13 ini files

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

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

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

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

LUATEX/XETEX

```
\documentclass{book}

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

```

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

\begin{document}

\tableofcontents

\chapter{სამხარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამხარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

```

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

```

Or also:

```

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

```

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

```

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

```

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

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

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

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

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

Southeast scripts Thai works in both `luatex` and `xetex`, but line breaking differs (rules can be modified in `luatex`; they are hard-coded in `xetex`). Lao seems to work, too, but there are no patterns for the latter in `luatex`. 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]{1໐ 1໙ 1໑ 1໘ 1໗} % 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}	bg	Bulgarian ^{ul}
agq	Aghem	bm	Bambara
ak	Akan	bn	Bangla ^{ul}
am	Amharic ^{ul}	bo	Tibetan ^u
ar	Arabic ^{ul}	brx	Bodo
ar-DZ	Arabic ^{ul}	bs-Cyrl	Bosnian
ar-MA	Arabic ^{ul}	bs-Latn	Bosnian ^{ul}
ar-SY	Arabic ^{ul}	bs	Bosnian ^{ul}
as	Assamese	ca	Catalan ^{ul}
asa	Asu	ce	Chechen
ast	Asturian ^{ul}	cgg	Chiga
az-Cyrl	Azerbaijani	chr	Cherokee
az-Latn	Azerbaijani	ckb	Central Kurdish
az	Azerbaijani ^{ul}	cop	Coptic
bas	Basaa	cs	Czech ^{ul}
be	Belarusian ^{ul}	cu	Church Slavic
bem	Bemba	cu-Cyrs	Church Slavic
bez	Bena	cu-Glag	Church Slavic

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

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

zh-Hant-MO	Chinese	zh	Chinese
zh-Hant	Chinese	zu	Zulu

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

aghem	burmese
akan	canadian
albanian	cantonese
american	catalan
amharic	centralatlastamazight
ancientgreek	centralkurdish
arabic	chechen
arabic-algeria	cherokee
arabic-DZ	chiga
arabic-morocco	chinese-hans-hk
arabic-MA	chinese-hans-mo
arabic-syria	chinese-hans-sg
arabic-SY	chinese-hans
armenian	chinese-hant-hk
assamese	chinese-hant-mo
asturian	chinese-hant
asu	chinese-simplified-hongkongsarchina
australian	chinese-simplified-macausarchina
austrian	chinese-simplified-singapore
azerbaijani-cyrillic	chinese-simplified
azerbaijani-cyrl	chinese-traditional-hongkongsarchina
azerbaijani-latin	chinese-traditional-macausarchina
azerbaijani-latn	chinese-traditional
azerbaijani	chinese
bafia	churchslavic
bambara	churchslavic-cyrs
basaa	churchslavic-oldcyrillic ¹²
basque	churchsslavic-glag
belarusian	churchsslavic-glagolitic
bemba	cognian
bena	cornish
bengali	croatian
bodo	czech
bosnian-cyrillic	danish
bosnian-cyrl	duala
bosnian-latin	dutch
bosnian-latn	dzongkha
bosnian	embu
brazilian	english-au
breton	english-australia
british	english-ca
bulgarian	english-canada

¹²The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

english-gb
english-newzealand
english-nz
english-unitedkingdom
english-unitedstates
english-us
english
esperanto
estonian
ewe
ewondo
faroese
filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi

kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali

newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym

sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq
telugu
teso
thai
tibetan
tigrinya
tongan
turkish
turkmen

ukenglish	vai-latn
ukrainian	vai-vai
uppersorbian	vai-vaii
urdu	vai
usenglish	vietnam
usorbian	vietnamese
uyghur	vunjo
uzbek-arab	walser
uzbek-arabic	welsh
uzbek-cyrillic	westernfrisian
uzbek-cyrl	yangben
uzbek-latin	yiddish
uzbek-latn	yoruba
uzbek	zarma
vai-latin	zulu afrikaans

Modifying and adding values to ini files

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

1.14 Selecting fonts

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

`\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.

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

There is a counterpart for code to be run when a language is unselected: \noextras⟨lang⟩.

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

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

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

first loads danish.ldf, and then redefines the captions for danish (as provided by the ini file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some additional tools if provided by the ini file, like extra counters.

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

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

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

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

import= *<language-tag>*

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

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

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

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

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

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

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

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

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

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

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

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

New 3.58 Another special value is unhyphenated, which activates a line breking mode that allows spaces to be stretched to arbitrary amounts.

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}
```

Remerber there is an alternative syntax for the latter:

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

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

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

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

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

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

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

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

Same for `\Alph`.

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

onchar= `ids` | `fonts`

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

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

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

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

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

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

justification= `kashida` | `elongated` | `unhyphenated`

New 3.59 There are currently three options, mainly for the Arabic script. It sets the linebreaking and justification method, which can be based on the the ARABIC TATWEEL character or in the ‘justification alternatives’ OpenType table (`jalt`). For an explanation see the [babel site](#).

linebreaking= **New 3.59** Just a synonymous for justification.

mapfont= `direction`

Assigns the font for the writing direction of this language (only with `bidi=basic`). Whenever possible, instead of this option use `onchar`, based on the script, which usually

makes more sense. More precisely, what `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

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

1.17 Digits and counters

New 3.20 About thirty ini files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With the option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on `\arabic`.)

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

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

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

The styles are:

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

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

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

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

\localeinfo `{\field}`

New 3.38 If an ini file has been loaded for the current language, you may access the information stored in it. This 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 `*{\macro}{\locale}{\property}`

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

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

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

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

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

NOTE To set hyphenation exceptions in the preamble before any language is explicitly set with a selector, use \babelhyphenation instead of \hyphenation. In the preamble the hyphenation rules are not always fully set up and an error can be raised.

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

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

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

New 3.9m *In luatex only*,¹⁴ adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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

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

New 3.31 (Only luatex.) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (**New 3.32** it is disabled in verbatim mode, or more precisely when the

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

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).

1.21 Transforms

Transforms (only luatex) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.¹⁵

It currently embraces `\babelprehyphenation` and `\babelposthyphenation`.

New 3.57 Several ini files predefine some transforms. They are activated with the key `transforms` in `\babelprovide`, either if the locale is being defined with this macro or the languages has been previously loaded as a class or package option, as the following example illustrates:

```
\usepackage[magyar]{babel}
\babelprovide[transforms = digraphs.hyphen]{magyar}
```

Here are the transforms currently predefined. (More to follow in future releases.)

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for dad (simple and T _E X-friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	<code>oneletter.nobreak</code>	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: <i>!?:;</i> .
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zsz</i> as <i>cs-cs, dz-dz</i> , etc.

¹⁵They are similar in concept, but not the same, as those in Unicode.

Indic scripts	danda.nobreak	Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.
Arabic, Persian	kashida.plain	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.
Serbian	transliteration.gajica	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.

\babelposthyphenation $\{\langle hyphenrules-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

New 3.37-3.39 With *luatex* it is 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. A few rules are currently provided (see above), 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 $([\text{t}\acute{u}])$, the replacement could be $\{1|\text{t}\acute{u}|\text{t}\acute{u}\}$, which maps $\text{t}\acute{}$ to $\text{t}\acute{}$, and \acute{u} to \acute{u} , so that the diaeresis is removed.

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

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

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

\babelprehyphenation $\{\langle locale-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

New 3.44-3.52 It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences: (1) the first argument is the locale instead of the name of the hyphenation patterns; (2) in the search patterns = has no special meaning, while | stands for an ordinary space; (3) in the replacement, discretionaries are not accepted. This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`.

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


```

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

```

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

```

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

```

NOTE With luatex there is another approach to make text transformations, with the function `fonts.handlers.otf.addfeature`, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and babel by default recognizes this setting if the font has been declared with `\babelfont`. The *transforms* mechanism supplements rather than replaces OTF features.

With xetex, where *transforms* are not available, there is still another approach, with font mappings, mainly meant to perform encoding conversions and transliterations. Mappings, however, are linked to fonts, not to languages.

1.22 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{
  autoloader.bcp47 = on,
  autoloader.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.

1.23 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

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.

LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

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

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection{<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. It attempts to do the same for `pgf/tikz`. Somewhat experimental. **New 3.32** .

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.25 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).

1.26 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 with a certain `{\name}` 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 T_EX 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).

1.27 Languages supported by babel with ldf files

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

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton
Bulgarian bulgarian
Catalan catalan
Croatian croatian
Czech czech
Danish danish

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

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

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

```

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

```

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

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

1.28 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}
```

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

1.30 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\extrarussian{\inputencoding{koi8-r}}
```

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

1.31 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.

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.

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.

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).

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with `language.dat`

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

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

2.1 Format

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

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

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.

```
% 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.

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are

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

discussed below. You must define all or none for a language (or a dialect); defining, say, `\date⟨lang⟩` but not `\captions⟨lang⟩` does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

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

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

3.1 Guidelines for contributed languages

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

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

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

²⁶But not removed, for backward compatibility.

- 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, ot f, 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://latex3.github.io/babel/guides/list-of-locale-templates.html>.

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

3.2 Basic macros

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

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

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

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

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

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

\providehyphenmins The macro \providehyphenmins should be used in the language definition files to set \lefthyphenmin and \righthyphenmin. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *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>.

<code>\bbl@declare@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \TeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \TeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions{<lang>}</code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

```

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

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

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

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

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

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

\StartBabelCommands*{<language>}{date}

```



```

\SetString\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

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

`\bbl@activate` The command `\bbl@activate` is used to change the way an active character expands.

`\bbl@deactivate` `\bbl@activate` ‘switches on’ the active behavior of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

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

`\bbl@add@special` The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]

`\bbl@remove@special` It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

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

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

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

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

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

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

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

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

3.8 Encoding-dependent strings

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

Furthermore, it must be activated explicitly, with the package option `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (and except if forced as described

²⁷This mechanism was introduced by Bernd Raichle.

below). In other words, the old way of defining/switching strings still works and it's used by default.

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

²⁸In future releases further categories may be added.

```

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

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

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

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


\StartBabelCommands{german,austrian}{date}
\SetString\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}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

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

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

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

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

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

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

\SetString {*<macro-name>*}{*<string>*}

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

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

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

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

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

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would typically be things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A *<map-list>* is a series of macros using the internal format of `\@uclclist` (eg, `\bb\BB\cc\CC`). The mandatory arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \TeX , 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=`I\relax
 \uccode`1=`I\relax}
{\lccode`I=`i\relax
 \lccode`I=`1\relax}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap {*<to-lower-macros>*}

New 3.9g Case mapping serves in \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 through the given uppercase codes, using the step, and assigns them the `lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{⟨uccode-from⟩}{⟨uccode-to⟩}{⟨step⟩}{⟨lccode⟩}` loops through the given uppercase codes, using the step, and assigns them the `lccode`, which is fixed (MO stands for *many-to-one*).

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.63.2517>>
2 <<date=2021/10/06>>
```

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

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

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

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

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

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

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

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

```
29 \def\bbl@exp#1{%
30   \begingroup
31   \let\ \noexpand
32   \let\<\bbl@exp@en
33   \let\[\bbl@exp@ue
34   \edef\bbl@exp@aux{\endgroup#1}%
```

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


```

35 \bbl@exp@aux}
36 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
37 \def\bbl@exp@ue#1]{%
38 \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

`\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.

```

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

```

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

```

52 \begingroup
53 \gdef\bbl@ifunset#1{%
54 \expandafter\ifx\csname#1\endcsname\relax
55 \expandafter\@firstoftwo
56 \else
57 \expandafter\@secondoftwo
58 \fi}
59 \bbl@ifunset{ifcsname}% TODO. A better test?
60 {}%
61 {\gdef\bbl@ifunset#1{%
62 \ifcsname#1\endcsname
63 \expandafter\ifx\csname#1\endcsname\relax
64 \bbl@afterelse\expandafter\@firstoftwo
65 \else
66 \bbl@afterfi\expandafter\@secondoftwo
67 \fi
68 \else
69 \expandafter\@firstoftwo
70 \fi}}
71 \endgroup

```

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

```

72 \def\bbl@ifblank#1{%
73 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
74 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
75 \def\bbl@ifset#1#2#3{%
76 \bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{#1}}{#3}{#2}}}

```

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

```

77 \def\bbl@forkv#1#2{%
78   \def\bbl@kvcmd##1##2##3{#2}%
79   \bbl@kvnext#1,\@nil,}
80 \def\bbl@kvnext#1,{%
81   \ifx\@nil#1\relax\else
82     \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
83     \expandafter\bbl@kvnext
84   \fi}
85 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
86   \bbl@trim@def\bbl@forkv@a{#1}%
87   \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).

```

88 \def\bbl@vforeach#1#2{%
89   \def\bbl@forcmd##1{#2}%
90   \bbl@fornext#1,\@nil,}
91 \def\bbl@fornext#1,{%
92   \ifx\@nil#1\relax\else
93     \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
94     \expandafter\bbl@fornext
95   \fi}
96 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

\bbl@replace Returns implicitly \toks@ with the modified string.

```

97 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
98   \toks@{}%
99   \def\bbl@replace@aux##1#2##2#2{%
100     \ifx\bbl@nil##2%
101       \toks@\expandafter{\the\toks@##1}%
102     \else
103       \toks@\expandafter{\the\toks@##1#3}%
104       \bbl@afterfi
105       \bbl@replace@aux##2#2%
106     \fi}%
107   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
108   \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).

```

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

```

```

125         \\makeatletter % "internal" macros with @ are assumed
126         \\scantokens{%
127             \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
128             \catcode64=\the\catcode64\relax}% Restore @
129     \else
130         \let\bbl@tempc\@empty % Not \relax
131     \fi
132     \bbl@exp{%      For the 'uplevel' assignments
133 \endgroup
134     \bbl@tempc}} % empty or expand to set #1 with changes
135 \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 pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter in your language style if you want.

```

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

```

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

```

158 \def\bbl@bsphack{%
159 \ifhmode
160     \hskip\z@skip
161     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
162 \else
163     \let\bbl@esphack\@empty
164 \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`.

```

165 \def\bbl@cased{%
166 \ifx\oe\OE
167     \expandafter\in@\expandafter
168     {\expandafter\OE\expandafter}\expandafter{\oe}%
169 \ifin@
170     \bbl@afterelse\expandafter\MakeUppercase
171 \else
172     \bbl@afterfi\expandafter\MakeLowercase
173 \fi
174 \else

```

```

175 \expandafter\@firstofone
176 \fi}

```

An alternative to `\IfFormatAtLeastTF` for old versions. Temporary.

```

177 \ifx\IfFormatAtLeastTF\undefined
178 \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
179 \else
180 \let\bbl@ifformatlater\IfFormatAtLeastTF
181 \fi

```

The following adds some code to `\extras...` both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with `#`'s. Used to deal with `alph`, `Alph` and `frenchspacing` when there are already changes (with `\babel@save`).

```

182 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
183 \toks@\expandafter\expandafter\expandafter{%
184 \csname extras\language\endcsname}%
185 \bbl@exp{\in@{#1}{\the\toks@}}%
186 \ifin@ \else
187 \@temptokena{#2}%
188 \edef\bbl@tempc{\the\@temptokena\the\toks@}%
189 \toks@\expandafter{\bbl@tempc#3}%
190 \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
191 \fi}
192 <</Basic macros>>

```

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

```

193 <<*Make sure ProvidesFile is defined>> ≡
194 \ifx\ProvidesFile\undefined
195 \def\ProvidesFile#1[#2 #3 #4]{%
196 \wlog{File: #1 #4 #3 <#2>}%
197 \let\ProvidesFile\undefined}
198 \fi
199 <</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.

```

200 <<*Define core switching macros>> ≡
201 \ifx\language\undefined
202 \csname newcount\endcsname\language
203 \fi
204 <</Define core switching macros>>

```

`\last@language` Another counter is used to keep track of the allocated languages. \TeX and \LaTeX reserves for this purpose the count 19.

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

```

205 <<*Define core switching macros>> ≡
206 \countdef\last@language=19
207 \def\addlanguage{\csname newlanguage\endcsname}
208 <</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. 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 (~~La~~T_EX, `babel.sty`)

```

209 <*package>
210 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
211 \ProvidesPackage{babel}[\<<date>> \<<version>>] The Babel package]

Start with some “private” debugging tool, and then define macros for errors.
212 \@ifpackagewith{babel}{debug}
213   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
214    \let\bbl@debug\@firstofone
215    \ifx\directlua\@undefined\else
216      \directlua{ Babel = Babel or {}
217        Babel.debug = true }%
218      \input{babel-debug.tex}%
219    \fi}
220 {\providecommand\bbl@trace[1]{}%
221  \let\bbl@debug\@gobble
222  \ifx\directlua\@undefined\else
223    \directlua{ Babel = Babel or {}
224      Babel.debug = false }%
225  \fi}
226 \def\bbl@error#1#2{%
227   \begingroup
228     \def\{\MessageBreak}%
229     \PackageError{babel}{#1}{#2}%
230   \endgroup}
231 \def\bbl@warning#1{%
232   \begingroup
233     \def\{\MessageBreak}%
234     \PackageWarning{babel}{#1}%
235   \endgroup}
236 \def\bbl@infowarn#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \GenericWarning
240       {(babel) \spaces\@spaces\@spaces}%
241       {Package babel Info: #1}%
242   \endgroup}
243 \def\bbl@info#1{%
244   \begingroup
245     \def\{\MessageBreak}%
246     \PackageInfo{babel}{#1}%
247   \endgroup}

```

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.

But first, include here the *Basic macros* defined above.

```

248 <<Basic macros>>
249 \@ifpackagewith{babel}{silent}
250   {\let\bbl@info\@gobble
251    \let\bbl@infowarn\@gobble

```

```

252 \let\bbl@warning\@gobble}
253 {}
254 %
255 \def\AfterBabelLanguage#1{%
256 \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.
257 \ifx\bbl@languages\@undefined\else
258 \begingroup
259 \catcode\^^I=12
260 \@ifpackagewith{babel}{showlanguages}{%
261 \begingroup
262 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
263 \wlog{<*languages>}%
264 \bbl@languages
265 \wlog{</languages>}%
266 \endgroup}{%
267 \endgroup
268 \def\bbl@elt#1#2#3#4{%
269 \ifnum#2=\z@
270 \gdef\bbl@nulllanguage{#1}%
271 \def\bbl@elt##1##2##3##4{}}%
272 \fi}%
273 \bbl@languages
274 \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`.

```

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

```

7.4 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no

modifiers have been given, the former is `\relax`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\for` or load `keyval`, for example.

```

295 \bbl@trace{key=value and another general options}
296 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
297 \def\bbl@tempb#1.#2{% Remove trailing dot
298   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
299 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
300   \ifx\@empty#2%
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302   \else
303     \in@{,provide=}{, #1}%
304     \ifin@
305       \edef\bbl@tempc{%
306         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
307     \else
308       \in@{=}{#1}%
309       \ifin@
310         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
311       \else
312         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
313         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
314       \fi
315     \fi
316   \fi}
317 \let\bbl@tempc\@empty
318 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
319 \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.

```

320 \DeclareOption{KeepShorthandsActive}{}
321 \DeclareOption{activeacute}{}
322 \DeclareOption{activegrave}{}
323 \DeclareOption{debug}{}
324 \DeclareOption{noconfigs}{}
325 \DeclareOption{showlanguages}{}
326 \DeclareOption{silent}{}
327 % \DeclareOption{mono}{}
328 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
329 \chardef\bbl@iniflag\z@
330 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
331 \DeclareOption{provide=+}{\chardef\bbl@iniflag\tw@} % add = 2
332 \DeclareOption{provide=*+}{\chardef\bbl@iniflag\thr@@} % add + main
333 % A separate option
334 \let\bbl@autoload@options\@empty
335 \DeclareOption{provide=@}{\def\bbl@autoload@options{import}}
336 % Don't use. Experimental. TODO.
337 \newif\ifbbl@single
338 \DeclareOption{selectors=off}{\bbl@singletrue}
339 <<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.

```

340 \let\bbl@opt@shorthands\@nnil
341 \let\bbl@opt@config\@nnil
342 \let\bbl@opt@main\@nnil

```

```

343 \let\bbl@opt@headfoot\@nnil
344 \let\bbl@opt@layout\@nnil
345 \let\bbl@opt@provide\@nnil

```

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

```

346 \def\bbl@tempa#1=#2\bbl@tempa{%
347   \bbl@csarg\ifx{opt@#1}\@nnil
348     \bbl@csarg\edef{opt@#1}{#2}%
349   \else
350     \bbl@error
351     {Bad option '#1=#2'. Either you have misspelled the\\%
352       key or there is a previous setting of '#1'. Valid\\%
353       keys are, among others, 'shorthands', 'main', 'bidi',\\%
354       'strings', 'config', 'headfoot', 'safe', 'math'.}%
355     {See the manual for further details.}
356   \fi}

```

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

```

357 \let\bbl@language@opts\@empty
358 \DeclareOption*{%
359   \bbl@xin@{\string=}{\CurrentOption}%
360   \ifin@
361     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
362   \else
363     \bbl@xin@{,\CurrentOption,}{,\bbl@language@opts,%
364     \ifin@
365       \bbl@exp{\\bbl@replace\\bbl@language@opts{,\CurrentOption,}{}}%
366     \fi
367     \edef\bbl@language@opts{\bbl@language@opts,\CurrentOption,}
368   \fi}

```

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

```

369 \ProcessOptions*
370 \ifx\bbl@opt@provide\@nnil
371   \let\bbl@opt@provide\@empty %%%% MOVE above
372 \else
373   \chardef\bbl@iniflag\@ne
374   \bbl@exp{\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
375     \in@{,provide,}{,#1,}%
376     \ifin@
377       \def\bbl@opt@provide{#2}%
378       \bbl@replace\bbl@opt@provide{;}{,}%
379     \fi}
380 \fi
381 %

```

7.5 Conditional loading of shorthands

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

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

```

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

```



```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

416 \ifx\bbbl@opt@safe\@undefined
417 \def\bbbl@opt@safe{BR}
418 \fi

```

Make sure the language set with ‘main’ is the last one.

```

419 \ifx\bbbl@opt@main\@nnil\else
420 \edef\bbbl@language@opts{\bbbl@language@opts,\bbbl@opt@main,}
421 \fi

```

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

```

422 \bbbl@trace{Defining IfBabelLayout}
423 \ifx\bbbl@opt@layout\@nnil
424 \newcommand\IfBabelLayout[3]{#3}%
425 \else
426 \newcommand\IfBabelLayout[1]{%

```

```

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

```

7.6 Interlude for Plain

Because of the way docstrip works, we need to insert some code for Plain here. However, the tools provided by the babel installer for literate programming makes this section a short interlude, because the actual code is below, tagged as *Emulate LaTeX*.

```

436 \ifx\ldf@quit\undefined\else
437 \endinput\fi % Same line!
438 <<Make sure ProvidesFile is defined>>
439 \ProvidesFile{babel.def}[\<date>] \<version>] Babel common definitions]
440 \ifx\AtBeginDocument\undefined % TODO. change test.
441 <<Emulate LaTeX>>
442 \fi

```

That is all for the moment. Now follows some common stuff, for both Plain and \LaTeX . After it, we will resume the \LaTeX -only stuff.

```

443 \endcore
444 \package | core

```

8 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.

```

445 \def\bbl@version{\<version>}
446 \def\bbl@date{\<date>}
447 <<Define core switching macros>>

```

`\adddialect` The macro `\adddialect` can be used to add the name of a dialect or variant language, for which an already defined hyphenation table can be used.

```

448 \def\adddialect#1#2{%
449 \global\chardef#1#2\relax
450 \bbl@usehooks{adddialect}{\#1}{\#2}}%
451 \begingroup
452 \count@#1\relax
453 \def\bbl@elt##1##2##3##4{%
454 \ifnum\count@=##2\relax
455 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
456 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
457 set to \expandafter\string\csname l@##1\endcsname\\%
458 (\string\language\the\count@). Reported}%
459 \def\bbl@elt####1####2####3####4}%
460 \fi}%
461 \bbl@cs{languages}%
462 \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 an attempt 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.

```

463 \def\bbl@fixname#1{%
464   \begingroup
465   \def\bbl@tempe{l@}%
466   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
467   \bbl@tempd
468   {\lowercase\expandafter{\bbl@tempd}%
469    \uppercase\expandafter{\bbl@tempd}%
470    \@empty
471    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
472     \uppercase\expandafter{\bbl@tempd}}}%
473    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
474     \lowercase\expandafter{\bbl@tempd}}}%
475   \@empty
476   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
477   \bbl@tempd
478   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
479 \def\bbl@iflanguage#1{%
480   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

481 \def\bbl@bcpcase#1#2#3#4\@#5{%
482   \ifx\@empty#3%
483     \uppercase{\def#5{#1#2}}%
484   \else
485     \uppercase{\def#5{#1}}%
486     \lowercase{\edef#5{#5#2#3#4}}%
487   \fi}
488 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
489   \let\bbl@bcp\relax
490   \lowercase{\def\bbl@tempa{#1}}%
491   \ifx\@empty#2%
492     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
493   \else\ifx\@empty#3%
494     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
495     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
496     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
497     }%
498     \ifx\bbl@bcp\relax
499       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
500     \fi
501   \else
502     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
503     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
504     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
505     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
506     }%
507     \ifx\bbl@bcp\relax
508       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
509       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
510     }%

```

```

511 \fi
512 \ifx\bb1@bcp\relax
513 \IfFileExists{babel-\bb1@tempa-\bb1@tempc.ini}%
514 {\edef\bb1@bcp{\bb1@tempa-\bb1@tempc}}%
515 {}%
516 \fi
517 \ifx\bb1@bcp\relax
518 \IfFileExists{babel-\bb1@tempa.ini}{\let\bb1@bcp\bb1@tempa}{}%
519 \fi
520 \fi\fi}
521 \let\bb1@initoload\relax
522 \def\bb1@provide@locale{%
523 \ifx\babelprovide\@undefined
524 \bb1@error{For a language to be defined on the fly 'base'\\%
525 is not enough, and the whole package must be\\%
526 loaded. Either delete the 'base' option or\\%
527 request the languages explicitly}%
528 {See the manual for further details.}%
529 \fi
530 % TODO. Option to search if loaded, with \LocaleForEach
531 \let\bb1@auxname\language % Still necessary. TODO
532 \bb1@ifunset{\bb1@bcp@map@\language}{}% Move uplevel??
533 {\edef\language{\@nameuse{\bb1@bcp@map@\language}}}%
534 \ifbb1@bcpallowed
535 \expandafter\ifx\csname date\language\endcsname\relax
536 \expandafter
537 \bb1@bcplookup\language-\@empty-\@empty-\@empty\@@
538 \ifx\bb1@bcp\relax\else % Returned by \bb1@bcplookup
539 \edef\language{\bb1@bcp@prefix\bb1@bcp}%
540 \edef\localename{\bb1@bcp@prefix\bb1@bcp}%
541 \expandafter\ifx\csname date\language\endcsname\relax
542 \let\bb1@initoload\bb1@bcp
543 \bb1@exp{\@babelprovide[\bb1@autoload@bcptoptions]{\language}}%
544 \let\bb1@initoload\relax
545 \fi
546 \bb1@csarg\xdef{bcp@map@\bb1@bcp}{\localename}%
547 \fi
548 \fi
549 \fi
550 \expandafter\ifx\csname date\language\endcsname\relax
551 \IfFileExists{babel-\language.tex}%
552 {\bb1@exp{\@babelprovide[\bb1@autoload@options]{\language}}}%
553 {}%
554 \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.

```

555 \def\iflanguage#1{%
556 \bb1@iflanguage{#1}{%
557 \ifnum\csname l@#1\endcsname=\language
558 \expandafter\@firstoftwo
559 \else
560 \expandafter\@secondoftwo
561 \fi}}

```

8.1 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.

```
562 \let\bbl@select@type\z@
563 \edef\selectlanguage{%
564   \noexpand\protect
565   \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`.

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

The following definition is preserved for backwards compatibility (eg, arabi, koma). It is related to a trick for 2.09, now discarded.

```
567 \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.

```
568 \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`

```
569 \def\bbl@push@language{%
570   \ifx\language\@undefined\else
571     \ifx\currentgrouplevel\@undefined
572       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
573     \else
574       \ifnum\currentgrouplevel=\z@
575         \xdef\bbl@language@stack{\language+}%
576       \else
577         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
578       \fi
579     \fi
580 \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`.

```
581 \def\bbl@pop@lang#1+#2\@{%
582   \edef\language{#1}%
583   \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).

```
584 \let\bbl@ifrestoring\@secondoftwo
585 \def\bbl@pop@language{%
586   \expandafter\bbl@pop@lang\bbl@language@stack\@@
587   \let\bbl@ifrestoring\@firstoftwo
588   \expandafter\bbl@set@language\expandafter{\language}%
589   \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).

```
590 \chardef\localeid\z@
591 \def\bbl@id@last{0} % No real need for a new counter
592 \def\bbl@id@assign{%
593   \bbl@ifunset{\bbl@id@\language}%
594   {\count@\bbl@id@last\relax
595     \advance\count@\@ne
596     \bbl@csarg\chardef{id@\language}\count@
597     \edef\bbl@id@last{\the\count@}%
598     \ifcase\bbl@engine\or
599       \directlua{
600         Babel = Babel or {}
601         Babel.locale_props = Babel.locale_props or {}
602         Babel.locale_props[\bbl@id@last] = {}
603         Babel.locale_props[\bbl@id@last].name = '\language'
604       }%
605     \fi}%
606   }%
607   \chardef\localeid\bbl@ccl{id@}}
```

The unprotected part of `\selectlanguage`.

```
608 \expandafter\def\csname selectlanguage \endcsname#1{%
609   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
610   \bbl@push@language
611   \aftergroup\bbl@pop@language
612   \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.

`\bbl@savelastskip` is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from `hyperref`, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in `luatex`, is to avoid the `\write` altogether when not needed).

```
613 \def\BabelContentsFiles{toc,lof,lot}
614 \def\bbl@set@language#1{% from selectlanguage, pop@
```

```

615 % The old buggy way. Preserved for compatibility.
616 \edef\language{%
617   \ifnum\escapechar=\expandafter`\string#1\@empty
618   \else\string#1\@empty\fi}%
619 \ifcat\relax\noexpand#1%
620   \expandafter\ifx\csname date\language\endcsname\relax
621     \edef\language{#1}%
622     \let\locale\language
623   \else
624     \bbl@info{Using '\string\language' instead of 'language' is\\%
625               deprecated. If what you want is to use a\\%
626               macro containing the actual locale, make\\%
627               sure it does not not match any language.\\%
628               Reported}%
629     \ifx\scantokens\@undefined
630       \def\locale{??}%
631     \else
632       \scantokens\expandafter{\expandafter
633         \def\expandafter\locale\expandafter{\language}}%
634     \fi
635   \fi
636 \else
637   \def\locale{#1}% This one has the correct catcodes
638 \fi
639 \select@language{\language}%
640 % write to aux
641 \expandafter\ifx\csname date\language\endcsname\relax\else
642   \if@files
643     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
644       \bbl@savelastskip
645       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
646       \bbl@restorelastskip
647     \fi
648     \bbl@usehooks{write}{}%
649   \fi
650 \fi}
651 %
652 \let\bbl@restorelastskip\relax
653 \let\bbl@savelastskip\relax
654 %
655 \newif\ifbbl@bcpallowed
656 \bbl@bcpallowedfalse
657 \def\select@language#1{% from set@, babel@aux
658   % set hymap
659   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
660   % set name
661   \edef\language{#1}%
662   \bbl@fixname\language
663   % TODO. name@map must be here?
664   \bbl@provide@locale
665   \bbl@iflanguage\language{%
666     \expandafter\ifx\csname date\language\endcsname\relax
667       \bbl@error
668       {Unknown language '\language'. Either you have\\%
669        misspelled its name, it has not been installed,\\%
670        or you requested it in a previous run. Fix its name,\\%
671        install it or just rerun the file, respectively. In\\%
672        some cases, you may need to remove the aux file}%
673       {You may proceed, but expect wrong results}%

```

```

674 \else
675 % set type
676 \let\bbl@select@type\z@
677 \expandafter\bbl@switch\expandafter{\language}%
678 \fi}}
679 \def\babel@aux#1#2{%
680 \select@language{#1}%
681 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
682 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
683 \def\babel@toc#1#2{%
684 \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.

```

685 \newif\ifbbl@usedatagroup
686 \def\bbl@switch#1{% from select@, foreign@
687 % make sure there is info for the language if so requested
688 \bbl@ensureinfo{#1}%
689 % restore
690 \originalTeX
691 \expandafter\def\expandafter\originalTeX\expandafter{%
692 \csname noextras#1\endcsname
693 \let\originalTeX\@empty
694 \babel@beginsave}%
695 \bbl@usehooks{afterreset}}}%
696 \languageshorthands{none}%
697 % set the locale id
698 \bbl@id@assign
699 % switch captions, date
700 % No text is supposed to be added here, so we remove any
701 % spurious spaces.
702 \bbl@bsphack
703 \ifcase\bbl@select@type
704 \csname captions#1\endcsname\relax
705 \csname date#1\endcsname\relax
706 \else
707 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
708 \ifin@
709 \csname captions#1\endcsname\relax
710 \fi
711 \bbl@xin@{,date,}{, \bbl@select@opts,}%
712 \ifin@ % if \foreign... within \<lang>date
713 \csname date#1\endcsname\relax
714 \fi
715 \fi
716 \bbl@esphack
717 % switch extras
718 \bbl@usehooks{beforeextras}}}%
719 \csname extras#1\endcsname\relax

```



```

720 \bbl@usehooks{afterextras}{}%
721 % > babel-ensure
722 % > babel-sh-<short>
723 % > babel-bidi
724 % > babel-fontspec
725 % hyphenation - case mapping
726 \ifcase\bbl@opt@hyphenmap\or
727   \def\BabelLower##1##2{\lccode##1=##2\relax}%
728   \ifnum\bbl@hymapsel>4\else
729     \csname\language\language @bbl@hyphenmap\endcsname
730     \fi
731     \chardef\bbl@opt@hyphenmap\z@
732   \else
733     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
734       \csname\language\language @bbl@hyphenmap\endcsname
735       \fi
736   \fi
737   \let\bbl@hymapsel\@cclv
738 % hyphenation - select rules
739 \ifnum\csname l@\language\endcsname=\l@unhyphenated
740   \edef\bbl@tempa{u}%
741 \else
742   \edef\bbl@tempa{\bbl@c1{lnbrk}}%
743 \fi
744 % linebreaking - handle u, e, k (v in the future)
745 \bbl@xin@{/u}{/\bbl@tempa}%
746 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
747 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
748 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
749 \ifin@
750   % unhyphenated/kashida/elongated = allow stretching
751   \language\l@unhyphenated
752   \babel@savevariable\emergencystretch
753   \emergencystretch\maxdimen
754   \babel@savevariable\hbadness
755   \hbadness\@M
756 \else
757   % other = select patterns
758   \bbl@patterns{#1}%
759 \fi
760 % hyphenation - mins
761 \babel@savevariable\lefthyphenmin
762 \babel@savevariable\righthyphenmin
763 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
764   \set@hyphenmins\tw@\thr@\relax
765 \else
766   \expandafter\expandafter\expandafter\set@hyphenmins
767   \csname #1hyphenmins\endcsname\relax
768 \fi}

```

otherlanguage The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.

```

769 \long\def\otherlanguage#1{%
770   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
771   \csname selectlanguage \endcsname{#1}%

```

```
772 \ignorespaces}
```

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

```
773 \long\def\endotherlanguage{%
774 \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`.

```
775 \expandafter\def\csname otherlanguage*\endcsname{%
776 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
777 \def\bbl@otherlanguage@s[#1]#2{%
778 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
779 \def\bbl@select@opts{#1}%
780 \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”.

```
781 \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.

```
782 \providecommand\bbl@beforeforeign{}
783 \edef\foreignlanguage{%
784 \noexpand\protect
785 \expandafter\noexpand\csname foreignlanguage \endcsname}
786 \expandafter\def\csname foreignlanguage \endcsname{%
787 \@ifstar\bbl@foreign@s\bbl@foreign@x}
788 \providecommand\bbl@foreign@x[3][]{%
789 \begingroup
790 \def\bbl@select@opts{#1}%
791 \let\BabelText\@firstofone
792 \bbl@beforeforeign
793 \foreign@language{#2}%
794 \bbl@usehooks{foreign}{}}%
795 \BabelText{#3}% Now in horizontal mode!
796 \endgroup}
797 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
798 \begingroup
```

```

799     {\par}%
800     \let\bbl@select@opts\@empty
801     \let\BabelText\@firstofone
802     \foreign@language{#1}%
803     \bbl@usehooks{foreign*}{}%
804     \bbl@dirparastext
805     \BabelText{#2}% Still in vertical mode!
806     {\par}%
807 \endgroup}

```

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

```

808 \def\foreign@language#1{%
809   % set name
810   \edef\language{#1}%
811   \ifbbl@usedategroup
812     \bbl@add\bbl@select@opts{,date,}%
813     \bbl@usedategroupfalse
814   \fi
815   \bbl@fixname\language
816   % TODO. name@map here?
817   \bbl@provide@locale
818   \bbl@iflanguage\language{%
819     \expandafter\ifx\csname date\language\endcsname\relax
820       \bbl@warning % TODO - why a warning, not an error?
821       {Unknown language '#1'. Either you have\\%
822        misspelled its name, it has not been installed,\\%
823        or you requested it in a previous run. Fix its name,\\%
824        install it or just rerun the file, respectively. In\\%
825        some cases, you may need to remove the aux file.\\%
826        I'll proceed, but expect wrong results.\\%
827        Reported}%
828     \fi
829     % set type
830     \let\bbl@select@type\@ne
831     \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.

```

832 \let\bbl@hyphlist\@empty
833 \let\bbl@hyphenation@relax
834 \let\bbl@pttnlist\@empty
835 \let\bbl@patterns@relax
836 \let\bbl@hymapsel=\ccclv
837 \def\bbl@patterns#1{%
838   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
839     \csname l@#1\endcsname
840     \edef\bbl@tempa{#1}%
841   \else
842     \csname l@#1:\f@encoding\endcsname
843     \edef\bbl@tempa{#1:\f@encoding}%

```

```

844 \fi
845 \@expandtwoargs\bb1@usehooks{patterns}{\#1}{\bb1@tempa}}%
846 % > luatex
847 \@ifundefined{bb1@hyphenation@}{\% Can be \relax!
848 \begingroup
849 \bb1@xin@{\, \number\language,}{\, \bb1@hyphlist}%
850 \ifin@ \else
851 \@expandtwoargs\bb1@usehooks{hyphenation}{\#1}{\bb1@tempa}}%
852 \hyphenation{%
853 \bb1@hyphenation@
854 \@ifundefined{bb1@hyphenation@#1}%
855 \@empty
856 {\space\csname bb1@hyphenation@#1\endcsname}}%
857 \xdef\bb1@hyphlist{\bb1@hyphlist\number\language,}%
858 \fi
859 \endgroup}}

```

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

```

860 \def\hyphenrules#1{%
861 \edef\bb1@tempf{\#1}%
862 \bb1@fixname\bb1@tempf
863 \bb1@iflanguage\bb1@tempf{%
864 \expandafter\bb1@patterns\expandafter{\bb1@tempf}%
865 \ifx\languageshorthands\@undefined\else
866 \languageshorthands{none}%
867 \fi
868 \expandafter\ifx\csname\bb1@tempf hyphenmins\endcsname\relax
869 \set@hyphenmins\tw@\thr@@\relax
870 \else
871 \expandafter\expandafter\expandafter\set@hyphenmins
872 \csname\bb1@tempf hyphenmins\endcsname\relax
873 \fi}}
874 \let\endhyphenrules\@empty

```

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

```

875 \def\providehyphenmins#1#2{%
876 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
877 \@namedef{\#1hyphenmins}{\#2}%
878 \fi}

```

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

```

879 \def\set@hyphenmins#1#2{%
880 \lefthyphenmin#1\relax
881 \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.

```

882 \ifx\ProvidesFile\@undefined
883 \def\ProvidesLanguage#1[#2 #3 #4]{%
884 \wlog{Language: #1 #4 #3 <#2>}%

```

```

885     }
886 \else
887   \def\ProvidesLanguage#1{%
888     \begingroup
889     \catcode`\ 10 %
890     \@makeother\/%
891     \@ifnextchar[%]
892       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
893 \def\@provideslanguage#1[#2]{%
894   \wlog{Language: #1 #2}%
895   \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
896   \endgroup}
897 \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`.

```
898 \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.

```
899 \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’:

```

900 \providecommand\setlocale{%
901   \bbl@error
902   {Not yet available}%
903   {Find an armchair, sit down and wait}}
904 \let\uselocale\setlocale
905 \let\locale\setlocale
906 \let\selectlocale\setlocale
907 \let\localename\setlocale
908 \let\textlocale\setlocale
909 \let\textlanguage\setlocale
910 \let\languagetext\setlocale

```

8.2 Errors

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

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

```

911 \edef\bbl@nulllanguage{\string\language=0}
912 \def\bbl@nocaption{\protect\bbl@nocaption@i}
913 \def\bbl@nocaption@i#1#2% 1: text to be printed 2: caption macro \langXname
914   \global\@namedef{#2}{\textbf{?#1?}}%
915   \@nameuse{#2}%
916 \edef\bbl@tempa{#1}%
917 \bbl@sreplace\bbl@tempa{name}{}%
918 \bbl@warning{% TODO.
919   \@backslashchar#1 not set for '\language'. Please,\\%
920   define it after the language has been loaded\\%

```

```

921 (typically in the preamble) with:\%
922 \string\setlocalecaption{\language}\bbl@tempa}{..\%
923 Reported}}
924 \def\bbl@tentative{\protect\bbl@tentative@i}
925 \def\bbl@tentative@i#1{%
926 \bbl@warning{%
927 Some functions for '#1' are tentative.\%
928 They might not work as expected and their behavior\%
929 could change in the future.\%
930 Reported}}
931 \def\@nolanerr#1{%
932 \bbl@error
933 {You haven't defined the language '#1' yet.\%
934 Perhaps you misspelled it or your installation\%
935 is not complete}%
936 {Your command will be ignored, type <return> to proceed}}
937 \def\@nopatterns#1{%
938 \bbl@warning
939 {No hyphenation patterns were preloaded for\%
940 the language '#1' into the format.\%
941 Please, configure your TeX system to add them and\%
942 rebuild the format. Now I will use the patterns\%
943 preloaded for \bbl@nulllanguage\space instead}}
944 \let\bbl@usehooks\@gobbletwo
945 \ifx\bbl@onlyswitch\@empty\endinput\fi
946 % Here ended switch.def

```

Here ended the now discarded switch.def. Here also (currently) ends the base option.

```

947 \ifx\directlua\@undefined\else
948 \ifx\bbl@luapatterns\@undefined
949 \input luababel.def
950 \fi
951 \fi
952 <<Basic macros>>
953 \bbl@trace{Compatibility with language.def}
954 \ifx\bbl@languages\@undefined
955 \ifx\directlua\@undefined
956 \openin1 = language.def % TODO. Remove hardcoded number
957 \ifeof1
958 \closein1
959 \message{I couldn't find the file language.def}
960 \else
961 \closein1
962 \begingroup
963 \def\addlanguage#1#2#3#4#5{%
964 \expandafter\ifx\csname lang@#1\endcsname\relax\else
965 \global\expandafter\let\csname l@#1\endcsname\expandafter\endcsname
966 \csname lang@#1\endcsname
967 \fi}%
968 \def\uselanguage#1{%
969 \input language.def
970 \endgroup
971 \fi
972 \fi
973 \chardef\l@english\z@
974 \fi

```

\addto It takes two arguments, a *<control sequence>* and T_EX-code to be added to the *<control sequence>*.
If the *<control sequence>* has not been defined before it is defined now. The control sequence could

also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

975 \def\addto#1#2{%
976   \ifx#1\@undefined
977     \def#1{#2}%
978   \else
979     \ifx#1\relax
980       \def#1{#2}%
981     \else
982       {\toks@\expandafter{#1#2}%
983        \xdef#1{\the\toks@}}%
984   \fi
985 \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?

```

986 \def\bbl@withactive#1#2{%
987   \begingroup
988   \lccode`~=#2\relax
989   \lowercase{\endgroup#1~}}

```

`\bbl@redefine` To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don’t want to redefine the \TeX macros completely in case their definitions change (they have changed in the past). A macro named `\macro` will be saved new control sequences named `\org@macro`.

```

990 \def\bbl@redefine#1{%
991   \edef\bbl@tempa{\bbl@stripslash#1}%
992   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
993   \expandafter\def\csname\bbl@tempa\endcsname{
994   \@onlypreamble\bbl@redefine

```

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

```

995 \def\bbl@redefine@long#1{%
996   \edef\bbl@tempa{\bbl@stripslash#1}%
997   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
998   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
999   \@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_`.

```

1000 \def\bbl@redefineroobust#1{%
1001   \edef\bbl@tempa{\bbl@stripslash#1}%
1002   \bbl@ifunset{\bbl@tempa\space}%
1003   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1004    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1005   {\bbl@exp{\let\<org@\bbl@tempa\>\<\bbl@tempa\space>}}}%
1006   \@namedef{\bbl@tempa\space}}
1007 \@onlypreamble\bbl@redefineroobust

```

8.3 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.

```

1008 \bbl@trace{Hooks}
1009 \newcommand\AddBabelHook[3][{}%
1010   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1011   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1012   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1013   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1014     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1015     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1016   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1017 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1018 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1019 \def\bbl@usehooks#1#2{%
1020   \ifx\UseHook\undefined\else\UseHook{babel/#1}\fi
1021   \def\bbl@elth##1{%
1022     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1023     \bbl@cs{ev@#1@}%
1024     \ifx\language\undefined\else % Test required for Plain (?)
1025       \ifx\UseHook\undefined\else\UseHook{babel/\language/#1}\fi
1026       \def\bbl@elth##1{%
1027         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1028         \bbl@cl{ev@#1}%
1029       \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).

```

1030 \def\bbl@evargs{,% <- don't delete this comma
1031   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1032   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1033   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1034   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1035   beforestart=0,language=2}
1036 \ifx\NewHook\undefined\else
1037   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1038   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1039 \fi

```

`\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.

```

1040 \bbl@trace{Defining babelensure}
1041 \newcommand\babelensure[2][{}% TODO - revise test files
1042   \AddBabelHook{babel-ensure}{afterextras}{%
1043     \ifcase\bbl@select@type
1044       \bbl@cl{e}%
1045     \fi}%
1046   \beginingroup
1047     \let\bbl@ens@include\@empty
1048     \let\bbl@ens@exclude\@empty
1049     \def\bbl@ens@fontenc{\relax}%
1050     \def\bbl@tempb##1{%
1051       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1052     \edef\bbl@tempa{\bbl@tempb#1\@empty}%

```



```

1053 \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1054 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1055 \def\bbl@tempc{\bbl@ensure}%
1056 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1057 \expandafter{\bbl@ens@include}}%
1058 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1059 \expandafter{\bbl@ens@exclude}}%
1060 \toks@\expandafter{\bbl@tempc}%
1061 \bbl@exp{%
1062 \endgroup
1063 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1064 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1065 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1066 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1067 \edef##1{\noexpand\bbl@nocaption
1068 {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1069 \fi
1070 \ifx##1\@empty\else
1071 \in@{##1}{#2}%
1072 \ifin@ \else
1073 \bbl@ifunset{bbl@ensure@\language}%
1074 {\bbl@exp{%
1075 \\\DeclareRobustCommand<bbl@ensure>[1]{%
1076 \\\foreignlanguage{\language}%
1077 {\ifx\relax#3\else
1078 \\\fontencoding{#3}\selectfont
1079 \fi
1080 #####1}}}%
1081 {}}%
1082 \toks@\expandafter{##1}%
1083 \edef##1{%
1084 \bbl@csarg\noexpand{ensure@\language}%
1085 {\the\toks@}}%
1086 \fi
1087 \expandafter\bbl@tempb
1088 \fi}%
1089 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1090 \def\bbl@tempa##1{% elt for include list
1091 \ifx##1\@empty\else
1092 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1093 \ifin@ \else
1094 \bbl@tempb##1\@empty
1095 \fi
1096 \expandafter\bbl@tempa
1097 \fi}%
1098 \bbl@tempa#1\@empty}
1099 \def\bbl@captionslist{%
1100 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1101 \contentsname\listfigurename\listtablename\indexname\figurename
1102 \tablename\partname\encname\ccname\headtoname\pagename\seename
1103 \alsoname\proofname\glossaryname}

```

8.4 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`.

```

1104 \bbl@trace{Macros for setting language files up}
1105 \def\bbl@ldfinit{%
1106   \let\bbl@screset\@empty
1107   \let\BabelStrings\bbl@opt@string
1108   \let\BabelOptions\@empty
1109   \let\BabelLanguages\relax
1110   \ifx\originalTeX\@undefined
1111     \let\originalTeX\@empty
1112   \else
1113     \originalTeX
1114   \fi}
1115 \def\LdfInit#1#2{%
1116   \chardef\atcatcode=\catcode`\@
1117   \catcode`\@=11\relax
1118   \chardef\eqcatcode=\catcode`\=
1119   \catcode`\==12\relax
1120   \expandafter\if\expandafter\@backslashchar
1121     \expandafter\@car\string#2\@nil
1122   \ifx#2\@undefined\else
1123     \ldf@quit{#1}%
1124   \fi
1125 \else
1126   \expandafter\ifx\csname#2\endcsname\relax\else
1127     \ldf@quit{#1}%
1128   \fi
1129 \fi
1130 \bbl@ldfinit}

```

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

```

1131 \def\ldf@quit#1{%
1132   \expandafter\main@language\expandafter{#1}%
1133   \catcode`\@=\atcatcode \let\atcatcode\relax
1134   \catcode`\=\eqcatcode \let\eqcatcode\relax
1135   \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.

```

1136 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1137   \bbl@afterlang
1138   \let\bbl@afterlang\relax
1139   \let\BabelModifiers\relax
1140   \let\bbl@screset\relax}%

```

```

1141 \def\ldf@finish#1{%
1142   \loadlocalcfg{#1}%
1143   \bbl@afterldf{#1}%
1144   \expandafter\main@language\expandafter{#1}%
1145   \catcode`\@=\atcatcode \let\atcatcode\relax
1146   \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 .

```

1147 \@onlypreamble\LdfInit
1148 \@onlypreamble\ldf@quit
1149 \@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.

```

1150 \def\main@language#1{%
1151   \def\bbl@main@language{#1}%
1152   \let\language\name\bbl@main@language % TODO. Set localename
1153   \bbl@id@assign
1154   \bbl@patterns{\language\name}}

```

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`.

```

1155 \def\bbl@beforestart{%
1156   \def\@nolanerr##1{%
1157     \bbl@warning{Undefined language '##1' in aux.\Reported}}%
1158   \bbl@usehooks{beforestart}{}%
1159   \global\let\bbl@beforestart\relax}
1160 \AtBeginDocument{%
1161   {\@nameuse\bbl@beforestart}}% Group!
1162   \if@filesw
1163     \providecommand\babel@aux[2]{}%
1164     \immediate\write\@mainaux{%
1165       \string\providecommand\string\babel@aux[2]{}%
1166       \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1167   \fi
1168   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1169   \ifbbl@single % must go after the line above.
1170     \renewcommand\selectlanguage[1]{}%
1171     \renewcommand\foreignlanguage[2]{#2}%
1172     \global\let\babel@aux\@gobbles % Also as flag
1173   \fi
1174   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1175 \def\select@language@x#1{%
1176   \ifcase\bbl@select@type
1177     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1178   \else
1179     \select@language{#1}%
1180   \fi}

```

8.5 Shorthands

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

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

```

1181 \bbl@trace{Shorhands}
1182 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1183 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1184 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1185 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1186 \begingroup
1187 \catcode`#1\active
1188 \nfss@catcodes
1189 \ifnum\catcode`#1=\active
1190 \endgroup
1191 \bbl@add\nfss@catcodes{\@makeother#1}%
1192 \else
1193 \endgroup
1194 \fi
1195 \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.

```

1196 \def\bbl@remove@special#1{%
1197 \begingroup
1198 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1199 \else\noexpand##1\noexpand##2\fi}%
1200 \def\do{\x\do}%
1201 \def\@makeother{\x\@makeother}%
1202 \edef\x{\endgroup
1203 \def\noexpand\dospecials{\dospecials}%
1204 \expandafter\ifx\cname @sanitize\endcname\relax\else
1205 \def\noexpand\@sanitize{\@sanitize}%
1206 \fi}%
1207 \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).

```

1208 \def\bbl@active@def#1#2#3#4{%
1209 \namedef{#3#1}{%
1210 \expandafter\ifx\cname#2@sh#1@\endcname\relax
1211 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1212 \else
1213 \bbl@afterfi\cname#2@sh#1@\endcname
1214 \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.

```

1215 \long\@namedef{#3@arg#1}##1{%
1216   \expandafter\ifx\csname#2@sh@#1@string##1@endcsname\relax
1217     \bbl@afterelse\csname#4#1@endcsname##1%
1218   \else
1219     \bbl@afterfi\csname#2@sh@#1@string##1@endcsname
1220   \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.

```

1221 \def\initiate@active@char#1{%
1222   \bbl@ifunset{active@char\string#1}%
1223   {\bbl@withactive
1224     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1225   {}}

```

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 and preserving some degree of protection).

```

1226 \def\@initiate@active@char#1#2#3{%
1227   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1228   \ifx#1\@undefined
1229     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1230   \else
1231     \bbl@csarg\let{oridef@@#2}#1%
1232     \bbl@csarg\edef{oridef@#2}{%
1233       \let\noexpand#1%
1234       \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
1235   \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*").

```

1236 \ifx#1#3\relax
1237   \expandafter\let\csname normal@char#2@endcsname#3%
1238 \else
1239   \bbl@info{Making #2 an active character}%
1240   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1241   \@namedef{normal@char#2}{%
1242     \textormath{#3}{\csname bbl@oridef@@#2@endcsname}}%
1243   \else
1244     \@namedef{normal@char#2}{#3}%
1245   \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).

```

1246 \bbl@restoreactive{#2}%
1247 \AtBeginDocument{%
1248   \catcode`#2\active
1249   \if@filesw
1250     \immediate\write\@mainaux{\catcode`\string#2\active}%

```

```

1251 \fi}%
1252 \expandafter\bb1@add@special\csname#2\endcsname
1253 \catcode`#2\active
1254 \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⟩`).

```

1255 \let\bb1@tempa\@firstoftwo
1256 \if\string^#2%
1257 \def\bb1@tempa{\noexpand\textormath}%
1258 \else
1259 \ifx\bb1@mathnormal\@undefined\else
1260 \let\bb1@tempa\bb1@mathnormal
1261 \fi
1262 \fi
1263 \expandafter\edef\csname active@char#2\endcsname{%
1264 \bb1@tempa
1265 {\noexpand\if@safe@actives
1266 \noexpand\expandafter
1267 \expandafter\noexpand\csname normal@char#2\endcsname
1268 \noexpand\else
1269 \noexpand\expandafter
1270 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1271 \noexpand\fi}%
1272 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1273 \bb1@csarg\edef{doactive#2}{%
1274 \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!).

```

1275 \bb1@csarg\edef{active@#2}{%
1276 \noexpand\active@prefix\noexpand#1%
1277 \expandafter\noexpand\csname active@char#2\endcsname}%
1278 \bb1@csarg\edef{normal@#2}{%
1279 \noexpand\active@prefix\noexpand#1%
1280 \expandafter\noexpand\csname normal@char#2\endcsname}%
1281 \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.

```

1282 \bb1@active@def#2\user@group{user@active}{language@active}%
1283 \bb1@active@def#2\language@group{language@active}{system@active}%
1284 \bb1@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.

```

1285 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1286 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1287 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1288 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```
1289 \if\string'#2%
1290 \let\prim@s\bbl@prim@s
1291 \let\active@math@prime#1%
1292 \fi
1293 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}
```

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

```
1294 <<{*More package options}>> ≡
1295 \DeclareOption{math=active}{}
1296 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1297 <</More package options>>
```

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

```
1298 \@ifpackagewith{babel}{KeepShorthandsActive}%
1299 {\let\bbl@restoreactive\@gobble}%
1300 {\def\bbl@restoreactive#1{%
1301   \bbl@exp{%
1302     \\\AfterBabelLanguage\\CurrentOption
1303     {\catcode`#1=\the\catcode`#1\relax}%
1304     \\\AtEndOfPackage
1305     {\catcode`#1=\the\catcode`#1\relax}}}%
1306 \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.

```
1307 \def\bbl@sh@select#1#2{%
1308   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1309     \bbl@afterelse\bbl@scndcs
1310   \else
1311     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1312   \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.

```
1313 \begingroup
1314 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1315 {\gdef\active@prefix#1{%
1316   \ifx\protect\@typeset@protect
1317   \else
1318     \ifx\protect\@unexpandable@protect
1319       \noexpand#1%
1320     \else
1321       \protect#1%
1322     \fi
1323   \fi}}
```

```

1323     \expandafter\@gobble
1324     \fi}}
1325 {\gdef\active@prefix#1{%
1326     \ifincsname
1327     \string#1%
1328     \expandafter\@gobble
1329     \else
1330     \ifx\protect\@typeset@protect
1331     \else
1332     \ifx\protect\@unexpandable@protect
1333     \noexpand#1%
1334     \else
1335     \protect#1%
1336     \fi
1337     \expandafter\expandafter\expandafter\@gobble
1338     \fi
1339     \fi}}
1340 \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>`.

```

1341 \newif\if@safe@actives
1342 \@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.

```

1343 \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`.

```

1344 \chardef\bbl@activated\z@
1345 \def\bbl@activate#1{%
1346     \chardef\bbl@activated\@ne
1347     \bbl@withactive{\expandafter\let\expandafter}#1%
1348     \csname bbl@active@\string#1\endcsname}
1349 \def\bbl@deactivate#1{%
1350     \chardef\bbl@activated\tw@
1351     \bbl@withactive{\expandafter\let\expandafter}#1%
1352     \csname bbl@normal@\string#1\endcsname}

```

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

`\bbl@scndcs`

```

1353 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1354 \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.


```

1355 \def\babel@texpdf#1#2#3#4{%
1356   \ifx\texorpdfstring\undefined
1357     \textormath{#1}{#3}%
1358   \else
1359     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1360     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1361   \fi}
1362 %
1363 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1364 \def\@decl@short#1#2#3\@nil#4{%
1365   \def\bbl@tempa{#3}%
1366   \ifx\bbl@tempa\empty
1367     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1368     \bbl@ifunset{#1@sh@\string#2@}{}%
1369     {\def\bbl@tempa{#4}%
1370      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1371      \else
1372        \bbl@info
1373          {Redefining #1 shorthand \string#2\%
1374           in language \CurrentOption}%
1375      \fi}%
1376     \@namedef{#1@sh@\string#2@}{#4}%
1377   \else
1378     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1379     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1380     {\def\bbl@tempa{#4}%
1381      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1382      \else
1383        \bbl@info
1384          {Redefining #1 shorthand \string#2\string#3\%
1385           in language \CurrentOption}%
1386      \fi}%
1387     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1388   \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.

```

1389 \def\textormath{%
1390   \ifmmode
1391     \expandafter\@secondoftwo
1392   \else
1393     \expandafter\@firstoftwo
1394   \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
`\language@group` group ‘english’ and have a system group called ‘system’.
`\system@group`

```

1395 \def\user@group{user}
1396 \def\language@group{english} % TODO. I don't like defaults
1397 \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.

```

1398 \def\useshorthands{%
1399   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1400 \def\bbl@usesh@s#1{%
1401   \bbl@usesh@x

```

```

1402     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1403     {#1}}
1404 \def\bbl@usesesh@x#1#2{%
1405   \bbl@ifshorthand{#2}%
1406   {\def\user@group{user}%
1407     \initiate@active@char{#2}%
1408     #1%
1409     \bbl@activate{#2}}%
1410   {\bbl@error
1411     {I can't declare a shorthand turned off (\string#2)}
1412     {Sorry, but you can't use shorthands which have been\\%
1413       turned off in the package options}}}

```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user@<lang>` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1414 \def\user@language@group{user@\language@group}
1415 \def\bbl@set@user@generic#1#2{%
1416   \bbl@ifunset{user@generic@active#1}%
1417   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1418     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1419     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1420       \expandafter\noexpand\csname normal@char#1\endcsname}%
1421     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1422       \expandafter\noexpand\csname user@active#1\endcsname}}%
1423   \@empty}
1424 \newcommand\defineshorthand[3][user]{%
1425   \edef\bbl@tempa{\zap@space#1 \@empty}%
1426   \bbl@for\bbl@tempb\bbl@tempa{%
1427     \if*\expandafter\@car\bbl@tempb\@nil
1428       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1429       \@expandtwoargs
1430       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1431     \fi
1432     \declare@shorthand{\bbl@tempb}{#2}{#3}}}

```

`\languageshorthands` A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

```

1433 \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`".

```

1434 \def\aliasshorthand#1#2{%
1435   \bbl@ifshorthand{#2}%
1436   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1437     \ifx\document\@notprerr
1438       \@notshorthand{#2}%
1439     \else
1440       \initiate@active@char{#2}%
1441       \expandafter\let\csname active@char\string#2\endcsname
1442         \csname active@char\string#1\endcsname
1443       \expandafter\let\csname normal@char\string#2\endcsname
1444         \csname normal@char\string#1\endcsname
1445       \bbl@activate{#2}%

```

```

1446     \fi
1447     \fi}%
1448     {\bbl@error
1449       {Cannot declare a shorthand turned off (\string#2)}
1450       {Sorry, but you cannot use shorthands which have been\\%
1451         turned off in the package options}}}

```

\@notshorthand

```

1452 \def\@notshorthand#1{%
1453   \bbl@error{%
1454     The character '\string #1' should be made a shorthand character;\\%
1455     add the command \string\usesshorthands\string{#1\string} to
1456     the preamble.\\%
1457     I will ignore your instruction}%
1458   {You may proceed, but expect unexpected results}}

```

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

```

1459 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1460 \DeclareRobustCommand*\shorthandoff{%
1461   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1462 \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.

```

1463 \def\bbl@switch@sh#1#2{%
1464   \ifx#2\@nnil\else
1465     \bbl@ifunset{\bbl@active@\string#2}%
1466     {\bbl@error
1467       {I can't switch '\string#2' on or off--not a shorthand}%
1468       {This character is not a shorthand. Maybe you made\\%
1469         a typing mistake? I will ignore your instruction.}}%
1470     {\ifcase#1%   off, on, off*
1471       \catcode`#2\relax
1472       \or
1473       \catcode`#2\active
1474       \bbl@ifunset{\bbl@shdef@\string#2}%
1475       {}%
1476       {\bbl@withactive{\expandafter\let\expandafter}#2%
1477         \csname bbl@shdef@\string#2\endcsname
1478         \bbl@csarg\let{\shdef@\string#2}\relax}%
1479       \ifcase\bbl@activated\or
1480         \bbl@activate{#2}%
1481       \else
1482         \bbl@deactivate{#2}%
1483       \fi
1484       \or
1485       \bbl@ifunset{\bbl@shdef@\string#2}%
1486       {\bbl@withactive{\bbl@csarg\let{\shdef@\string#2}}#2}%
1487       {}%
1488       \csname bbl@oricat@\string#2\endcsname
1489       \csname bbl@oridef@\string#2\endcsname
1490     \fi}%

```

```

1491 \bbl@afterfi\bbl@switch@sh#1%
1492 \fi}

```

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

```

1493 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1494 \def\bbl@putsh#1{%
1495 \bbl@ifunset{bbl@active@\string#1}%
1496 {\bbl@putsh@i#1\@empty\@nnil}%
1497 {\csname bbl@active@\string#1\endcsname}}
1498 \def\bbl@putsh@i#1#2\@nnil{%
1499 \csname\language@group @sh@\string#1@%
1500 \ifx\@empty#2\else\string#2\fi\endcsname}
1501 \ifx\bbl@opt@shorthands\@nnil\else
1502 \let\bbl@s@initiate@active@char\initiate@active@char
1503 \def\initiate@active@char#1{%
1504 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1505 \let\bbl@s@switch@sh\bbl@switch@sh
1506 \def\bbl@switch@sh#1#2{%
1507 \ifx#2\@nnil\else
1508 \bbl@afterfi
1509 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1510 \fi}
1511 \let\bbl@s@activate\bbl@activate
1512 \def\bbl@activate#1{%
1513 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1514 \let\bbl@s@deactivate\bbl@deactivate
1515 \def\bbl@deactivate#1{%
1516 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1517 \fi

```

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

```

1518 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

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

```

1519 \def\bbl@prim@s{%
1520 \prime\futurelet\@let@token\bbl@pr@m@s}
1521 \def\bbl@if@primes#1#2{%
1522 \ifx#1\@let@token
1523 \expandafter\@firstoftwo
1524 \else\ifx#2\@let@token
1525 \bbl@afterelse\expandafter\@firstoftwo
1526 \else
1527 \bbl@afterfi\expandafter\@secondoftwo
1528 \fi\fi}
1529 \begingroup
1530 \catcode`\^=7 \catcode`\*= \active \lccode`\*=`^
1531 \catcode`\'=12 \catcode`\`= \active \lccode`\`= ` '
1532 \lowercase{%
1533 \gdef\bbl@pr@m@s{%
1534 \bbl@if@primes" "%
1535 \pr@@@s
1536 {\bbl@if@primes*^\pr@@@t\egroup}}
1537 \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).

```
1538 \initiate@active@char{~}
1539 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1540 \bbl@activate{~}
```

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

```
1541 \expandafter\def\csname OT1dqpos\endcsname{127}
1542 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1543 \ifx\f@encoding\undefined
1544   \def\f@encoding{OT1}
1545 \fi
```

8.6 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.

```
1546 \bbl@trace{Language attributes}
1547 \newcommand\languageattribute[2]{%
1548   \def\bbl@tempc{#1}%
1549   \bbl@fixname\bbl@tempc
1550   \bbl@iflanguage\bbl@tempc{%
1551     \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.

```
1552     \ifx\bbl@known@attribs\undefined
1553       \in@false
1554     \else
1555       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1556     \fi
1557     \ifin@
1558       \bbl@warning{%
1559         You have more than once selected the attribute '##1'\%
1560         for language #1. Reported}%
1561     \else
```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX -code.

```
1562       \bbl@exp{%
1563         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1564       \edef\bbl@tempa{\bbl@tempc-##1}%
1565       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1566       {\csname\bbl@tempc @attr##1\endcsname}%
1567       {\@attrerr{\bbl@tempc}{##1}}%
1568     \fi}}
1569 \@onlypreamble\languageattribute
```

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

```
1570 \newcommand*{\@attrerr}[2]{%
1571   \bbl@error
1572   {The attribute #2 is unknown for language #1.}%
1573   {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}`.

```
1574 \def\bbl@declare@ttribute#1#2#3{%
1575   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1576   \ifin@
1577     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1578   \fi
1579   \bbl@add@list\bbl@attributes{#1-#2}%
1580   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

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

```
1581 \def\bbl@ifattributeset#1#2#3#4{%
1582   \ifx\bbl@known@attribs\@undefined
1583     \in@false
1584   \else
1585     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1586   \fi
1587   \ifin@
1588     \bbl@afterelse#3%
1589   \else
1590     \bbl@afterfi#4%
1591   \fi}
```

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

```
1592 \def\bbl@ifknown@ttrib#1#2{%
1593   \let\bbl@tempa\@secondoftwo
1594   \bbl@loopx\bbl@tempb{#2}{%
1595     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1596     \ifin@
1597       \let\bbl@tempa\@firstoftwo
1598     \else
1599       \fi}%
1600   \bbl@tempa}
```

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

```
1601 \def\bbl@clear@ttribs{%
1602   \ifx\bbl@attributes\@undefined\else
1603     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1604       \expandafter\bbl@clear@ttrib\bbl@tempa.
1605     }%
1606     \let\bbl@attributes\@undefined
```

```

1607 \fi}
1608 \def\bbl@clear@ttrib#1-#2.{%
1609 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1610 \AtBeginDocument{\bbl@clear@ttribs}

```

8.7 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`

```

1611 \bbl@trace{Macros for saving definitions}
1612 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1613 \newcount\babel@savecnt
1614 \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.

```

1615 \def\babel@save#1{%
1616 \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1617 \toks@\expandafter{\originalTeX\let#1=}%
1618 \bbl@exp{%
1619 \def\\originalTeX{\the\toks@<\babel@\number\babel@savecnt>\relax}}%
1620 \advance\babel@savecnt\@ne}
1621 \def\babel@savevariable#1{%
1622 \toks@\expandafter{\originalTeX #1=}%
1623 \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

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

```

1624 \def\bbl@frenchspacing{%
1625 \ifnum\the\sffcode\`.\=@m
1626 \let\bbl@nonfrenchspacing\relax
1627 \else
1628 \frenchspacing
1629 \let\bbl@nonfrenchspacing\nonfrenchspacing
1630 \fi}
1631 \let\bbl@nonfrenchspacing\nonfrenchspacing
1632 \let\bbl@elt\relax
1633 \edef\bbl@fs@chars{%
1634 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1635 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1636 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1637 \def\bbl@pre@fs{%

```

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

```

1638 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1639 \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1640 \def\bbl@post@fs{%
1641 \bbl@save@sfcodes
1642 \edef\bbl@tempa{\bbl@cl{frspc}}%
1643 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1644 \if u\bbl@tempa % do nothing
1645 \else\if n\bbl@tempa % non french
1646 \def\bbl@elt##1##2##3{%
1647 \ifnum\sfcode`##1=##2\relax
1648 \babel@savevariable{\sfcode`##1}%
1649 \sfcode`##1=##3\relax
1650 \fi}%
1651 \bbl@fs@chars
1652 \else\if y\bbl@tempa % french
1653 \def\bbl@elt##1##2##3{%
1654 \ifnum\sfcode`##1=##3\relax
1655 \babel@savevariable{\sfcode`##1}%
1656 \sfcode`##1=##2\relax
1657 \fi}%
1658 \bbl@fs@chars
1659 \fi\fi\fi}

```

8.8 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.

```

1660 \bbl@trace{Short tags}
1661 \def\babeltags#1{%
1662 \edef\bbl@tempa{\zap@space#1 \@empty}%
1663 \def\bbl@tempb##1=##2\@{ }%
1664 \edef\bbl@tempc{%
1665 \noexpand\newcommand
1666 \expandafter\noexpand\csname ##1\endcsname{%
1667 \noexpand\protect
1668 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1669 \noexpand\newcommand
1670 \expandafter\noexpand\csname text##1\endcsname{%
1671 \noexpand\foreignlanguage{##2}}
1672 \bbl@tempc}%
1673 \bbl@for\bbl@tempa\bbl@tempa{%
1674 \expandafter\bbl@tempb\bbl@tempa\@{ }

```

8.9 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.

```

1675 \bbl@trace{Hyphens}
1676 \@onlypreamble\babelhyphenation
1677 \AtEndOfPackage{%
1678 \newcommand\babelhyphenation[2][\@empty]{%
1679 \ifx\bbl@hyphenation@relax
1680 \let\bbl@hyphenation@\@empty
1681 \fi
1682 \ifx\bbl@hyphlist\@empty\else
1683 \bbl@warning{%

```



```

1684         You must not intermingle \string\selectlanguage\space and\\%
1685         \string\babelhyphenation\space or some exceptions will not\\%
1686         be taken into account. Reported}%
1687     \fi
1688     \ifx\@empty#1%
1689         \protected@edef\bb1@hyphenation@{\bb1@hyphenation@\space#2}%
1690     \else
1691         \bb1@vforeach{#1}{%
1692             \def\bb1@tempa{##1}%
1693             \bb1@fixname\bb1@tempa
1694             \bb1@iflanguage\bb1@tempa{%
1695                 \bb1@csarg\protected@edef{hyphenation@\bb1@tempa}{%
1696                     \bb1@ifunset{bb1@hyphenation@\bb1@tempa}%
1697                     {}%
1698                     {\csname bb1@hyphenation@\bb1@tempa\endcsname\space}%
1699                     #2}}}%
1700     \fi}}

```

`\bb1@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`³².

```

1701 \def\bb1@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1702 \def\bb1@t@one{T1}
1703 \def\allowhyphens{\ifx\cf@encoding\bb1@t@one\else\bb1@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`.

```

1704 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1705 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
1706 \def\bb1@hyphen{%
1707     \@ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i \@empty}}
1708 \def\bb1@hyphen@i#1#2{%
1709     \bb1@ifunset{bb1@hy@#1#2\@empty}%
1710     {\csname bb1@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1711     {\csname bb1@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.

```

1712 \def\bb1@usehyphen#1{%
1713     \leavevmode
1714     \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1715     \nobreak\hskip\z@skip}
1716 \def\bb1@usehyphen#1{%
1717     \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1718 \def\bb1@hyphenchar{%
1719     \ifnum\hyphenchar\font=\m@ne
1720         \babelnullhyphen
1721     \else
1722         \char\hyphenchar\font
1723     \fi}

```

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

Finally, we define the hyphen “types”. Their names will not change, so you may use them in ldf’s. After a space, the `\mbox` in `\bbl@hy@nbreak` is redundant.

```

1724 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1725 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1726 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1727 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1728 \def\bbl@hy@nbreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1729 \def\bbl@hy@nbreak{\mbox{\bbl@hyphenchar}}
1730 \def\bbl@hy@repeat{%
1731   \bbl@usehyphen{%
1732     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1733 \def\bbl@hy@repeat{%
1734   \bbl@usehyphen{%
1735     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1736 \def\bbl@hy@empty{\hskip\z@skip}
1737 \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.

```

1738 \def\bbl@disc#1#2{\nbreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

8.10 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.

```

1739 \bbl@trace{Multiencoding strings}
1740 \def\bbl@tglobal#1{\global\let#1#1}
1741 \def\bbl@reacatcode#1{% TODO. Used only once?
1742   \@tempcnta="7F
1743   \def\bbl@tempa{%
1744     \ifnum\@tempcnta>"FF\else
1745       \catcode\@tempcnta=#1\relax
1746       \advance\@tempcnta@ne
1747       \expandafter\bbl@tempa
1748     \fi}%
1749   \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 `\langle lang\rangle\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).

```

1750 \ifpackagewith{babel}{nocase}%
1751   {\let\bbl@patchuclc\relax}%
1752   {\def\bbl@patchuclc{%
1753     \global\let\bbl@patchuclc\relax
1754     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}}%

```

```

1755 \gdef\bbl@uclc##1{%
1756 \let\bbl@encoded\bbl@encoded@uclc
1757 \bbl@ifunset{\language @bbl@uclc}% and resumes it
1758 {##1}%
1759 {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1760 \csname\language @bbl@uclc\endcsname}%
1761 {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1762 \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1763 \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
1764 << *More package options >> ≡
1765 \DeclareOption{nocase}{}
1766 << /More package options >>

```

The following package options control the behavior of \SetString.

```

1767 << *More package options >> ≡
1768 \let\bbl@opt@strings\@nnil % accept strings=value
1769 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1770 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1771 \def\BabelStringsDefault{generic}
1772 << /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.

```

1773 \@onlypreamble\StartBabelCommands
1774 \def\StartBabelCommands{%
1775 \begingroup
1776 \bbl@recatcode{11}%
1777 <<Macros local to BabelCommands>>
1778 \def\bbl@provstring##1##2{%
1779 \providecommand##1{##2}%
1780 \bbl@tglobal##1}%
1781 \global\let\bbl@scafter\@empty
1782 \let\StartBabelCommands\bbl@startcmds
1783 \ifx\BabelLanguages\relax
1784 \let\BabelLanguages\CurrentOption
1785 \fi
1786 \begingroup
1787 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1788 \StartBabelCommands}
1789 \def\bbl@startcmds{%
1790 \ifx\bbl@screset\@nnil\else
1791 \bbl@usehooks{stopcommands}{}%
1792 \fi
1793 \endgroup
1794 \begingroup
1795 \@ifstar
1796 {\ifx\bbl@opt@strings\@nnil
1797 \let\bbl@opt@strings\BabelStringsDefault
1798 \fi
1799 \bbl@startcmds@i}%
1800 \bbl@startcmds@i}
1801 \def\bbl@startcmds@i#1#2{%
1802 \edef\bbl@L{\zap@space#1 \@empty}%
1803 \edef\bbl@G{\zap@space#2 \@empty}%
1804 \bbl@startcmds@ii}
1805 \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.

```
1806 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1807   \let\SetString\@gobbletwo
1808   \let\bbbl@stringdef\@gobbletwo
1809   \let\AfterBabelCommands\@gobble
1810   \ifx\@empty#1%
1811     \def\bbbl@sc@label{generic}%
1812     \def\bbbl@encstring##1##2{%
1813       \ProvideTextCommandDefault##1{##2}%
1814       \bbbl@toglobal##1%
1815       \expandafter\bbbl@toglobal\csname\string?\string##1\endcsname}%
1816     \let\bbbl@sctest\in@true
1817   \else
1818     \let\bbbl@sc@charset\space % <- zapped below
1819     \let\bbbl@sc@fontenc\space % <- " "
1820     \def\bbbl@tempa##1=##2\@nil{%
1821       \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1822     \bbbl@vforeach{label=#1}{\bbbl@tempa##1\@nil}%
1823     \def\bbbl@tempa##1 ##2{% space -> comma
1824       ##1%
1825       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
1826     \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
1827     \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
1828     \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
1829     \def\bbbl@encstring##1##2{%
1830       \bbbl@foreach\bbbl@sc@fontenc{%
1831         \bbbl@ifunset{T@####1}%
1832         {}%
1833         {\ProvideTextCommand##1{####1}{##2}%
1834         \bbbl@toglobal##1%
1835         \expandafter
1836         \bbbl@toglobal\csname####1\string##1\endcsname}}}%
1837     \def\bbbl@sctest{%
1838       \bbbl@xin@{\bbbl@opt@strings,}{,\bbbl@sc@label,\bbbl@sc@fontenc,}}%
1839   \fi
1840   \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
1841   \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded
1842     \let\AfterBabelCommands\bbbl@aftercmds
1843     \let\SetString\bbbl@setstring
1844     \let\bbbl@stringdef\bbbl@encstring
1845   \else % ie, strings=value
1846     \bbbl@sctest
1847   \ifin@
1848     \let\AfterBabelCommands\bbbl@aftercmds
1849     \let\SetString\bbbl@setstring
1850     \let\bbbl@stringdef\bbbl@provstring
1851   \fi\fi\fi
1852   \bbbl@scswitch
1853   \ifx\bbbl@G\@empty
1854     \def\SetString##1##2{%
```

```

1855 \bbl@error{Missing group for string \string##1}%
1856 {You must assign strings to some category, typically\\%
1857 captions or extras, but you set none}}%
1858 \fi
1859 \ifx\@empty#1%
1860 \bbl@usehooks{defaultcommands}{}%
1861 \else
1862 \@expandtwoargs
1863 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}}%
1864 \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).

```

1865 \def\bbl@forlang#1#2{%
1866 \bbl@for#1\bbl@L{%
1867 \bbl@xin@{,#1},{, \BabelLanguages,}%
1868 \ifin@#2\relax\fi}}
1869 \def\bbl@scswitch{%
1870 \bbl@forlang\bbl@tempa{%
1871 \ifx\bbl@G\@empty\else
1872 \ifx\SetString\@gobbletwo\else
1873 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1874 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1875 \ifin@\else
1876 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1877 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1878 \fi
1879 \fi
1880 \fi}}
1881 \AtEndOfPackage{%
1882 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1883 \let\bbl@scswitch\relax
1884 \@onlypreamble\EndBabelCommands
1885 \def\EndBabelCommands{%
1886 \bbl@usehooks{stopcommands}{}%
1887 \endgroup
1888 \endgroup
1889 \bbl@scafter}
1890 \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.

```

1891 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1892 \bbl@forlang\bbl@tempa{%
1893 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1894 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1895 {\bbl@exp{%
1896 \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%

```

```

1897     {}%
1898     \def\BabelString{#2}%
1899     \bbl@usehooks{stringprocess}{}%
1900     \expandafter\bbl@stringdef
1901     \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`.

```

1902 \ifx\bbl@opt@strings\relax
1903   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1904   \bbl@patchuclc
1905   \let\bbl@encoded\relax
1906   \def\bbl@encoded@uclc#1{%
1907     \@inmathwarn#1%
1908     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1909       \expandafter\ifx\csname ?\string#1\endcsname\relax
1910         \TextSymbolUnavailable#1%
1911       \else
1912         \csname ?\string#1\endcsname
1913       \fi
1914     \else
1915       \csname\cf@encoding\string#1\endcsname
1916     \fi}
1917 \else
1918   \def\bbl@scset#1#2{\def#1{#2}}
1919 \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.

```

1920 <<(*Macros local to BabelCommands)>> ≡
1921 \def\SetStringLoop##1##2{%
1922   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1923   \count@\z@
1924   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1925     \advance\count@\@ne
1926     \toks@\expandafter{\bbl@tempa}%
1927     \bbl@exp{%
1928       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1929       \count@=\the\count@\relax}}}%
1930 <</Macros local to BabelCommands>>

```

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

```

1931 \def\bbl@aftercmds#1{%
1932   \toks@\expandafter{\bbl@scafter#1}%
1933   \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.

```

1934 <<(*Macros local to BabelCommands)>> ≡
1935 \newcommand\SetCase[3][{}{%
1936   \bbl@patchuclc
1937   \bbl@forlang\bbl@tempa{%
1938     \expandafter\bbl@encstring
1939     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1940     \expandafter\bbl@encstring

```

```

1941      \csname\bb1@tempa @bb1@uc\endcsname{##2}%
1942      \expandafter\bb1@encstring
1943      \csname\bb1@tempa @bb1@lc\endcsname{##3}}}%
1944 <</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.

```

1945 <<*Macros local to BabelCommands>> ≡
1946 \newcommand\SetHyphenMap[1]{%
1947   \bb1@forlang\bb1@tempa{%
1948     \expandafter\bb1@stringdef
1949     \csname\bb1@tempa @bb1@hyphenmap\endcsname{##1}}}%
1950 <</Macros local to BabelCommands>>

```

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

```

1951 \newcommand\BabelLower[2]{% one to one.
1952   \ifnum\lccode#1=#2\else
1953     \babel@savevariable{\lccode#1}%
1954     \lccode#1=#2\relax
1955   \fi}
1956 \newcommand\BabelLowerMM[4]{% many-to-many
1957   \@tempcnta=#1\relax
1958   \@tempcntb=#4\relax
1959   \def\bb1@tempa{%
1960     \ifnum\@tempcnta>#2\else
1961       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1962       \advance\@tempcnta#3\relax
1963       \advance\@tempcntb#3\relax
1964       \expandafter\bb1@tempa
1965     \fi}%
1966   \bb1@tempa}
1967 \newcommand\BabelLowerMO[4]{% many-to-one
1968   \@tempcnta=#1\relax
1969   \def\bb1@tempa{%
1970     \ifnum\@tempcnta>#2\else
1971       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1972       \advance\@tempcnta#3
1973       \expandafter\bb1@tempa
1974     \fi}%
1975   \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1976 <<*More package options>> ≡
1977 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
1978 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
1979 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
1980 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
1981 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1982 <</More package options>>

```

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

```

1983 \AtEndOfPackage{%
1984   \ifx\bb1@opt@hyphenmap\undefined
1985     \bb1@xin@{,}{\bb1@language@opts}%
1986     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
1987   \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.

```

1988 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
1989 \@ifstar\bb1@setcaption@s\bb1@setcaption@x}
1990 \def\bb1@setcaption@x#1#2#3{% language caption-name string
1991 \bb1@trim@def\bb1@tempa{#2}%
1992 \bb1@xin@{.template}{\bb1@tempa}%
1993 \ifin@
1994 \bb1@ini@captions@template{#3}{#1}%
1995 \else
1996 \edef\bb1@tempd{%
1997 \expandafter\expandafter\expandafter
1998 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1999 \bb1@xin@
2000 {\expandafter\string\csname #2name\endcsname}%
2001 {\bb1@tempd}%
2002 \ifin@ % Renew caption
2003 \bb1@xin@{\string\bb1@scset}{\bb1@tempd}%
2004 \ifin@
2005 \bb1@exp{%
2006 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2007 {\\\bb1@scset\<#2name>\<#1#2name>}%
2008 {}}%
2009 \else % Old way converts to new way
2010 \bb1@ifunset{#1#2name}%
2011 {\bb1@exp{%
2012 \\\bb1@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2013 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2014 {\def\<#2name>\<#1#2name>}}%
2015 {}}}%
2016 {}%
2017 \fi
2018 \else
2019 \bb1@xin@{\string\bb1@scset}{\bb1@tempd}% New
2020 \ifin@ % New way
2021 \bb1@exp{%
2022 \\\bb1@add\<captions#1>\\\bb1@scset\<#2name>\<#1#2name>}%
2023 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2024 {\\\bb1@scset\<#2name>\<#1#2name>}%
2025 {}}%
2026 \else % Old way, but defined in the new way
2027 \bb1@exp{%
2028 \\\bb1@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2029 \\\bb1@ifsamestring{\bb1@tempa}{\language}%
2030 {\def\<#2name>\<#1#2name>}}%
2031 {}}%
2032 \fi%
2033 \fi
2034 \@namedef{#1#2name}{#3}%
2035 \toks@\expandafter{\bb1@captionslist}%
2036 \bb1@exp{\in@{\<#2name>}{\the\toks@}}%
2037 \ifin@\else
2038 \bb1@exp{\\\bb1@add\\bb1@captionslist{\<#2name>}}%
2039 \bb1@toggle\bb1@captionslist
2040 \fi
2041 \fi}
2042 % \def\bb1@setcaption@s#1#2#3{} % TODO. Not yet implemented

```


8.11 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.

```
2043 \bbl@trace{Macros related to glyphs}
2044 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2045   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2046   \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.

```
2047 \def\save@sf@q#1{\leavevmode
2048   \begingroup
2049   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2050   \endgroup}
```

8.12 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`.

8.12.1 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.

```
2051 \ProvideTextCommand{\quotedblbase}{OT1}{%
2052   \save@sf@q{\set@low@box{\textquotedblright\}%
2053   \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.

```
2054 \ProvideTextCommandDefault{\quotedblbase}{%
2055   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2056 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2057   \save@sf@q{\set@low@box{\textquoteright\}%
2058   \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.

```
2059 \ProvideTextCommandDefault{\quotesinglbase}{%
2060   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2061 \ProvideTextCommand{\guillemetleft}{OT1}{%
2062   \ifmmode
2063     \ll
2064   \else
2065     \save@sf@q{\nobreak
2066       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2067     \fi}
2068 \ProvideTextCommand{\guillemetright}{OT1}{%
2069   \ifmmode
2070     \gg
2071   \else
2072     \save@sf@q{\nobreak
2073       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2074     \fi}
```

```

2075 \ProvideTextCommand{\guillemotleft}{OT1}{%
2076   \ifmmode
2077     \ll
2078   \else
2079     \save@sf@q{\nobreak
2080       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2081   \fi}
2082 \ProvideTextCommand{\guillemotright}{OT1}{%
2083   \ifmmode
2084     \gg
2085   \else
2086     \save@sf@q{\nobreak
2087       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2088   \fi}

```

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

```

2089 \ProvideTextCommandDefault{\guillemetleft}{%
2090   \UseTextSymbol{OT1}{\guillemetleft}}
2091 \ProvideTextCommandDefault{\guillemetright}{%
2092   \UseTextSymbol{OT1}{\guillemetright}}
2093 \ProvideTextCommandDefault{\guillemotleft}{%
2094   \UseTextSymbol{OT1}{\guillemotleft}}
2095 \ProvideTextCommandDefault{\guillemotright}{%
2096   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2097 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2098   \ifmmode
2099     <%
2100   \else
2101     \save@sf@q{\nobreak
2102       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2103   \fi}
2104 \ProvideTextCommand{\guilsinglright}{OT1}{%
2105   \ifmmode
2106     >%
2107   \else
2108     \save@sf@q{\nobreak
2109       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2110   \fi}

```

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

```

2111 \ProvideTextCommandDefault{\guilsinglleft}{%
2112   \UseTextSymbol{OT1}{\guilsinglleft}}
2113 \ProvideTextCommandDefault{\guilsinglright}{%
2114   \UseTextSymbol{OT1}{\guilsinglright}}

```

8.12.2 Letters

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

```

2115 \DeclareTextCommand{\ij}{OT1}{%
2116   i\kern-0.02em\bbl@allowhyphens j}
2117 \DeclareTextCommand{\IJ}{OT1}{%
2118   I\kern-0.02em\bbl@allowhyphens J}
2119 \DeclareTextCommand{\ij}{T1}{\char188}
2120 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2121 \ProvideTextCommandDefault{\ij}{%
2122   \UseTextSymbol{OT1}{\ij}}
2123 \ProvideTextCommandDefault{\IJ}{%
2124   \UseTextSymbol{OT1}{\IJ}}
```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, 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).

```
2125 \def\crrtic@{\hrule height0.1ex width0.3em}
2126 \def\crttic@{\hrule height0.1ex width0.33em}
2127 \def\ddj@{%
2128   \setbox0\hbox{d}\dimen@=\ht0
2129   \advance\dimen@1ex
2130   \dimen@.45\dimen@
2131   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2132   \advance\dimen@ii.5ex
2133   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic}}}}
2134 \def\DDJ@{%
2135   \setbox0\hbox{D}\dimen@=.55\ht0
2136   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2137   \advance\dimen@ii.15ex % correction for the dash position
2138   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2139   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2140   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic}}}}
2141 %
2142 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2143 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2144 \ProvideTextCommandDefault{\dj}{%
2145   \UseTextSymbol{OT1}{\dj}}
2146 \ProvideTextCommandDefault{\DJ}{%
2147   \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.

```
2148 \DeclareTextCommand{\SS}{OT1}{SS}
2149 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

8.12.3 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`

```
2150 \ProvideTextCommandDefault{\glq}{%
2151   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2152 \ProvideTextCommand{\grq}{T1}{%
2153   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2154 \ProvideTextCommand{\grq}{TU}{%
2155   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2156 \ProvideTextCommand{\grq}{OT1}{%
2157   \save@sf@q{\kern-.0125em
```

```

2158 \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2159 \kern.07em\relax}}
2160 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

\glqq The ‘german’ double quotes.
\grqq 2161 \ProvideTextCommandDefault{\glqq}{%
2162 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2163 \ProvideTextCommand{\grqq}{T1}{%
2164 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2165 \ProvideTextCommand{\grqq}{TU}{%
2166 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2167 \ProvideTextCommand{\grqq}{OT1}{%
2168 \save@sf@q{\kern-.07em
2169 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2170 \kern.07em\relax}}
2171 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

\flq The ‘french’ single guillemets.
\frq 2172 \ProvideTextCommandDefault{\flq}{%
2173 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2174 \ProvideTextCommandDefault{\frq}{%
2175 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

\flqq The ‘french’ double guillemets.
\frqq 2176 \ProvideTextCommandDefault{\flqq}{%
2177 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2178 \ProvideTextCommandDefault{\frqq}{%
2179 \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

8.12.4 Umlauts and tremas

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

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

```

2180 \def\umlauthigh{%
2181 \def\bbl@umlauta##1{\leavevmode\bgroup%
2182 \expandafter\accent\csname\fontencoding dqpos\endcsname
2183 ##1\bbl@allowhyphens\egroup}%
2184 \let\bbl@umlaute\bbl@umlauta}
2185 \def\umlautlow{%
2186 \def\bbl@umlauta{\protect\lower@umlaut}}
2187 \def\umlautelow{%
2188 \def\bbl@umlaute{\protect\lower@umlaut}}
2189 \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.

```

2190 \expandafter\ifx\csname U@D\endcsname\relax
2191 \csname newdimen\endcsname U@D
2192 \fi

```

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

```

2193 \def\lower@umlaut#1{%
2194   \leavevmode\bggroup
2195     \U@D 1ex%
2196     {\setbox\z@\hbox{%
2197       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2198       \dimen@ -.45ex\advance\dimen@\ht\z@
2199       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2200     \expandafter\accent\csname\fontencoding dqpos\endcsname
2201     \fontdimen5\font\U@D #1%
2202   \egroup}

```

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

```

2203 \AtBeginDocument{%
2204   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2205   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2206   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2207   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2208   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2209   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2210   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2211   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2212   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2213   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2214   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%

```

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

```

2215 \ifx\l@english\undefined
2216   \chardef\l@english\z@
2217 \fi
2218 % The following is used to cancel rules in ini files (see Amharic).
2219 \ifx\l@unhyphenated\undefined
2220   \newlanguage\l@unhyphenated
2221 \fi

```

8.13 Layout

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

```

2222 \bbl@trace{Bidi layout}
2223 \providecommand\IfBabelLayout[3]{#3}%
2224 \newcommand\BabelPatchSection[1]{%
2225   \ifundefined{#1}{%
2226     \bbl@exp{\let\bbl@ss@#1>\<#1}%
2227     \@namedef{#1}{%
2228       \ifstar{\bbl@presec{s}{#1}}%
2229       {\@dblarg{\bbl@presec{x}{#1}}}}%

```

```

2230 \def\bbl@presec@x#1[#2]#3{%
2231   \bbl@exp{%
2232     \\select@language@x{\bbl@main@language}%
2233     \\bbl@cs{sspre@#1}%
2234     \\bbl@cs{ss@#1}%
2235     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2236     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2237     \\select@language@x{\language}}%
2238 \def\bbl@presec@s#1#2{%
2239   \bbl@exp{%
2240     \\select@language@x{\bbl@main@language}%
2241     \\bbl@cs{sspre@#1}%
2242     \\bbl@cs{ss@#1}*%
2243     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2244     \\select@language@x{\language}}%
2245 \IfBabelLayout{sectioning}%
2246   {\BabelPatchSection{part}%
2247    \BabelPatchSection{chapter}%
2248    \BabelPatchSection{section}%
2249    \BabelPatchSection{subsection}%
2250    \BabelPatchSection{subsubsection}%
2251    \BabelPatchSection{paragraph}%
2252    \BabelPatchSection{subparagraph}%
2253    \def\babel@toc#1{%
2254      \select@language@x{\bbl@main@language}}}%
2255 \IfBabelLayout{captions}%
2256   {\BabelPatchSection{caption}}%

```

8.14 Load engine specific macros

```

2257 \bbl@trace{Input engine specific macros}
2258 \ifcase\bbl@engine
2259   \input txtbabel.def
2260 \or
2261   \input luababel.def
2262 \or
2263   \input xebabel.def
2264 \fi

```

8.15 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.

```

2265 \bbl@trace{Creating languages and reading ini files}
2266 \let\bbl@extend@ini\@gobble
2267 \newcommand\babelprovide[2][{}]{%
2268   \let\bbl@savelangname\language
2269   \edef\bbl@savelocaleid{\the\localeid}%
2270   % Set name and locale id
2271   \edef\language{#2}%
2272   \bbl@id@assign
2273   % Initialize keys
2274   \let\bbl@KVP@captions\@nil
2275   \let\bbl@KVP@date\@nil
2276   \let\bbl@KVP@import\@nil
2277   \let\bbl@KVP@main\@nil
2278   \let\bbl@KVP@script\@nil
2279   \let\bbl@KVP@language\@nil

```

```

2280 \let\bb1@KVP@hyphenrules\@nil
2281 \let\bb1@KVP@linebreaking\@nil
2282 \let\bb1@KVP@justification\@nil
2283 \let\bb1@KVP@mapfont\@nil
2284 \let\bb1@KVP@maparabic\@nil
2285 \let\bb1@KVP@mapdigits\@nil
2286 \let\bb1@KVP@intraspace\@nil
2287 \let\bb1@KVP@intrapenalty\@nil
2288 \let\bb1@KVP@onchar\@nil
2289 \let\bb1@KVP@transforms\@nil
2290 \global\let\bb1@release@transforms\@empty
2291 \let\bb1@KVP@alph\@nil
2292 \let\bb1@KVP@Alph\@nil
2293 \let\bb1@KVP@labels\@nil
2294 \bb1@csarg\let{KVP@labels*}\@nil
2295 \global\let\bb1@inidata\@empty
2296 \global\let\bb1@extend@ini\@gobble
2297 \gdef\bb1@key@list{;}%
2298 \bb1@forkv{#1}{% TODO - error handling
2299   \in{/{}}{##1}%
2300   \ifin@
2301     \global\let\bb1@extend@ini\bb1@extend@ini@aux
2302     \bb1@renewinikey##1\@{##2}%
2303   \else
2304     \bb1@csarg\def{KVP@##1}{##2}%
2305   \fi}%
2306 \chardef\bb1@howloaded=% 0:none; 1:ldf without ini; 2:ini
2307 \bb1@ifunset{date#2}\z@{\bb1@ifunset{\bb1@llevel@#2}\@ne\tw}%
2308 % == init ==
2309 \ifx\bb1@screset\@undefined
2310   \bb1@ldfinit
2311 \fi
2312 % ==
2313 \let\bb1@lbkflag\relax % \@empty = do setup linebreak
2314 \ifcase\bb1@howloaded
2315   \let\bb1@lbkflag\@empty % new
2316 \else
2317   \ifx\bb1@KVP@hyphenrules\@nil\else
2318     \let\bb1@lbkflag\@empty
2319   \fi
2320   \ifx\bb1@KVP@import\@nil\else
2321     \let\bb1@lbkflag\@empty
2322   \fi
2323 \fi
2324 % == import, captions ==
2325 \ifx\bb1@KVP@import\@nil\else
2326   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
2327   {\ifx\bb1@initload\relax
2328     \begingroup
2329       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
2330       \bb1@input@texini{##2}%
2331     \endgroup
2332   \else
2333     \xdef\bb1@KVP@import{\bb1@initload}%
2334   \fi}%
2335 {}%
2336 \fi
2337 \ifx\bb1@KVP@captions\@nil
2338   \let\bb1@KVP@captions\bb1@KVP@import

```

```

2339 \fi
2340 % ==
2341 \ifx\bbbl@KVP@transforms\@nil\else
2342   \bbbl@replace\bbbl@KVP@transforms{ }{,}%
2343 \fi
2344 % == Load ini ==
2345 \ifcase\bbbl@howloaded
2346   \bbbl@provide@new{#2}%
2347 \else
2348   \bbbl@ifblank{#1}%
2349   {}% With \bbbl@load@basic below
2350   {\bbbl@provide@renew{#2}}%
2351 \fi
2352 % Post tasks
2353 % -----
2354 % == subsequent calls after the first provide for a locale ==
2355 \ifx\bbbl@inidata\@empty\else
2356   \bbbl@extend@ini{#2}%
2357 \fi
2358 % == ensure captions ==
2359 \ifx\bbbl@KVP@captions\@nil\else
2360   \bbbl@ifunset{bbbl@extracaps@#2}%
2361     {\bbbl@exp{\labelensure[exclude=\\today]{#2}}}%
2362     {\bbbl@exp{\labelensure[exclude=\\today,
2363       include=\bbbl@extracaps@#2]}{#2}}%
2364   \bbbl@ifunset{bbbl@ensure@language}%
2365     {\bbbl@exp{%
2366       \\DeclareRobustCommand\<bbbl@ensure@language>[1]{%
2367         \\foreignlanguage{language}%
2368         {###1}}}%
2369     }%
2370   \bbbl@exp{%
2371     \\bbbl@tglobal\<bbbl@ensure@language>%
2372     \\bbbl@tglobal\<bbbl@ensure@language\space>}%
2373 \fi
2374 % ==
2375 % At this point all parameters are defined if 'import'. Now we
2376 % execute some code depending on them. But what about if nothing was
2377 % imported? We just set the basic parameters, but still loading the
2378 % whole ini file.
2379 \bbbl@load@basic{#2}%
2380 % == script, language ==
2381 % Override the values from ini or defines them
2382 \ifx\bbbl@KVP@script\@nil\else
2383   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2384 \fi
2385 \ifx\bbbl@KVP@language\@nil\else
2386   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2387 \fi
2388 % == onchar ==
2389 \ifx\bbbl@KVP@onchar\@nil\else
2390   \bbbl@luahyphenate
2391   \directlua{
2392     if Babel.locale_mapped == nil then
2393       Babel.locale_mapped = true
2394       Babel.linebreaking.add_before(Babel.locale_map)
2395       Babel.loc_to_scr = {}
2396       Babel.chr_to_loc = Babel.chr_to_loc or {}
2397     end}%

```



```

2398 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2399 \ifin@
2400 \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2401 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2402 \fi
2403 \bbl@exp{\bbl@add\bbl@starthyphens
2404 {\bbl@patterns@lua{\language}}}%
2405 % TODO - error/warning if no script
2406 \directlua{
2407   if Babel.script_blocks['\bbl@cl{sbc}'] then
2408     Babel.loc_to_scr[\the\localeid] =
2409       Babel.script_blocks['\bbl@cl{sbc}']
2410     Babel.locale_props[\the\localeid].lc = \the\localeid\space
2411     Babel.locale_props[\the\localeid].lg = \the\nameuse{l\language}\space
2412   end
2413 }%
2414 \fi
2415 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2416 \ifin@
2417 \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}}}%
2418 \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}}}%
2419 \directlua{
2420   if Babel.script_blocks['\bbl@cl{sbc}'] then
2421     Babel.loc_to_scr[\the\localeid] =
2422       Babel.script_blocks['\bbl@cl{sbc}']
2423   end}%
2424 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2425 \AtBeginDocument{%
2426   \bbl@patchfont{\bbl@mapselect}%
2427   {\selectfont}}%
2428 \def\bbl@mapselect{%
2429   \let\bbl@mapselect\relax
2430   \edef\bbl@prefontid{\fontid\font}}%
2431 \def\bbl@mapdir##1{%
2432   {\def\language{##1}%
2433     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2434     \bbl@switchfont
2435     \directlua{
2436       Babel.locale_props[\the\csname bbl@id@##1\endcsname]
2437         [\bbl@prefontid] = \fontid\font\space}}}%
2438 \fi
2439 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir\language}}}%
2440 \fi
2441 % TODO - catch non-valid values
2442 \fi
2443 % == mapfont ==
2444 % For bidi texts, to switch the font based on direction
2445 \ifx\bbl@KVP@mapfont\nil\else
2446 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
2447 {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2448   mapfont. Use 'direction'.%
2449   {See the manual for details.}}}%
2450 \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}}}%
2451 \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}}}%
2452 \ifx\bbl@mapselect\undefined % TODO. See onchar.
2453 \AtBeginDocument{%
2454   \bbl@patchfont{\bbl@mapselect}%
2455   {\selectfont}}%
2456 \def\bbl@mapselect{%

```

```

2457 \let\bbl@mapselect\relax
2458 \edef\bbl@prefontid{\fontid\font}}%
2459 \def\bbl@mapdir##1{%
2460   {\def\language{##1}%
2461     \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2462     \bbl@switchfont
2463     \directlua{Babel.fontmap
2464       [\the\csname bbl@wdir@##1\endcsname]%
2465       [\bbl@prefontid]=\fontid\font}}}%
2466 \fi
2467 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2468 \fi
2469 % == Line breaking: intraspace, intrapenalty ==
2470 % For CJK, East Asian, Southeast Asian, if interspace in ini
2471 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2472   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2473 \fi
2474 \bbl@provide@intraspace
2475 % == Line breaking: CJK quotes ==
2476 \ifcase\bbl@engine\or
2477   \bbl@xin@{/c}{/\bbl@cl{lbrk}}%
2478   \ifin@
2479     \bbl@ifunset{bbl@quote@\language}{}%
2480     {\directlua{
2481       Babel.locale_props[\the\localeid].cjk_quotes = {}
2482       local cs = 'op'
2483       for c in string.utfvalues(
2484         [[\csname bbl@quote@\language\endcsname]]) do
2485         if Babel.cjk_characters[c].c == 'qu' then
2486           Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2487         end
2488         cs = ( cs == 'op') and 'cl' or 'op'
2489       end
2490     }}%
2491   \fi
2492 \fi
2493 % == Line breaking: justification ==
2494 \ifx\bbl@KVP@justification\@nil\else
2495   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2496 \fi
2497 \ifx\bbl@KVP@linebreaking\@nil\else
2498   \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2499   \ifin@
2500     \bbl@csarg\xdef
2501       {lbrk@\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2502   \fi
2503 \fi
2504 \bbl@xin@{/e}{/\bbl@cl{lbrk}}%
2505 \ifin@else\bbl@xin@{/k}{/\bbl@cl{lbrk}}\fi
2506 \ifin@\bbl@arabicjust\fi
2507 % == Line breaking: hyphenate.other.(locale|script) ==
2508 \ifx\bbl@lbkflag\@empty
2509   \bbl@ifunset{bbl@hyotl@\language}{}%
2510   {\bbl@csarg\bbl@replace{hyotl@\language}{ }{,}%
2511     \bbl@startcommands*\@language}%
2512   \bbl@csarg\bbl@foreach{hyotl@\language}{%
2513     \ifcase\bbl@engine
2514       \ifnum##1<257
2515         \SetHyphenMap{BabelLower{##1}{##1}}%

```

```

2516         \fi
2517     \else
2518         \SetHyphenMap{\BabelLower{##1}{##1}}%
2519     \fi}%
2520 \bbl@endcommands}%
2521 \bbl@ifunset{\bbl@hyots@\language\language}{}%
2522 {\bbl@csarg\bbl@replace{hyots@\language\language}{ }{,}}%
2523 \bbl@csarg\bbl@foreach{hyots@\language\language}{%
2524     \ifcase\bbl@engine
2525     \ifnum##1<257
2526         \global\lccode##1=##1\relax
2527     \fi
2528     \else
2529         \global\lccode##1=##1\relax
2530     \fi}}%
2531 \fi
2532 % == Counters: maparabic ==
2533 % Native digits, if provided in ini (TeX level, xe and lua)
2534 \ifcase\bbl@engine\else
2535     \bbl@ifunset{\bbl@dgnat@\language\language}{}%
2536     {\expandafter\ifx\csname\bbl@dgnat@\language\language\endcsname\@empty\else
2537         \expandafter\expandafter\expandafter
2538         \bbl@setdigits\csname\bbl@dgnat@\language\language\endcsname
2539         \ifx\bbl@KVP@maparabic\@nil\else
2540             \ifx\bbl@latinarabic\@undefined
2541                 \expandafter\let\expandafter\@arabic
2542                 \csname\bbl@counter@\language\language\endcsname
2543             \else % ie, if layout=counters, which redefines \@arabic
2544                 \expandafter\let\expandafter\bbl@latinarabic
2545                 \csname\bbl@counter@\language\language\endcsname
2546             \fi
2547         \fi
2548     \fi}%
2549 \fi
2550 % == Counters: mapdigits ==
2551 % Native digits (lua level).
2552 \ifodd\bbl@engine
2553     \ifx\bbl@KVP@mapdigits\@nil\else
2554         \bbl@ifunset{\bbl@dgnat@\language\language}{}%
2555         {\RequirePackage{luatexbase}%
2556         \bbl@activate@preotf
2557         \directlua{
2558             Babel = Babel or {} %%% -> presets in luababel
2559             Babel.digits_mapped = true
2560             Babel.digits = Babel.digits or {}
2561             Babel.digits[\the\localeid] =
2562                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2563             if not Babel.numbers then
2564                 function Babel.numbers(head)
2565                     local LOCALE = Babel.attr_locale
2566                     local GLYPH = node.id'glyph'
2567                     local inmath = false
2568                     for item in node.traverse(head) do
2569                         if not inmath and item.id == GLYPH then
2570                             local temp = node.get_attribute(item, LOCALE)
2571                             if Babel.digits[temp] then
2572                                 local chr = item.char
2573                                 if chr > 47 and chr < 58 then
2574                                     item.char = Babel.digits[temp][chr-47]

```

```

2575             end
2576         end
2577         elseif item.id == node.id'math' then
2578             inmath = (item.subtype == 0)
2579         end
2580     end
2581     return head
2582 end
2583 end
2584 }}%
2585 \fi
2586 \fi
2587 % == Counters: alph, Alph ==
2588 % What if extras<lang> contains a \babel@save\@alph? It won't be
2589 % restored correctly when exiting the language, so we ignore
2590 % this change with the \bbl@alph@saved trick.
2591 \ifx\bbl@KVP@alph\@nil\else
2592     \bbl@extras@wrap{\bbl@alph@saved}%
2593     {\let\bbl@alph@saved\@alph}%
2594     {\let\@alph\bbl@alph@saved
2595     \babel@save\@alph}%
2596     \bbl@exp{%
2597         \bbl@add\<extras\language>{%
2598             \let\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
2599 \fi
2600 \ifx\bbl@KVP@Alph\@nil\else
2601     \bbl@extras@wrap{\bbl@Alph@saved}%
2602     {\let\bbl@Alph@saved\@Alph}%
2603     {\let\@Alph\bbl@Alph@saved
2604     \babel@save\@Alph}%
2605     \bbl@exp{%
2606         \bbl@add\<extras\language>{%
2607             \let\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
2608 \fi
2609 % == require.babel in ini ==
2610 % To load or reload the babel-*.tex, if require.babel in ini
2611 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2612     \bbl@ifunset{\bbl@rtex@\language}{}%
2613     {\expandafter\ifx\csname bbl@rtex@\language\endcsname\@empty\else
2614         \let\BabelBeforeIni\@gobbletwo
2615         \chardef\atcatcode=\catcode`\@
2616         \catcode`\@=11\relax
2617         \bbl@input@texini{\bbl@cs{rtex@\language}}%
2618         \catcode`\@=\atcatcode
2619         \let\atcatcode\relax
2620         \global\bbl@csarg\let{rtex@\language}\relax
2621     \fi}%
2622 \fi
2623 % == frenchspacing ==
2624 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2625 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2626 \ifin@
2627     \bbl@extras@wrap{\bbl@pre@fs}%
2628     {\bbl@pre@fs}%
2629     {\bbl@post@fs}%
2630 \fi
2631 % == Release saved transforms ==
2632 \bbl@release@transforms\relax % \relax closes the last item.
2633 % == main ==

```

```

2634 \ifx\bbbl@KVP@main\@nil % Restore only if not 'main'
2635 \let\language\bbbl@savelangname
2636 \chardef\localeid\bbbl@savelocaleid\relax
2637 \fi}

```

Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbbl@startcommands opens a group.

```

2638 \def\bbbl@provide@new#1{%
2639 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2640 \namedef{extras#1}{}%
2641 \namedef{noextras#1}{}%
2642 \bbbl@startcommands*{#1}{captions}%
2643 \ifx\bbbl@KVP@captions\@nil % and also if import, implicit
2644 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2645 \ifx##1\@empty\else
2646 \bbbl@exp{%
2647 \SetString\##1{%
2648 \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2649 \expandafter\bbbl@tempb
2650 \fi}%
2651 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2652 \else
2653 \ifx\bbbl@initoload\relax
2654 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2655 \else
2656 \bbbl@read@ini{\bbbl@initoload}2% % Same
2657 \fi
2658 \fi
2659 \StartBabelCommands*{#1}{date}%
2660 \ifx\bbbl@KVP@import\@nil
2661 \bbbl@exp{%
2662 \SetString\today{\bbbl@nocaption{today}{#1today}}}%
2663 \else
2664 \bbbl@savetoday
2665 \bbbl@savedate
2666 \fi
2667 \bbbl@endcommands
2668 \bbbl@load@basic{#1}%
2669 % == hyphenmins == (only if new)
2670 \bbbl@exp{%
2671 \gdef\<#1hyphenmins>{%
2672 {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2673 {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}%
2674 % == hyphenrules (also in renew) ==
2675 \bbbl@provide@hyphens{#1}%
2676 \ifx\bbbl@KVP@main\@nil\else
2677 \expandafter\main@language\expandafter{#1}%
2678 \fi}
2679 %
2680 \def\bbbl@provide@renew#1{%
2681 \ifx\bbbl@KVP@captions\@nil\else
2682 \StartBabelCommands*{#1}{captions}%
2683 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2684 \EndBabelCommands
2685 \fi
2686 \ifx\bbbl@KVP@import\@nil\else
2687 \StartBabelCommands*{#1}{date}%
2688 \bbbl@savetoday
2689 \bbbl@savedate

```

```

2690 \EndBabelCommands
2691 \fi
2692 % == hyphenrules (also in new) ==
2693 \ifx\bbbl@lbkflag\@empty
2694 \bbbl@provide@hyphens{#1}%
2695 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

```

2696 \def\bbbl@load@basic#1{%
2697 \ifcase\bbbl@howloaded\or\or
2698 \ifcase\csname bbl@llevel@\language\endcsname
2699 \bbbl@csarg\let\lname@\language\relax
2700 \fi
2701 \fi
2702 \bbbl@ifunset{bbl@lname@#1}%
2703 {\def\BabelBeforeIni##1##2{%
2704 \begingroup
2705 \let\bbbl@ini@captions@aux\@gobbletwo
2706 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2707 \bbbl@read@ini{##1}1%
2708 \ifx\bbbl@initoload\relax\endinput\fi
2709 \endgroup}%
2710 \begingroup % boxed, to avoid extra spaces:
2711 \ifx\bbbl@initoload\relax
2712 \bbbl@input@texini{#1}%
2713 \else
2714 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
2715 \fi
2716 \endgroup}%
2717 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

2718 \def\bbbl@provide@hyphens#1{%
2719 \let\bbbl@tempa\relax
2720 \ifx\bbbl@KVP@hyphenrules\@nil\else
2721 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
2722 \bbbl@foreach\bbbl@KVP@hyphenrules{%
2723 \ifx\bbbl@tempa\relax % if not yet found
2724 \bbbl@ifsamestring{##1}{+}%
2725 {\bbbl@exp{\addlanguage\<l@##1>}}}%
2726 }%
2727 \bbbl@ifunset{l@##1}%
2728 }%
2729 {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}%
2730 \fi}%
2731 \fi
2732 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
2733 \ifx\bbbl@KVP@import\@nil
2734 \ifx\bbbl@initoload\relax\else
2735 \bbbl@exp{% and hyphenrules is not empty
2736 \bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
2737 }%
2738 {\let\bbbl@tempa\<l@bbbl@cl{hyphr}>}}%
2739 \fi
2740 \else % if importing
2741 \bbbl@exp{% and hyphenrules is not empty
2742 \bbbl@ifblank{\bbbl@cs{hyphr@#1}}%

```

```

2743      {}%
2744      {\let\\bbl@tempa<l@bbl@c1{hyphr}>}}%
2745      \fi
2746      \fi
2747      \bbl@ifunset{bbl@tempa}%          ie, relax or undefined
2748      {\bbl@ifunset{l@#1}%              no hyphenrules found - fallback
2749       {\bbl@exp{\\adddialect<l@#1>\language}}%
2750       {}}%                             so, l@<lang> is ok - nothing to do
2751      {\bbl@exp{\\adddialect<l@#1>bbl@tempa}}}% found in opt list or ini

```

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

```

2752 \def\bbl@input@texini#1{%
2753   \bbl@bsphack
2754   \bbl@exp{%
2755     \catcode`\\%=14 \catcode`\\%=0
2756     \catcode`\\%=1 \catcode`\\%=2
2757     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2758     \catcode`\\%=\the\catcode`\%relax
2759     \catcode`\\%=\the\catcode`\%relax
2760     \catcode`\\%=\the\catcode`\%relax
2761     \catcode`\\%=\the\catcode`\%relax}%
2762   \bbl@esphack}

```

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

```

2763 \def\bbl@inline#1\bbl@inline{%
2764   \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@% ]
2765   \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2766   \def\bbl@iniskip#1\@@{%          if starts with ;
2767   \def\bbl@inistore#1=#2\@@{%      full (default)
2768     \bbl@trim@def\bbl@tempa{#1}%
2769     \bbl@trim\toks@{#2}%
2770     \bbl@xin@{\bbl@section/\bbl@tempa}{\bbl@key@list}%
2771     \ifin@else
2772       \bbl@exp{%
2773         \\g@addto@macro\\bbl@inidata{%
2774           \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2775       \fi}
2776   \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2777     \bbl@trim@def\bbl@tempa{#1}%
2778     \bbl@trim\toks@{#2}%
2779     \bbl@xin@{.identification.}{.\bbl@section.}%
2780     \ifin@
2781       \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2782         \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2783     \fi}

```

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

```

2784 \ifx\bbl@readstream\undefined
2785   \csname newread\endcsname\bbl@readstream
2786 \fi
2787 \def\bbl@read@ini#1#2{%
2788   \global\let\bbl@extend@ini@gobble

```

```

2789 \openin\bbl@readstream=babel-#1.ini
2790 \ifeof\bbl@readstream
2791   \bbl@error
2792     {There is no ini file for the requested language\\%
2793       (#1). Perhaps you misspelled it or your installation\\%
2794       is not complete.}%
2795     {Fix the name or reinstall babel.}%
2796 \else
2797   % == Store ini data in \bbl@inidata ==
2798   \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
2799   \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2800   \bbl@info{Importing
2801     \ifcase#2font and identification \or basic \fi
2802     data for \language\\%
2803     from babel-#1.ini. Reported}%
2804   \ifnum#2=\z@
2805     \global\let\bbl@inidata\@empty
2806     \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2807   \fi
2808   \def\bbl@section{identification}%
2809   \bbl@exp{\ \bbl@inistore tag.ini=#1\ \ \ \ @}%
2810   \bbl@inistore load.level=#2\ \ \ @
2811   \loop
2812   \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2813     \newlinechar\m@ne
2814     \read\bbl@readstream to \bbl@line
2815     \newlinechar\^^M
2816     \ifx\bbl@line\@empty\else
2817       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2818     \fi
2819   \repeat
2820   % == Process stored data ==
2821   \bbl@csarg\xdef{lini@\language}{#1}%
2822   \bbl@read@ini@aux
2823   % == 'Export' data ==
2824   \bbl@ini@exports{#2}%
2825   \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2826   \global\let\bbl@inidata\@empty
2827   \bbl@exp{\ \bbl@add@list\ \bbl@ini@loaded{\language}}%
2828   \bbl@tglobal\bbl@ini@loaded
2829   \fi}
2830 \def\bbl@read@ini@aux{%
2831   \let\bbl@savestrings\@empty
2832   \let\bbl@savetoday\@empty
2833   \let\bbl@savestate\@empty
2834   \def\bbl@elt##1##2##3{%
2835     \def\bbl@section{##1}%
2836     \in@{=date.}{=##1}% Find a better place
2837     \ifin@
2838       \bbl@ini@calendar{##1}%
2839     \fi
2840     \bbl@ifunset{bbl@inikv@##1}{}%
2841     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2842   \bbl@inidata}

A variant to be used when the ini file has been already loaded, because it's not the first
\babelprovide for this language.

2843 \def\bbl@extend@ini@aux#1{%
2844   \bbl@startcommands*{#1}{captions}%

```



```

2845 % Activate captions/... and modify exports
2846 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2847   \setlocalecaption{#1}{##1}{##2}}%
2848 \def\bbl@inikv@captions##1##2{%
2849   \bbl@ini@captions@aux{##1}{##2}}%
2850 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2851 \def\bbl@exportkey##1##2##3{%
2852   \bbl@ifunset{bbl@kv@##2}{}%
2853     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2854       \bbl@exp{\global\let\<bbl@##1@language\>\<bbl@kv@##2>}}%
2855     \fi}}%
2856 % As with \bbl@read@ini, but with some changes
2857 \bbl@read@ini@aux
2858 \bbl@ini@exports\tw@
2859 % Update inidata@lang by pretending the ini is read.
2860 \def\bbl@elt##1##2##3{%
2861   \def\bbl@section{##1}%
2862   \bbl@iniline##2=##3\bbl@iniline}%
2863   \csname bbl@inidata@#1\endcsname
2864   \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2865 \StartBabelCommands*{#1}{date}% And from the import stuff
2866 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2867 \bbl@savetoday
2868 \bbl@savestate
2869 \bbl@endcommands}

```

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

```

2870 \def\bbl@ini@calendar#1{%
2871   \lowercase{\def\bbl@tempa{= #1=}}%
2872   \bbl@replace\bbl@tempa{=date.gregorian}{}}%
2873   \bbl@replace\bbl@tempa{=date.}{}}%
2874   \in@{.licr=}{#1=}%
2875   \ifin@
2876     \ifcase\bbl@engine
2877       \bbl@replace\bbl@tempa{.licr=}{}}%
2878     \else
2879       \let\bbl@tempa\relax
2880     \fi
2881   \fi
2882   \ifx\bbl@tempa\relax\else
2883     \bbl@replace\bbl@tempa{=}{}}%
2884     \bbl@exp{%
2885       \def\<bbl@inikv@#1>####1####2{%
2886         \bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
2887     \fi}

```

A key with a slash in \babelprovide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in \bbl@inistore above).

```

2888 \def\bbl@renewinikey#1/#2\@#3{%
2889   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2890   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2891   \bbl@trim\toks@{#3}% value
2892   \bbl@exp{%
2893     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2894     \\g@addto@macro\\bbl@inidata{%
2895       \\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

2896 \def\bbl@exportkey#1#2#3{%
2897   \bbl@ifunset{bbl@kv@#2}%
2898     {\bbl@csarg\gdef{#1@language}{#3}}%
2899     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2900       \bbl@csarg\gdef{#1@language}{#3}}%
2901     \else
2902       \bbl@exp{\global\let\<bbl@#1@language>\<bbl@kv@#2>}%
2903       \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

```

2904 \def\bbl@iniwarning#1{%
2905   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2906   {\bbl@warning{%
2907     From babel-\bbl@cs{lini@language}.ini:\%
2908     \bbl@cs{kv@identification.warning#1}\%
2909     Reported }}%
2910 %
2911 \let\bbl@release@transforms\@empty
2912 %
2913 \def\bbl@ini@exports#1{%
2914   % Identification always exported
2915   \bbl@iniwarning{%
2916     \ifcase\bbl@engine
2917       \bbl@iniwarning{.pdflatex}%
2918     \or
2919       \bbl@iniwarning{.lualatex}%
2920     \or
2921       \bbl@iniwarning{.xelatex}%
2922     \fi%
2923     \bbl@exportkey{llevel}{identification.load.level}{}%
2924     \bbl@exportkey{elname}{identification.name.english}{}%
2925     \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2926       {\csname bbl@elname@language\endcsname}}%
2927     \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2928     \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2929     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2930     \bbl@exportkey{esname}{identification.script.name}{}%
2931     \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2932       {\csname bbl@esname@language\endcsname}}%
2933     \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2934     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2935     % Also maps bcp47 -> language
2936     \ifbbl@bcptoname
2937       \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{\language}%
2938     \fi
2939     % Conditional
2940     \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
2941       \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2942       \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2943       \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2944       \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2945       \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2946       \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2947       \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2948       \bbl@exportkey{intsp}{typography.intraspace}{}%

```

```

2949 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2950 \bbl@exportkey{chrng}{characters.ranges}{}%
2951 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2952 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2953 \ifnum#1=\tw@ % only (re)new
2954 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2955 \bbl@toglobal\bbl@savetoday
2956 \bbl@toglobal\bbl@savestate
2957 \bbl@savestrings
2958 \fi
2959 \fi}

```

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

```

2960 \def\bbl@inikv#1#2{%      key=value
2961 \toks@{#2}%              This hides #'s from ini values
2962 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2963 \let\bbl@inikv@identification\bbl@inikv
2964 \let\bbl@inikv@typography\bbl@inikv
2965 \let\bbl@inikv@characters\bbl@inikv
2966 \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’.

```

2967 \def\bbl@inikv@counters#1#2{%
2968 \bbl@ifsamestring{#1}{digits}%
2969 {\bbl@error{The counter name 'digits' is reserved for mapping\\%
2970 decimal digits}%
2971 {Use another name.}}%
2972 }%
2973 \def\bbl@tempc{#1}%
2974 \bbl@trim@def{\bbl@tempb*}{#2}%
2975 \in@{.1$}{#1$}%
2976 \ifin@
2977 \bbl@replace\bbl@tempc{.1}{}%
2978 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
2979 \noexpand\bbl@alphanumeric{\bbl@tempc}%
2980 \fi
2981 \in@{.F.}{#1}%
2982 \ifin@else\in@{.S.}{#1}\fi
2983 \ifin@
2984 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
2985 \else
2986 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2987 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2988 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
2989 \fi}

```

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

```

2990 \ifcase\bbl@engine
2991 \bbl@csarg\def{inikv@captions.licr}#1#2{%
2992 \bbl@ini@captions@aux{#1}{#2}}
2993 \else
2994 \def\bbl@inikv@captions#1#2{%
2995 \bbl@ini@captions@aux{#1}{#2}}
2996 \fi

```

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

```

2997 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2998   \bbl@replace\bbl@tempa{.template}{}%
2999   \def\bbl@toreplace{#1}{}%
3000   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3001   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3002   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3003   \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}%
3004   \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3005   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3006   \ifin@
3007     \@nameuse{\bbl@patch\bbl@tempa}%
3008     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3009   \fi
3010   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3011   \ifin@
3012     \toks@\expandafter{\bbl@toreplace}%
3013     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3014   \fi}
3015 \def\bbl@ini@captions@aux#1#2{%
3016   \bbl@trim@def\bbl@tempa{#1}%
3017   \bbl@xin@{.template}{\bbl@tempa}%
3018   \ifin@
3019     \bbl@ini@captions@template{#2}\languagename
3020   \else
3021     \bbl@ifblank{#2}%
3022     {\bbl@exp{%
3023       \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3024     {\bbl@trim\toks@{#2}}%
3025     \bbl@exp{%
3026       \bbl@add\bbl@savestrings{%
3027         \SetString\<\bbl@tempa name>{\the\toks@}}%
3028       \toks@\expandafter{\bbl@captionslist}%
3029       \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3030       \ifin@
3031         \bbl@exp{%
3032           \bbl@add\<\bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3033           \bbl@toglobal\<\bbl@extracaps@\languagename>}%
3034       \fi
3035     \fi}

```

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

```

3036 \def\bbl@list@the{%
3037   part,chapter,section,subsection,subsubsection,paragraph,%
3038   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3039   table,page,footnote,mpfootnote,mpfn}
3040 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3041   \bbl@ifunset{\bbl@map@#1@\languagename}%
3042   {\@nameuse{#1}}%
3043   {\@nameuse{\bbl@map@#1@\languagename}}}
3044 \def\bbl@inikv@labels#1#2{%
3045   \in@{.map}{#1}%
3046   \ifin@
3047     \ifx\bbl@KVP@labels\@nil\else
3048       \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3049     \ifin@
3050       \def\bbl@tempc{#1}%
3051       \bbl@replace\bbl@tempc{.map}{}%
3052       \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%

```

```

3053 \bbl@exp{%
3054 \gdef\<bbl@map@bbl@tempc @\languagename>%
3055 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3056 \bbl@foreach\bbl@list@the{%
3057 \bbl@ifunset{the##1}{}%
3058 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3059 \bbl@exp{%
3060 \\bbl@sreplace\<the##1>%
3061 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3062 \\bbl@sreplace\<the##1>%
3063 {\<\@empty @\bbl@tempc>\<c##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3064 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3065 \toks\expandafter\expandafter\expandafter{%
3066 \csname the##1\endcsname}%
3067 \expandafter\def\csname the##1\endcsname{\the\toks@}}%
3068 \fi}}%
3069 \fi
3070 \fi
3071 %
3072 \else
3073 %
3074 % The following code is still under study. You can test it and make
3075 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3076 % language dependent.
3077 \in@{enumerate.}{#1}%
3078 \ifin@
3079 \def\bbl@tempa{#1}%
3080 \bbl@replace\bbl@tempa{enumerate.}{}%
3081 \def\bbl@toreplace{#2}%
3082 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3083 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3084 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}}%
3085 \toks\expandafter{\bbl@toreplace}%
3086 % TODO. Execute only once:
3087 \bbl@exp{%
3088 \\bbl@add\<extras\languagename>{%
3089 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3090 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3091 \\bbl@toggle\<extras\languagename>}%
3092 \fi
3093 \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.

```

3094 \def\bbl@chapttype{chapter}
3095 \ifx\@makechapterhead\@undefined
3096 \let\bbl@patchchapter\relax
3097 \else\ifx\thechapter\@undefined
3098 \let\bbl@patchchapter\relax
3099 \else\ifx\ps@headings\@undefined
3100 \let\bbl@patchchapter\relax
3101 \else
3102 \def\bbl@patchchapter{%
3103 \global\let\bbl@patchchapter\relax
3104 \gdef\bbl@chfmt{%
3105 \bbl@ifunset{bbl@bbl@chapttype fmt@\languagename}%
3106 {\@chapapp\space\thechapter}

```

```

3107     {\nameuse{bbl@bbl@chapttype fmt@language}}
3108     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3109     \bbl@sreplace\ps@headings{\@chapapp\ thechapter}{\bbl@chfmt}%
3110     \bbl@sreplace\chaptermark{\@chapapp\ thechapter}{\bbl@chfmt}%
3111     \bbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3112     \bbl@toglobal\appendix
3113     \bbl@toglobal\ps@headings
3114     \bbl@toglobal\chaptermark
3115     \bbl@toglobal\makechapterhead}
3116 \let\bbl@patchappendix\bbl@patchchapter
3117 \fi\fi\fi
3118 \ifx\@part\@undefined
3119 \let\bbl@patchpart\relax
3120 \else
3121 \def\bbl@patchpart{%
3122 \global\let\bbl@patchpart\relax
3123 \gdef\bbl@partformat{%
3124 \bbl@ifunset{bbl@partfmt@language}%
3125 {\partname\nobreakspace\thepart}
3126 {\nameuse{bbl@partfmt@language}}}
3127 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3128 \bbl@toglobal\@part}
3129 \fi

```

Date. TODO. Document

```

3130 % Arguments are _not_ protected.
3131 \let\bbl@calendar\@empty
3132 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3133 \def\bbl@localedate#1#2#3#4{%
3134 \begingroup
3135 \ifx\@empty#1\@empty\else
3136 \let\bbl@ld@calendar\@empty
3137 \let\bbl@ld@variant\@empty
3138 \edef\bbl@tempa{\zap@space#1 \@empty}%
3139 \def\bbl@tempb##1=##2@@{\@namedef{bbl@ld@##1}{##2}}%
3140 \bbl@foreach\bbl@tempa{\bbl@tempb##1@@}%
3141 \edef\bbl@calendar{%
3142 \bbl@ld@calendar
3143 \ifx\bbl@ld@variant\@empty\else
3144 .\bbl@ld@variant
3145 \fi}%
3146 \bbl@replace\bbl@calendar{gregorian}{}}%
3147 \fi
3148 \bbl@cased
3149 {\nameuse{bbl@date@language @\bbl@calendar}{#2}{#3}{#4}}%
3150 \endgroup}
3151 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3152 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3153 \bbl@trim@def\bbl@tempa{#1.#2}%
3154 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3155 {\bbl@trim@def\bbl@tempa{#3}%
3156 \bbl@trim\toks@{#5}%
3157 \@temptokena\expandafter{\bbl@savedate}%
3158 \bbl@exp{% Reverse order - in ini last wins
3159 \def\\bbl@savedate{%
3160 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3161 \the\@temptokena}}}%
3162 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3163 {\lowercase{\def\bbl@tempb{#6}}}%

```

```

3164 \bbl@trim@def\bbl@toreplace{#5}%
3165 \bbl@TG@date
3166 \bbl@ifunset\bbl@date@\language @}%
3167 {\bbl@exp{% TODO. Move to a better place.
3168 \gdef\<\language date>{\protect\<\language date >}%
3169 \gdef\<\language date >####1####2####3{%
3170 \bbl@usedategroupttrue
3171 \<bbl@ensure@\language >{%
3172 \bbl@localedate{####1}{####2}{####3}}}%
3173 \bbl@add\bbl@savetoday{%
3174 \SetString\today{%
3175 \<\language date>%
3176 {\the\year}{\the\month}{\the\day}}}%
3177 }%
3178 \global\bbl@csarg\let{date@\language @}\bbl@toreplace
3179 \ifx\bbl@tempb\empty\else
3180 \global\bbl@csarg\let{date@\language @}\bbl@tempb\bbl@toreplace
3181 \fi}%
3182 {}%

```

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. Note after \bbl@replace \toks@ contains the resulting string, which is used by \bbl@replace@finish@iii (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3183 \let\bbl@calendar\empty
3184 \newcommand\BabelDateSpace{\nobreakspace}
3185 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3186 \newcommand\BabelDated[1]{\number#1}
3187 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3188 \newcommand\BabelDateM[1]{\number#1}
3189 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3190 \newcommand\BabelDateMMM[1]{\number#1}
3191 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3192 \newcommand\BabelDatey[1]{\number#1}%
3193 \newcommand\BabelDateyy[1]{%
3194 \ifnum#1<10 0\number#1 %
3195 \else\ifnum#1<100 \number#1 %
3196 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3197 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3198 \else
3199 \bbl@error
3200 {Currently two-digit years are restricted to the\
3201 range 0-9999.}%
3202 {There is little you can do. Sorry.}%
3203 \fi\fi\fi\fi}%
3204 \newcommand\BabelDateyyy[1]{\number#1} % TODO - add leading 0
3205 \def\bbl@replace@finish@iii#1{%
3206 \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
3207 \def\bbl@TG@date{%
3208 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3209 \bbl@replace\bbl@toreplace{.}{\BabelDateDot}}%
3210 \bbl@replace\bbl@toreplace{d}{\BabelDated{####3}}%
3211 \bbl@replace\bbl@toreplace{dd}{\BabelDatedd{####3}}%
3212 \bbl@replace\bbl@toreplace{M}{\BabelDateM{####2}}%
3213 \bbl@replace\bbl@toreplace{MM}{\BabelDateMM{####2}}%
3214 \bbl@replace\bbl@toreplace{MMM}{\BabelDateMMM{####2}}%
3215 \bbl@replace\bbl@toreplace{y}{\BabelDatey{####1}}%
3216 \bbl@replace\bbl@toreplace{yy}{\BabelDateyy{####1}}%

```

```

3217 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3218 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3219 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3220 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3221 \bbl@replace@finish@iii\bbl@toreplace}
3222 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3223 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3224 \let\bbl@release@transforms\@empty
3225 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3226 \bbl@transforms\babelprehyphenation}
3227 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3228 \bbl@transforms\babelposthyphenation}
3229 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3230 \begingroup % A hack. TODO. Don't require an specific order
3231 \catcode`\%=12
3232 \catcode`\&=14
3233 \gdef\bbl@transforms#1#2#3{&%
3234 \ifx\bbl@KVP@transforms\@nil\else
3235 \directlua{
3236 str = [==[#2]==]
3237 str = str:gsub('%.%d+%.%d+$', '')
3238 tex.print([[ \def\string\babeltempa{]] .. str .. [[]]])
3239 }&%
3240 \bbl@xin@{, \babeltempa,}{, \bbl@KVP@transforms,}&%
3241 \fin@
3242 \in@{.0$}{#2$}&%
3243 \fin@
3244 \g@addto@macro\bbl@release@transforms{&%
3245 \relax\bbl@transforms@aux#1{\language}\{#3}&%
3246 \else
3247 \g@addto@macro\bbl@release@transforms{, {#3}&%
3248 \fi
3249 \fi
3250 \fi}
3251 \endgroup

```

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

```

3252 \def\bbl@provide@lsys#1{%
3253 \bbl@ifunset{bbl@lname@#1}%
3254 {\bbl@load@info{#1}}%
3255 }%
3256 \bbl@csarg\let{lsys@#1}\@empty
3257 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3258 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3259 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3260 \bbl@ifunset{bbl@lname@#1}{}%
3261 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3262 \ifcase\bbl@engine\or\or
3263 \bbl@ifunset{bbl@prehc@#1}{}%
3264 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3265 }%
3266 {\ifx\bbl@xenohyph\@undefined
3267 \let\bbl@xenohyph\bbl@xenohyph@d
3268 \ifx\AtBeginDocument\@notprerr
3269 \expandafter\@secondoftwo % to execute right now
3270 \fi

```



```

3271      \AtBeginDocument{%
3272      \bbl@patchfont{\bbl@xenoxyph}%
3273      \expandafter\selectlanguage\expandafter{\language}%
3274      \fi}%
3275  \fi
3276  \bbl@csarg\bbl@tglobal{lsys@#1}%
3277  \def\bbl@xenoxyph@{
3278    \bbl@ifset{\bbl@prehc@language}%
3279    {\ifnum\hyphenchar\font=\defaultshphenchar
3280     \iffontchar\font\bbl@c{prehc}\relax
3281     \hyphenchar\font\bbl@c{prehc}\relax
3282     \else\iffontchar\font"200B
3283     \hyphenchar\font"200B
3284     \else
3285     \bbl@warning
3286     {Neither 0 nor ZERO WIDTH SPACE are available\\%
3287     in the current font, and therefore the hyphen\\%
3288     will be printed. Try changing the fontspec's\\%
3289     'HyphenChar' to another value, but be aware\\%
3290     this setting is not safe (see the manual)}%
3291     \hyphenchar\font\defaultshphenchar
3292    \fi\fi
3293    \fi}%
3294    {\hyphenchar\font\defaultshphenchar}}
3295    % \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).

```

3296  \def\bbl@load@info#1{%
3297    \def\BabelBeforeIni##1##2{%
3298      \begingroup
3299      \bbl@read@ini{##1}0%
3300      \endinput          % babel- .tex may contain only preamble's
3301      \endgroup}%        boxed, to avoid extra spaces:
3302    {\bbl@input@texini{#1}}}

```

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

```

3303  \def\bbl@setdigits#1#2#3#4#5{%
3304    \bbl@exp{%
3305      \def<\language digits>####1%      ie, \langdigits
3306      \<bbl@digits@language>####1\\\nil}%
3307      \let<bbl@cntr@digits@language>\<\language digits>%
3308      \def<\language counter>####1%      ie, \langcounter
3309      \\\expandafter\<bbl@counter@language>%
3310      \\\csname c#####1\endcsname}%
3311      \def<bbl@counter@language>####1% ie, \bbl@counter@lang
3312      \\\expandafter\<bbl@digits@language>%
3313      \\\number####1\\\nil}}%
3314  \def\bbl@tempa##1##2##3##4##5{%
3315    \bbl@exp{%      Wow, quite a lot of hashes! :-(
3316      \def<bbl@digits@language>#####1%
3317      \\\ifx#####1\\\nil          % ie, \bbl@digits@lang
3318      \\\else
3319      \\\ifx0#####1#1%
3320      \\\else\\\ifx1#####1#2%

```

[illegible]

```

3334 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3335   \ifx\\#1%                % \\ before, in case #1 is multiletter
3336     \bbl@exp{%
3337       \def\\ \bbl@tempa####1{%
3338         \<ifcase>####1\space\the\toks@\<else>\\ \ctrerr\<fi>}}%
3339     \else
3340       \toks@\expandafter{\the\toks@\or #1}%
3341       \expandafter\bbl@buildifcase
3342   \fi}

```

```

3343 \newcommand\locallenumerical[2]{\bbl@cs{cntr@#1\languageaname}{#2}}
3344 \def\bbl@localecntr#1#2{\locallenumerical{#2}{#1}}
3345 \newcommand\localecounter[2]{%
3346   \expandafter\bbl@localecntr
3347   \expandafter{\number\csname c@#2\endcsname}{#1}}
3348 \def\bbl@alphnumerical#1#2{%
3349   \expandafter\bbl@alphnumerical@i\number#2 76543210@@{#1}}
3350 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8@@@#9{%
3351   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
3352     \bbl@alphnumerical@ii{#9}000000#1\or
3353     \bbl@alphnumerical@ii{#9}00000#1#2\or
3354     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3355     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3356     \bbl@alphnum@invalid{>9999}%
3357   \fi}
3358 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
3359   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languageaname}%
3360     {\bbl@cs{cntr@#1.4@\languageaname}#5%
3361     \bbl@cs{cntr@#1.3@\languageaname}#6%
3362     \bbl@cs{cntr@#1.2@\languageaname}#7%
3363     \bbl@cs{cntr@#1.1@\languageaname}#8%
3364     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3365       \bbl@ifunset{bbl@cntr@#1.S.321@\languageaname}{}%
3366       {\bbl@cs{cntr@#1.S.321@\languageaname}}%
3367     \fi}%
3368   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languageaname}}}
3369 \def\bbl@alphnum@invalid#1{%
3370   \bbl@error{Alphabetic numeral too large (#1)}%
3371   {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.

```

3372 \newcommand\localeinfo[1]{%
3373   \bbl@ifunset{\bbl@csname bbl@info@#1\endcsname @\languagename}%
3374   {\bbl@error{I've found no info for the current locale.\%
3375             The corresponding ini file has not been loaded\%
3376             Perhaps it doesn't exist}%
3377   {See the manual for details.}}%
3378   {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}%
3379 % \@namedef{\bbl@info@name.locale}{lcname}
3380 \@namedef{\bbl@info@tag.ini}{lini}
3381 \@namedef{\bbl@info@name.english}{elname}
3382 \@namedef{\bbl@info@name.opentype}{lname}
3383 \@namedef{\bbl@info@tag.bcp47}{tbc}
3384 \@namedef{\bbl@info@language.tag.bcp47}{lbc}
3385 \@namedef{\bbl@info@tag.opentype}{lotf}
3386 \@namedef{\bbl@info@script.name}{esname}
3387 \@namedef{\bbl@info@script.name.opentype}{sname}
3388 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
3389 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3390 \let\bbl@ensureinfo\@gobble
3391 \newcommand\BabelEnsureInfo{%
3392   \ifx\InputIfFileExists\undefined\else
3393     \def\bbl@ensureinfo##1{%
3394       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3395   \fi
3396   \bbl@foreach\bbl@loaded{%
3397     \def\languagename{##1}%
3398     \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`.

```

3399 \newcommand\getlocaleproperty{%
3400   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3401 \def\bbl@getproperty@s#1#2#3{%
3402   \let#1\relax
3403   \def\bbl@elt##1##2##3{%
3404     \bbl@ifsamestring{##1/##2}{##3}%
3405     {\providecommand#1{##3}%
3406     \def\bbl@elt####1####2####3{}}}%
3407   {}}%
3408   \bbl@cs{inidata@#2}}%
3409 \def\bbl@getproperty@x#1#2#3{%
3410   \bbl@getproperty@s{#1}{#2}{#3}%
3411   \ifx#1\relax
3412     \bbl@error
3413     {Unknown key for locale '#2':\%
3414     #3\%
3415     \string#1 will be set to \relax}%
3416     {Perhaps you misspelled it.}%
3417   \fi}
3418 \let\bbl@ini@loaded\@empty
3419 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

9 Adjusting the Babel bahavior

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

```

3420 \newcommand\babeladjust[1]{% TODO. Error handling.
3421   \bbl@forkv{#1}{%
3422     \bbl@ifunset{bbl@ADJ@##1@##2}%
3423     {\bbl@cs{ADJ@##1}{##2}}%
3424     {\bbl@cs{ADJ@##1@##2}}}
3425 %
3426 \def\bbl@adjust@lua#1#2{%
3427   \ifvmode
3428     \ifnum\currentgrouplevel=\z@
3429       \directlua{ Babel.#2 }%
3430       \expandafter\expandafter\expandafter\@gobble
3431     \fi
3432   \fi
3433   {\bbl@error % The error is gobbled if everything went ok.
3434     {Currently, #1 related features can be adjusted only\\%
3435       in the main vertical list.}%
3436     {Maybe things change in the future, but this is what it is.}}}
3437 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3438   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3439 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3440   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3441 \@namedef{bbl@ADJ@bidi.text@on}{%
3442   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3443 \@namedef{bbl@ADJ@bidi.text@off}{%
3444   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3445 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3446   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3447 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3448   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3449 %
3450 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3451   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3452 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3453   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3454 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3455   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3456 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3457   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3458 \@namedef{bbl@ADJ@justify.arabic@on}{%
3459   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3460 \@namedef{bbl@ADJ@justify.arabic@off}{%
3461   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3462 %
3463 \def\bbl@adjust@layout#1{%
3464   \ifvmode
3465     #1%
3466     \expandafter\@gobble
3467   \fi
3468   {\bbl@error % The error is gobbled if everything went ok.
3469     {Currently, layout related features can be adjusted only\\%
3470       in vertical mode.}%
3471     {Maybe things change in the future, but this is what it is.}}}
3472 \@namedef{bbl@ADJ@layout.tabular@on}{%
3473   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3474 \@namedef{bbl@ADJ@layout.tabular@off}{%
3475   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3476 \@namedef{bbl@ADJ@layout.lists@on}{%
3477   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3478 \@namedef{bbl@ADJ@layout.lists@off}{%

```

```

3479 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3480 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3481 \bbl@activateposthyphen}
3482 %
3483 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3484 \bbl@bcpallowedtrue}
3485 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3486 \bbl@bcpallowedfalse}
3487 \@namedef{bbl@ADJ@autoload.bcp47.prefix}{#1}%
3488 \def\bbl@bcp@prefix{#1}}
3489 \def\bbl@bcp@prefix{bcp47-}
3490 \@namedef{bbl@ADJ@autoload.options}{#1}%
3491 \def\bbl@autoload@options{#1}}
3492 \let\bbl@autoload@bcptoptions\@empty
3493 \@namedef{bbl@ADJ@autoload.bcp47.options}{#1}%
3494 \def\bbl@autoload@bcptoptions{#1}}
3495 \newif\ifbbl@bcptoname
3496 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3497 \bbl@bcptonametrue
3498 \BabelEnsureInfo}
3499 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3500 \bbl@bcptonamefalse}
3501 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3502 \directlua{ Babel.ignore_pre_char = function(node)
3503     return (node.lang == \the\csname l@nohyphenation\endcsname)
3504     end }}
3505 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3506 \directlua{ Babel.ignore_pre_char = function(node)
3507     return false
3508     end }}
3509 \@namedef{bbl@ADJ@select.write@shift}{%
3510 \let\bbl@restorelastskip\relax
3511 \def\bbl@savelastskip{%
3512 \let\bbl@restorelastskip\relax
3513 \ifvmode
3514 \ifdim\lastskip=\z@
3515 \let\bbl@restorelastskip\nobreak
3516 \else
3517 \bbl@exp{%
3518 \def\\bbl@restorelastskip{%
3519 \skip@=\the\lastskip
3520 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3521 \fi
3522 \fi}}
3523 \@namedef{bbl@ADJ@select.write@keep}{%
3524 \let\bbl@restorelastskip\relax
3525 \let\bbl@savelastskip\relax}

```

As the final task, load the code for lua. TODO: use babel name, override

```

3526 \ifx\directlua\@undefined\else
3527 \ifx\bbl@luapatterns\@undefined
3528 \input luababel.def
3529 \fi
3530 \fi

```

Continue with \LaTeX .

```

3531 </package | core>
3532 <*package>

```

9.1 Cross referencing macros

The \TeX book states:

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

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

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

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

```
3533 <<*More package options>> ≡
3534 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3535 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3536 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3537 <</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.

```
3538 \bbl@trace{Cross referencing macros}
3539 \ifx\bbl@opt@safe\@empty\else
3540   \def\@newl@bel#1#2#3{%
3541     {\@safe@activestrue
3542       \bbl@ifunset{#1@#2}%
3543       \relax
3544       {\gdef\@multiplelabels{%
3545         \@latex@warning@no@line{There were multiply-defined labels}}%
3546         \@latex@warning@no@line{Label `#2' multiply defined}}%
3547       \global\@namedef{#1@#2}{#3}}}
```

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

```
3548 \CheckCommand*\@testdef[3]{%
3549   \def\reserved@a{#3}%
3550   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3551   \else
3552     \@tempswatrue
3553   \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.

```
3554 \def\@testdef#1#2#3{% TODO. With @samestring?
3555   \@safe@activestrue
3556   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3557   \def\bbl@tempb{#3}%
3558   \@safe@activesfalse
3559   \ifx\bbl@tempa\relax
3560   \else
3561     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3562   \fi
3563   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3564   \ifx\bbl@tempa\bbl@tempb
3565   \else
3566     \@tempswatrue
```

```

3567     \fi}
3568 \fi

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

3569 \bbl@xin@{R}\bbl@opt@safe
3570 \ifin@
3571   \bbl@redefineroobust\ref#1{%
3572     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3573   \bbl@redefineroobust\pageref#1{%
3574     \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3575 \else
3576   \let\org@ref\ref
3577   \let\org@pageref\pageref
3578 \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.

3579 \bbl@xin@{B}\bbl@opt@safe
3580 \ifin@
3581   \bbl@redefine\@citex[#1]#2{%
3582     \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3583     \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.

3584 \AtBeginDocument{%
3585   \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.)

3586   \def\@citex[#1][#2]#3{%
3587     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3588     \org@@citex[#1][#2]{\@tempa}}%
3589   }{}}

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

3590 \AtBeginDocument{%
3591   \ifpackageloaded{cite}{%
3592     \def\@citex[#1]#2{%
3593       \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3594     }{}}

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

3595 \bbl@redefine\nocite#1{%
3596   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

\bibcite The macro that is used in the .aux file to define citation labels. When packages such as natbib or
         cite are not loaded its second argument is used to typeset the citation label. In that case, this second
         argument can contain active characters but is used in an environment where \@safe@activetrue
         is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order

```

to determine during .aux file processing which definition of \bibtex is needed we define \bibtex in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibtex. This new definition is then activated.

```
3597 \bbl@redefine\bibtex{%
3598   \bbl@cite@choice
3599   \bibtex}
```

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

```
3600 \def\bbl@bibtex#1#2{%
3601   \org@bibtex{#1}{\@safe@activesfalse#2}}
```

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

```
3602 \def\bbl@cite@choice{%
3603   \global\let\bibtex\bbl@bibtex
3604   \@ifpackageloaded{natbib}{\global\let\bibtex\org@bibtex}{}%
3605   \@ifpackageloaded{cite}{\global\let\bibtex\org@bibtex}{}%
3606   \global\let\bbl@cite@choice\relax}
```

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

```
3607 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3608 \bbl@redefine\@bibitem#1{%
3609   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3610 \else
3611   \let\org@nocite\nocite
3612   \let\org@citex\citex
3613   \let\org@bibtex\bibtex
3614   \let\org@bibitem\@bibitem
3615 \fi
```

9.2 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.

```
3616 \bbl@trace{Marks}
3617 \IfBabelLayout{sectioning}
3618   {\ifx\bbl@opt@headfoot\@nnil
3619     \g@addto@macro\resetactivechars{%
3620       \set@typeset@protect
3621       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3622       \let\protect\noexpand
3623       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3624         \edef\thepage{%
3625           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3626       \fi}%
3627   \fi}
3628 {\ifbbl@single\else
3629   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3630     \markright#1{%
```



```

3631      \bbl@ifblank{#1}%
3632      {\org@markright{}}%
3633      {\toks@{#1}}%
3634      \bbl@exp{%
3635          \\org@markright{\\protect\\foreignlanguage{\language}%
3636              {\\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, L^AT_EX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3637      \ifx\@mkboth\markboth
3638          \def\bbl@tempc{\let\@mkboth\markboth}
3639      \else
3640          \def\bbl@tempc{
3641              \fi
3642              \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3643              \markboth#1#2{%
3644                  \protected@edef\bbl@tempb##1{%
3645                      \protect\foreignlanguage
3646                      {\language}{\protect\bbl@restore@actives##1}}%
3647                  \bbl@ifblank{#1}%
3648                  {\toks@{}}%
3649                  {\toks@\expandafter{\bbl@tempb{#1}}}%
3650                  \bbl@ifblank{#2}%
3651                  {\@temptokena{}}%
3652                  {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3653                  \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
3654                  \bbl@tempc
3655              \fi} % end ifbbl@single, end \IfBabelLayout

```

9.3 Preventing clashes with other packages

9.3.1 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.

```

3656 \bbl@trace{Preventing clashes with other packages}
3657 \bbl@xin@{R}\bbl@opt@safe
3658 \ifin@
3659 \AtBeginDocument{%
3660     \@ifpackageloaded{ifthen}{%
3661         \bbl@redefine@long\ifthenelse#1#2#3{%

```

```

3662      \let\bbl@temp@pref\pageref
3663      \let\pageref\org@pageref
3664      \let\bbl@temp@ref\ref
3665      \let\ref\org@ref
3666      \@safe@activetrue
3667      \org@ifthenelse{#1}%
3668        {\let\pageref\bbl@temp@pref
3669         \let\ref\bbl@temp@ref
3670         \@safe@activetrue
3671         #2}%
3672        {\let\pageref\bbl@temp@pref
3673         \let\ref\bbl@temp@ref
3674         \@safe@activetrue
3675         #3}%
3676      }%
3677    }{}%
3678  }

```

9.3.2 varioref

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

```

3679 \AtBeginDocument{%
3680   \@ifpackageloaded{varioref}{%
3681     \bbl@redefine\@vpageref#1[#2]#3{%
3682       \@safe@activetrue
3683       \org@@@vpageref{#1}[#2]{#3}%
3684       \@safe@activetrue}%
3685     \bbl@redefine\vrefpagemum#1#2{%
3686       \@safe@activetrue
3687       \org@vrefpagemum{#1}[#2]%
3688       \@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.

```

3689   \expandafter\def\csname Ref \endcsname#1{%
3690     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3691   }{}%
3692 }
3693 \fi

```

9.3.3 hline

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

```

3694 \AtEndOfPackage{%
3695   \AtBeginDocument{%
3696     \@ifpackageloaded{hhline}%
3697       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3698        \else
3699          \makeatletter

```

```

3700      \def\@currname{hhline}\input{hhline.sty}\makeatother
3701      \fi}%
3702      {}}}

```

`\substitutefontfamily` Deprecated. Use the tools provides by \LaTeX . 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.

```

3703 \def\substitutefontfamily#1#2#3{%
3704   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3705   \immediate\write15{%
3706     \string\ProvidesFile{#1#2.fd}%
3707     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3708     \space generated font description file]^{}
3709     \string\DeclareFontFamily{#1}{#2}{}}^{}
3710     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^{}
3711     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^{}
3712     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^{}
3713     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^{}
3714     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^{}
3715     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^{}
3716     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^{}
3717     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^{}
3718     }%
3719   \closeout15
3720 }
3721 \@onlypreamble\substitutefontfamily

```

9.4 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. Requested encodings are currently stored in `\@fontenc@load@list`. 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`

```

3722 \bbl@trace{Encoding and fonts}
3723 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3724 \newcommand\BabelNonText{TS1,T3,TS3}
3725 \let\org@TeX\TeX
3726 \let\org@LaTeX\LaTeX
3727 \let\ensureascii@firstofone
3728 \AtBeginDocument{%
3729   \def\@elt#1{, #1,}%
3730   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3731   \let\@elt\relax
3732   \let\bbl@tempb\@empty
3733   \def\bbl@tempc{OT1}%
3734   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3735     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3736   \bbl@foreach\bbl@tempa{%
3737     \bbl@xin@{#1}{\BabelNonASCII}%
3738     \ifin@
3739       \def\bbl@tempb{#1}% Store last non-ascii
3740     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3741     \ifin@
3742       \def\bbl@tempc{#1}% Store last ascii
3743     \fi

```

```

3744   \fi}%
3745   \ifx\bb1@tempb\@empty\else
3746     \bb1@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3747     \ifin@\else
3748       \edef\bb1@tempc{\cf@encoding}% The default if ascii wins
3749       \fi
3750       \edef\ensureascii#1{%
3751         {\noexpand\fontencoding{\bb1@tempc}\noexpand\selectfont#1}}%
3752       \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3753       \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3754   \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.

```

3755 \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.

```

3756 \AtBeginDocument{%
3757   \ifpackageloaded{fontspec}%
3758     {\xdef\latinencoding{%
3759       \ifx\UTFfencname\@undefined
3760         EU\ifcase\bb1@engine\or2\or1\fi
3761       \else
3762         \UTFfencname
3763       \fi}}%
3764   {\gdef\latinencoding{OT1}}%
3765   \ifx\cf@encoding\bb1@t@one
3766     \xdef\latinencoding{\bb1@t@one}%
3767   \else
3768     \def\@elt#1{,#1,}%
3769     \edef\bb1@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3770     \let\@elt\relax
3771     \bb1@xin@{,T1,}\bb1@tempa
3772     \ifin@
3773       \xdef\latinencoding{\bb1@t@one}%
3774     \fi
3775   \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.

```

3776 \DeclareRobustCommand{\latintext}{%
3777   \fontencoding{\latinencoding}\selectfont
3778   \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.

```

3779 \ifx\@undefined\DeclareTextFontCommand
3780   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3781 \else
3782   \DeclareTextFontCommand{\textlatin}{\latintext}
3783 \fi

```

For several functions, we need to execute some code with `\selectfont`. With \LaTeX 2021-06-01, there is a hook for this purpose, but in older versions the \LaTeX command is patched (the latter solution will be eventually removed).

```
3784 \bbl@ifformatlater{2021-06-01}%
3785 {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
3786 {\def\bbl@patchfont#1{%
3787     \expandafter\bbl@add\csname selectfont \endcsname{#1}%
3788     \expandafter\bbl@tglobal\csname selectfont \endcsname}}
```

9.5 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.

```
3789 \bbl@trace{Loading basic (internal) bidi support}
3790 \ifodd\bbl@engine
3791 \else % TODO. Move to txtbabel
3792     \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3793         \bbl@error
3794         {The bidi method 'basic' is available only in\\%
3795         luatex. I'll continue with 'bidi=default', so\\%
3796         expect wrong results}%
3797         {See the manual for further details.}%
3798     \let\bbl@beforeforeign\leavevmode
3799     \AtEndOfPackage{%
3800         \EnableBabelHook{babel-bidi}%
3801         \bbl@xebidipar}
3802 \fi\fi
3803 \def\bbl@loadxebidi#1{%
3804     \ifx\RTLfootnotetext\@undefined
3805         \AtEndOfPackage{%
3806             \EnableBabelHook{babel-bidi}%
3807             \ifx\fontspec\@undefined
3808                 \bbl@loadfontspec % bidi needs fontspec
3809             \fi
3810             \usepackage#1{bidi}}%
3811     \fi}
3812 \ifnum\bbl@bidimode>200
3813     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3814         \bbl@tentative{bidi=bidi}
3815         \bbl@loadxebidi{}
```

```

3816 \or
3817 \bbl@loadxebidi{[rldocument]}
3818 \or
3819 \bbl@loadxebidi{}
3820 \fi
3821 \fi
3822 \fi
3823 % TODO? Separate:
3824 \ifnum\bbl@bidimode=\@ne
3825 \let\bbl@beforeforeign\leavevmode
3826 \ifodd\bbl@engine
3827 \newattribute\bbl@attr@dir
3828 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3829 \bbl@exp{\output{\bodydir\pagedir\the\output}}
3830 \fi
3831 \AtEndOfPackage{%
3832 \EnableBabelHook{babel-bidi}%
3833 \ifodd\bbl@engine\else
3834 \bbl@xebidipar
3835 \fi}
3836 \fi

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

3837 \bbl@trace{Macros to switch the text direction}
3838 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3839 \def\bbl@rscripts{% TODO. Base on codes ??
3840 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3841 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
3842 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
3843 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3844 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3845 Old South Arabian,}%
3846 \def\bbl@provide@dirs#1{%
3847 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3848 \ifin@
3849 \global\bbl@csarg\chardef{wdir@#1}\@ne
3850 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3851 \ifin@
3852 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3853 \fi
3854 \else
3855 \global\bbl@csarg\chardef{wdir@#1}\z@
3856 \fi
3857 \ifodd\bbl@engine
3858 \bbl@csarg\ifcase{wdir@#1}%
3859 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3860 \or
3861 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3862 \or
3863 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3864 \fi
3865 \fi}
3866 \def\bbl@switchdir{%
3867 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3868 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3869 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
3870 \def\bbl@setdirs#1{% TODO - math
3871 \ifcase\bbl@select@type % TODO - strictly, not the right test

```

```

3872 \bbl@bodydir{#1}%
3873 \bbl@pardir{#1}%
3874 \fi
3875 \bbl@textdir{#1}}
3876 % TODO. Only if \bbl@bidimode > 0?:
3877 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3878 \DisableBabelHook{babel-bidi}

```

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

```

3879 \ifodd\bbl@engine % luatex=1
3880 \else % pdftex=0, xetex=2
3881 \newcount\bbl@dirlevel
3882 \chardef\bbl@thetextdir\z@
3883 \chardef\bbl@thepardir\z@
3884 \def\bbl@textdir#1{%
3885   \ifcase#1\relax
3886     \chardef\bbl@thetextdir\z@
3887     \bbl@textdir@i\beginL\endL
3888   \else
3889     \chardef\bbl@thetextdir\@ne
3890     \bbl@textdir@i\beginR\endR
3891   \fi}
3892 \def\bbl@textdir@i#1#2{%
3893   \ifhmode
3894     \ifnum\currentgrouplevel>\z@
3895       \ifnum\currentgrouplevel=\bbl@dirlevel
3896         \bbl@error{Multiple bidi settings inside a group}%
3897         {I'll insert a new group, but expect wrong results.}%
3898         \bgroup\aftergroup#2\aftergroup\egroup
3899       \else
3900         \ifcase\currentgrouptype\or % 0 bottom
3901           \aftergroup#2% 1 simple {}
3902         \or
3903           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3904         \or
3905           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
3906         \or\or\or % vbox vtop align
3907         \or
3908           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3909         \or\or\or\or\or\or % output math disc insert vcent mathchoice
3910         \or
3911           \aftergroup#2% 14 \begingroup
3912         \else
3913           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3914         \fi
3915       \fi
3916       \bbl@dirlevel\currentgrouplevel
3917     \fi
3918     #1%
3919   \fi}
3920 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
3921 \let\bbl@bodydir\@gobble
3922 \let\bbl@pagedir\@gobble
3923 \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).

```

3924 \def\bbl@xebidipar{%

```

```

3925 \let\bbl@xebidipar\relax
3926 \TeXeTstate\@ne
3927 \def\bbl@xeverypar{%
3928   \ifcase\bbl@thepardir
3929     \ifcase\bbl@thetextdir\else\beginR\fi
3930   \else
3931     {\setbox\z@\lastbox\beginR\box\z@}%
3932   \fi}%
3933 \let\bbl@severypar\everypar
3934 \newtoks\everypar
3935 \everypar=\bbl@severypar
3936 \bbl@severypar{\bbl@xeverypar\the\everypar}}
3937 \ifnum\bbl@bidimode>200
3938   \let\bbl@textdir\i\@gobbletwo
3939   \let\bbl@xebidipar\@empty
3940   \AddBabelHook{bidi}{foreign}{%
3941     \def\bbl@tempa{\def\BabelText####1}%
3942     \ifcase\bbl@thetextdir
3943       \expandafter\bbl@tempa\expandafter{\BabelText{LR{##1}}}%
3944     \else
3945       \expandafter\bbl@tempa\expandafter{\BabelText{RL{##1}}}%
3946     \fi}
3947   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
3948 \fi
3949 \fi

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

3950 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
3951 \AtBeginDocument{%
3952   \ifx\pdfstringdefDisableCommands\@undefined\else
3953     \ifx\pdfstringdefDisableCommands\relax\else
3954       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
3955     \fi
3956   \fi}

```

9.6 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`.

```

3957 \bbl@trace{Local Language Configuration}
3958 \ifx\loadlocalcfg\@undefined
3959   \@ifpackagewith{babel}{noconfigs}%
3960   {\let\loadlocalcfg\@gobble}%
3961   {\def\loadlocalcfg#1{%
3962     \InputIfFileExists{#1.cfg}%
3963     {\typeout{*****^J%
3964               * Local config file #1.cfg used^^J%
3965               *}}}%
3966   \@empty}}
3967 \fi

```

9.7 Language options

Languages are loaded when processing the corresponding option *except* if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs

the ldf file and does some additional checks (\input works, too, but possible errors are not caught).

```

3968 \bbl@trace{Language options}
3969 \let\bbl@afterlang\relax
3970 \let\BabelModifiers\relax
3971 \let\bbl@loaded@empty
3972 \def\bbl@load@language#1{%
3973   \InputIfFileExists{#1.ldf}%
3974   {\edef\bbl@loaded{\CurrentOption
3975     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
3976     \expandafter\let\expandafter\bbl@afterlang
3977       \csname\CurrentOption.ldf-h@k\endcsname
3978     \expandafter\let\expandafter\BabelModifiers
3979       \csname bbl@mod@\CurrentOption\endcsname}%
3980   {\bbl@error{%
3981     Unknown option '\CurrentOption'. Either you misspelled it\\%
3982     or the language definition file \CurrentOption.ldf was not found}}{%
3983     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
3984     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
3985     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.

```

3986 \def\bbl@try@load@lang#1#2#3{%
3987   \IfFileExists{\CurrentOption.ldf}%
3988   {\bbl@load@language{\CurrentOption}}}%
3989   {#1\bbl@load@language{#2}#3}}
3990 %
3991 \DeclareOption{hebrew}{%
3992   \input{rlbabel.def}}%
3993   \bbl@load@language{hebrew}}
3994 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
3995 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
3996 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
3997 \DeclareOption{polutonikogreek}{%
3998   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
3999 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4000 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4001 \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.

```

4002 \ifx\bbl@opt@config@nnil
4003   \@ifpackagewith{babel}{noconfigs}{}%
4004   {\InputIfFileExists{bblopts.cfg}%
4005     {\typeout{*****^J%
4006       * Local config file bblopts.cfg used^^J%
4007       *}}}%
4008   }{}%
4009 \else
4010   \InputIfFileExists{\bbl@opt@config.cfg}%
4011   {\typeout{*****^J%
4012     * Local config file \bbl@opt@config.cfg used^^J%
4013     *}}}%
4014   {\bbl@error{%
4015     Local config file '\bbl@opt@config.cfg' not found}}{%
4016     Perhaps you misspelled it.}}%

```

4017 \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 existing languages (note this list also contains the language given with `main` as the last element). If not declared above, the names of the option and the file are the same. There are two steps – first process option names and collect the result, which then do the actual declarations.

To allow multiple overlapping replacements, commas in `bbl@language@opts` are doubled.

```

4018 \let\bbl@elt\relax
4019 \let\bbl@tempe\@empty
4020 \bbl@foreach\@classoptionslist{%
4021   \bbl@xin@{, #1, $}{\bbl@language@opts$}% Match last
4022   \ifin@else
4023     \bbl@xin@{, #1, }{\bbl@language@opts}% Match non-last
4024     \ifin@
4025       \bbl@replace\bbl@language@opts{, #1, }{, , }%
4026       \edef\bbl@tempe{\bbl@tempe\bbl@elt{3}{#1}}%
4027     \else
4028       \babel@savecnt\z@ % Use as temp
4029       \ifnum\bbl@iniflag<\thr@@ % Optimization: 3 = always ini
4030         \IfFileExists{#1.ldf}{\advance\babel@savecnt\@ne}{}%
4031       \fi
4032       \ifnum\bbl@iniflag>\z@ % Optimization: 0 = always ldf
4033         \IfFileExists{babel-#1.tex}{\advance\babel@savecnt\tw@}{}%
4034       \fi
4035       \ifnum\babel@savecnt>\z@
4036         \edef\bbl@tempe{\bbl@tempe\bbl@elt{\the\babel@savecnt}{#1}}%
4037       \fi
4038     \fi
4039   \fi}
4040 %
4041 \let\bbl@savemain\@empty
4042 \bbl@foreach\bbl@language@opts{%
4043   \edef\bbl@tempe{\bbl@tempe\bbl@elt{3}{#1}}
4044   \def\bbl@elt#1#2#3{%
4045     \ifx#3\relax % if last
4046       \bbl@ifunset{ds@#2}{}%
4047       {\bbl@exp{\def\\bbl@savemain{\\DeclareOption{#2}{\[ds@#2]}}}%
4048       \bbl@add\bbl@savemain{\bbl@elt{#1}{#2}}% Save main
4049       \DeclareOption{#2}{}%
4050     \else
4051       \ifnum\bbl@iniflag<\tw@ % other as ldf
4052         \ifodd#1\relax % Class: if ldf exists 1,3. Package: always 3
4053           \bbl@ifunset{ds@#2}{%
4054             {\DeclareOption{#2}{\bbl@load@language{#2}}}%
4055           }%
4056         \fi
4057       \else % other as ini
4058         \ifnum#1>\@ne % % Class: if ini exists 2,3. Package: always 3
4059           \DeclareOption{#2}{%
4060             \bbl@ldfinit
4061             \babelprovide[import]{#2}%
4062             \bbl@afterldf}%
4063           \fi
4064         \fi
4065       \fi
4066     #3}
4067 \bbl@tempe\relax % \relax catches last

```

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.

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

```
4068 \def\AfterBabelLanguage#1{%
4069   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4070 \DeclareOption*{}
4071 \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`.

```
4072 \bbl@trace{Option 'main'}
4073 \ifx\bbl@opt@main\nnil
4074   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4075   \let\bbl@tempc\@empty
4076   \bbl@for\bbl@tempb\bbl@tempa{%
4077     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4078     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4079   \def\bbl@tempa#1,#2\nil{\def\bbl@tempb{#1}}
4080   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4081   \ifx\bbl@tempb\bbl@tempc\else
4082     \bbl@warning{%
4083       Last declared language option is '\bbl@tempc',\%
4084       but the last processed one was '\bbl@tempb'.\%
4085       The main language can't be set as both a global\%
4086       and a package option. Use 'main=\bbl@tempc' as\%
4087       option. Reported}%
4088     \fi
4089   \fi
4090   \def\bbl@elt#1#2{% main
4091     \ifodd\bbl@iniflag % as ini = 1(=), 3(=)
4092       \ifnum#1>\@ne % % Class: if ini exists 2,3. Package: always 3
4093       \def\CurrentOption{#2}% Directly, because luatexbase
4094       \bbl@ldfinit
4095       \babelprovide[\bbl@opt@provide,main,import]{#2}%
4096       \bbl@afterldf}%
4097       \DeclareOption{#2}{}%
4098     \fi
4099   \else % as ldf = 0(no), 2(=)
4100     \ifodd#1\relax % Class: if ldf exists 1,3. Package: always 3
4101     \bbl@ifunset{ds@#2}%
4102       {\DeclareOption{#2}{\bbl@load@language{#2}}}%
4103       {}%
4104     \ExecuteOptions{#2}%
4105     \fi
4106   \fi}
4107 \bbl@savemain
4108 \DeclareOption*{}%
4109 \ProcessOptions*
4110 \def\AfterBabelLanguage{%
4111   \bbl@error
4112   {Too late for \string\AfterBabelLanguage}%
4113   {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.

```
4114 \ifx\bbl@main@language\undefined
4115   \bbl@info{%
4116     You haven't specified a language. I'll use 'nil'\%
4117     as the main language. Reported}
4118   \bbl@load@language{nil}
4119 \fi
4120 \end{package}
```

10 The kernel of Babel (`babel.def`, `common`)

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

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

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

A proxy file for `switch.def`

```
4121 \kernel
4122 \let\bbl@onlyswitch\@empty
4123 \input babel.def
4124 \let\bbl@onlyswitch\@undefined
4125 \end{kernel}
4126 \end{patterns}
```

11 Loading hyphenation patterns

The following code is meant to be read by `ini \TeX` because it should instruct \TeX to read hyphenation patterns. To this end the `docstrip` option `patterns` is used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

```
4127 \langle\langle Make sure ProvidesFile is defined \rangle\rangle
4128 \ProvidesFile{hyphen.cfg}[\langle\langle date \rangle\rangle] [\langle\langle version \rangle\rangle] Babel hyphens]
4129 \xdef\bbl@format{\jobname}
4130 \def\bbl@version{\langle\langle version \rangle\rangle}
4131 \def\bbl@date{\langle\langle date \rangle\rangle}
4132 \ifx\AtBeginDocument\undefined
4133   \def\@empty{}
4134 \fi
4135 \langle\langle Define core switching macros \rangle\rangle
```

`\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.

```
4136 \def\process@line#1#2 #3 #4 {%
4137   \ifx=#1%
4138     \process@synonym{#2}%
4139   \else
4140     \process@language{#1#2}{#3}{#4}%
4141   \fi
4142   \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.

```
4143 \toks@{}
4144 \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.

```
4145 \def\process@synonym#1{%
4146   \ifnum\last@language=\m@ne
4147     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4148   \else
4149     \expandafter\chardef\csname l@#1\endcsname\last@language
4150     \wlog{\string\l@#1=\string\language\the\last@language}%
4151     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4152       \csname\language\hyphenmins\endcsname
4153     \let\bbl@elt\relax
4154     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
4155   \fi}
```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. T_EX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\(lang)hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` or `\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.

```
4156 \def\process@language#1#2#3{%
4157   \expandafter\addlanguage\csname l@#1\endcsname
4158   \expandafter\language\csname l@#1\endcsname
4159   \edef\language{#1}%
4160   \bbl@hook@everylanguage{#1}%
4161   % > luatex
4162   \bbl@get@enc#1::\@@@
4163   \begingroup
4164     \lefthyphenmin\m@ne
```

```

4165 \bbl@hook@loadpatterns{#2}%
4166 % > luatex
4167 \ifnum\lefthyphenmin=\m@ne
4168 \else
4169 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4170 \the\lefthyphenmin\the\righthyphenmin}%
4171 \fi
4172 \endgroup
4173 \def\bbl@tempa{#3}%
4174 \ifx\bbl@tempa\@empty\else
4175 \bbl@hook@loadexceptions{#3}%
4176 % > luatex
4177 \fi
4178 \let\bbl@elt\relax
4179 \edef\bbl@languages{%
4180 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4181 \ifnum\the\language=\z@
4182 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4183 \set@hyphenmins\tw@\thr@@\relax
4184 \else
4185 \expandafter\expandafter\expandafter\set@hyphenmins
4186 \csname #1hyphenmins\endcsname
4187 \fi
4188 \the\toks@
4189 \toks@{}%
4190 \fi}

```

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

```

4191 \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.

```

4192 \def\bbl@hook@everylanguage#1{}
4193 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4194 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4195 \def\bbl@hook@loadkernel#1{%
4196 \def\addlanguage{\csname newlanguage\endcsname}%
4197 \def\adddialect##1##2{%
4198 \global\chardef##1##2\relax
4199 \wlog{\string##1 = a dialect from \string\language##2}}%
4200 \def\iflanguage#1{%
4201 \expandafter\ifx\csname l@##1\endcsname\relax
4202 \@nolanerr{##1}%
4203 \else
4204 \ifnum\csname l@##1\endcsname=\language
4205 \expandafter\expandafter\expandafter\@firstoftwo
4206 \else
4207 \expandafter\expandafter\expandafter\@secondoftwo
4208 \fi
4209 \fi}%
4210 \def\providehyphenmins##1##2{%
4211 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4212 \@namedef{##1hyphenmins}{##2}%
4213 \fi}%
4214 \def\set@hyphenmins##1##2{%
4215 \lefthyphenmin##1\relax

```

```

4216 \righthyphenmin##2\relax}%
4217 \def\selectlanguage{%
4218 \errhelp{Selecting a language requires a package supporting it}%
4219 \errmessage{Not loaded}}%
4220 \let\foreignlanguage\selectlanguage
4221 \let\otherlanguage\selectlanguage
4222 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4223 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4224 \def\setlocale{%
4225 \errhelp{Find an armchair, sit down and wait}%
4226 \errmessage{Not yet available}}%
4227 \let\uselocale\setlocale
4228 \let\locale\setlocale
4229 \let\selectlocale\setlocale
4230 \let\localename\setlocale
4231 \let\textlocale\setlocale
4232 \let\textlanguage\setlocale
4233 \let\languagegettext\setlocale}
4234 \begingroup
4235 \def\AddBabelHook#1#2{%
4236 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4237 \def\next{\toks1}%
4238 \else
4239 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4240 \fi
4241 \next}
4242 \ifx\directlua@undefined
4243 \ifx\XeTeXinputencoding@undefined\else
4244 \input xebabel.def
4245 \fi
4246 \else
4247 \input luababel.def
4248 \fi
4249 \openin1 = babel-\bbl@format.cfg
4250 \ifeof1
4251 \else
4252 \input babel-\bbl@format.cfg\relax
4253 \fi
4254 \closein1
4255 \endgroup
4256 \bbl@hook@loadkernel{switch.def}

```

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

```

4257 \openin1 = language.dat

```

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

```

4258 \def\language{english}%
4259 \ifeof1
4260 \message{I couldn't find the file language.dat,\space
4261 I will try the file hyphen.tex}
4262 \input hyphen.tex\relax
4263 \chardef\l@english\z@
4264 \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.

```

4265 \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.

```
4266 \loop
4267 \endlinechar\m@ne
4268 \read1 to \bbl@line
4269 \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.

```
4270 \if T\ifeof1F\fi T\relax
4271 \ifx\bbl@line\@empty\else
4272 \edef\bbl@line{\bbl@line\space\space\space}%
4273 \expandafter\process@line\bbl@line\relax
4274 \fi
4275 \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.

```
4276 \begingroup
4277 \def\bbl@elt#1#2#3#4{%
4278 \global\language=#2\relax
4279 \gdef\language#1}%
4280 \def\bbl@elt##1##2##3##4{}}%
4281 \bbl@languages
4282 \endgroup
4283 \fi
4284 \closein1
```

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

```
4285 \if/\the\toks@/\else
4286 \errhelp{language.dat loads no language, only synonyms}
4287 \errmessage{Orphan language synonym}
4288 \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.

```
4289 \let\bbl@line\@undefined
4290 \let\process@line\@undefined
4291 \let\process@synonym\@undefined
4292 \let\process@language\@undefined
4293 \let\bbl@get@enc\@undefined
4294 \let\bbl@hyph@enc\@undefined
4295 \let\bbl@tempa\@undefined
4296 \let\bbl@hook@loadkernel\@undefined
4297 \let\bbl@hook@everylanguage\@undefined
4298 \let\bbl@hook@loadpatterns\@undefined
4299 \let\bbl@hook@loadexceptions\@undefined
4300 </patterns>
```

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].


```

4301 <<*More package options>> ≡
4302 \chardef\bbl@bidimode\z@
4303 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4304 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4305 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4306 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4307 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4308 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4309 <</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.

```

4310 <<*Font selection>> ≡
4311 \bbl@trace{Font handling with fontspec}
4312 \ifx\ExplSyntaxOn\@undefined\else
4313   \ExplSyntaxOn
4314   \catcode`\ =10
4315   \def\bbl@loadfontspec{%
4316     \usepackage{fontspec}% TODO. Apply patch always
4317     \expandafter
4318     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4319       Font '\l_fontspec_fontname_tl' is using the\\%
4320       default features for language '##1'.\\%
4321       That's usually fine, because many languages\\%
4322       require no specific features, but if the output is\\%
4323       not as expected, consider selecting another font.}
4324     \expandafter
4325     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4326       Font '\l_fontspec_fontname_tl' is using the\\%
4327       default features for script '##2'.\\%
4328       That's not always wrong, but if the output is\\%
4329       not as expected, consider selecting another font.}}
4330   \ExplSyntaxOff
4331 \fi
4332 \@onlypreamble\babelfont
4333 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4334   \bbl@foreach{#1}{%
4335     \expandafter\ifx\csname date##1\endcsname\relax
4336       \IfFileExists{babel-##1.tex}%
4337       {\babelprovide{##1}}%
4338     }%
4339   \fi}%
4340 \edef\bbl@tempa{#1}%
4341 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4342 \ifx\fontspec\@undefined
4343   \bbl@loadfontspec
4344 \fi
4345 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4346 \bbl@bblfont}
4347 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4348   \bbl@ifunset{\bbl@tempb family}%
4349   {\bbl@providfam{\bbl@tempb}}%
4350   {}%
4351   % For the default font, just in case:
4352   \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%

```

```

4353 \expandafter\bb1@ifblank\expandafter{\bb1@tempa}%
4354 {\bb1@csarg\edef{\bb1@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4355 \bbl@exp{%
4356 \let<\bbl@bb1@tempb dflt@\language\>\bbl@bb1@tempb dflt@>%
4357 \bbl@font@set<\bbl@bb1@tempb dflt@\language\>%
4358 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4359 {\bbl@foreach\bb1@tempa{% ie bbl@rmdflt@lang / *srt
4360 \bbl@csarg\def{\bb1@tempb dflt@##1}{<{#1}{#2}}}}%

```

If the family in the previous command does not exist, it must be defined. Here is how:

```

4361 \def\bbl@providfam#1{%
4362 \bbl@exp{%
4363 \\\newcommand<#1default>{}% Just define it
4364 \\\bbl@add@list\\bbl@font@fams{#1}%
4365 \\\DeclareRobustCommand<#1family>{%
4366 \\\not@math@alphabet<#1family>\relax
4367 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4368 \\\fontfamily<#1default>%
4369 \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4370 \\\selectfont}%
4371 \\\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.

```

4372 \def\bbl@nostdfont#1{%
4373 \bbl@ifunset{bbl@WFF@f@family}%
4374 {\bbl@csarg\gdef{WFF@f@family}}% Flag, to avoid dupl warns
4375 \bbl@infowarn{The current font is not a babel standard family:\\%
4376 #1%
4377 \fontname\font\\%
4378 There is nothing intrinsically wrong with this warning, and\\%
4379 you can ignore it altogether if you do not need these\\%
4380 families. But if they are used in the document, you should be\\%
4381 aware 'babel' will no set Script and Language for them, so\\%
4382 you may consider defining a new family with \string\babelfont.\\%
4383 See the manual for further details about \string\babelfont.\\%
4384 Reported}}
4385 {}}%
4386 \gdef\bbl@switchfont{%
4387 \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
4388 \bbl@exp{% eg Arabic -> arabic
4389 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4390 \bbl@foreach\bb1@font@fams{%
4391 \bbl@ifunset{bbl@##1dflt@\language}% (1) language?
4392 {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4393 {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4394 {}% 123=F - nothing!
4395 {\bbl@exp{% 3=T - from generic
4396 \global\let<bbl@##1dflt@\language>%
4397 \<bbl@##1dflt@>}}}%
4398 {\bbl@exp{% 2=T - from script
4399 \global\let<bbl@##1dflt@\language>%
4400 \<bbl@##1dflt@*\bbl@tempa>}}}%
4401 {}% 1=T - language, already defined
4402 \def\bbl@tempa{\bbl@nostdfont}}}%
4403 \bbl@foreach\bb1@font@fams{% don't gather with prev for
4404 \bbl@ifunset{bbl@##1dflt@\language}%
4405 {\bbl@cs{famrst@##1}%
4406 \global\bbl@csarg\let{famrst@##1}\relax}%

```

```

4407      {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4408      \\bbl@add\\originalTeX{%
4409      \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4410      \<##1default>\<##1family>{##1}}}%
4411      \\bbl@font@set\<bbl@##1dflt@language>% the main part!
4412      \<##1default>\<##1family>}}}%
4413      \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.

```

4414 \ifx\fbfamily\undefined\else      % if latex
4415 \ifcase\bbl@engine                  % if pdftex
4416   \let\bbl@cckcstdfonts\relax
4417 \else
4418   \def\bbl@cckcstdfonts{%
4419     \begingroup
4420     \global\let\bbl@cckcstdfonts\relax
4421     \let\bbl@tempa\@empty
4422     \bbl@foreach\bbl@font@fams{%
4423       \bbl@ifunset{bbl@##1dflt@}%
4424       {\nameuse{##1family}}%
4425       \bbl@csarg\gdef{WFF@fbfamily}}}% Flag
4426       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= fbfamily\\}%
4427       \space\space\fontname\font\\}%
4428       \bbl@csarg\xdef{##1dflt@}{fbfamily}}%
4429       \expandafter\xdef\csname ##1default\endcsname{fbfamily}}}%
4430       {}}%
4431   \ifx\bbl@tempa\@empty\else
4432     \bbl@infowarn{The following font families will use the default\\%
4433     settings for all or some languages:\\%
4434     \bbl@tempa
4435     There is nothing intrinsically wrong with it, but\\%
4436     'babel' will no set Script and Language, which could\\%
4437     be relevant in some languages. If your document uses\\%
4438     these families, consider redefining them with \string\babelfont.\\%
4439     Reported}%
4440   \fi
4441 \endgroup}
4442 \fi
4443 \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.

```

4444 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4445   \bbl@xin@{<>}{#1}}%
4446   \ifin@
4447     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4448   \fi
4449   \bbl@exp{%
4450     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4451     \\bbl@ifsamestring{#2}{fbfamily}}%
4452     {\#3%
4453     \\bbl@ifsamestring{fbseries}{bfdefault}{\bfseries}}}%
4454     \let\\bbl@tempa\relax}%
4455   {}}}
4456 % TODO - next should be global?, but even local does its job. I'm
4457 % still not sure -- must investigate:

```

```

4458 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4459 \let\bbl@tempe\bbl@mapselect
4460 \let\bbl@mapselect\relax
4461 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4462 \let#4\empty % Make sure \renewfontfamily is valid
4463 \bbl@exp{%
4464 \let\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4465 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4466 {\bbl@newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4467 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4468 {\bbl@newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4469 \renewfontfamily\#4%
4470 [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{..}{#3}
4471 \begingroup
4472 #4%
4473 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4474 \endgroup
4475 \let#4\bbl@temp@fam
4476 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4477 \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.

```

4478 \def\bbl@font@rst#1#2#3#4{%
4479 \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.

```

4480 \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 :-).

```

4481 \newcommand\babelFSstore[2][{%
4482 \bbl@ifblank{#1}%
4483 {\bbl@csarg\def{sname@#2}{Latin}}%
4484 {\bbl@csarg\def{sname@#2}{#1}}%
4485 \bbl@provide@dirs{#2}%
4486 \bbl@csarg\ifnum{wdir@#2}>\z@
4487 \let\bbl@beforeforeign\leavevmode
4488 \EnableBabelHook{babel-bidi}%
4489 \fi
4490 \bbl@foreach{#2}{%
4491 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4492 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4493 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4494 \def\bbl@FSstore#1#2#3#4{%
4495 \bbl@csarg\edef{#2default#1}{#3}%
4496 \expandafter\addto\csname extras#1\endcsname{%
4497 \let#4#3%
4498 \ifx#3\f@family
4499 \edef#3{\csname bbl@#2default#1\endcsname}%
4500 \fontfamily{#3}\selectfont
4501 \else
4502 \edef#3{\csname bbl@#2default#1\endcsname}%
4503 \fi}%
4504 \expandafter\addto\csname noextras#1\endcsname{%
4505 \ifx#3\f@family
4506 \fontfamily{#4}\selectfont
4507 \fi

```

```

4508 \let#3#4}}
4509 \let\bbbl@langfeatures\@empty
4510 \def\babelFSfeatures{% make sure \fontspec is redefined once
4511 \let\bbbl@ori@fontspec\fontspec
4512 \renewcommand\fontspec[1][{}]{%
4513 \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
4514 \let\babelFSfeatures\bbbl@FSfeatures
4515 \babelFSfeatures}
4516 \def\bbbl@FSfeatures#1#2{%
4517 \expandafter\addto\csname extras#1\endcsname{%
4518 \babel@save\bbbl@langfeatures
4519 \edef\bbbl@langfeatures{#2,}}}}
4520 <</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.

```

4521 <<{*Footnote changes}>> ≡
4522 \bbbl@trace{Bidi footnotes}
4523 \ifnum\bbbl@bidimode>\z@
4524 \def\bbbl@footnote#1#2#3{%
4525 \ifnextchar[%
4526 {\bbbl@footnote@o{#1}{#2}{#3}}%
4527 {\bbbl@footnote@x{#1}{#2}{#3}}}
4528 \long\def\bbbl@footnote@x#1#2#3#4{%
4529 \bgroup
4530 \select@language@x{\bbbl@main@language}%
4531 \bbbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4532 \egroup}
4533 \long\def\bbbl@footnote@o#1#2#3[#4]#5{%
4534 \bgroup
4535 \select@language@x{\bbbl@main@language}%
4536 \bbbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4537 \egroup}
4538 \def\bbbl@footnotetext#1#2#3{%
4539 \ifnextchar[%
4540 {\bbbl@footnotetext@o{#1}{#2}{#3}}%
4541 {\bbbl@footnotetext@x{#1}{#2}{#3}}}
4542 \long\def\bbbl@footnotetext@x#1#2#3#4{%
4543 \bgroup
4544 \select@language@x{\bbbl@main@language}%
4545 \bbbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4546 \egroup}
4547 \long\def\bbbl@footnotetext@o#1#2#3[#4]#5{%
4548 \bgroup
4549 \select@language@x{\bbbl@main@language}%
4550 \bbbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4551 \egroup}
4552 \def\BabelFootnote#1#2#3#4{%
4553 \ifx\bbbl@fn@footnote\@undefined
4554 \let\bbbl@fn@footnote\footnote
4555 \fi
4556 \ifx\bbbl@fn@footnotetext\@undefined
4557 \let\bbbl@fn@footnotetext\footnotetext
4558 \fi

```

```

4559 \bbl@ifblank{#2}%
4560 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4561 \namedef{\bbl@stripslash#1text}%
4562 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4563 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}{#3}{#4}}}%
4564 \namedef{\bbl@stripslash#1text}%
4565 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}}
4566 \fi
4567 <</Footnote changes>>

```

Now, the code.

```

4568 (*xetex)
4569 \def\BabelStringsDefault{unicode}
4570 \let\xebbl@stop\relax
4571 \AddBabelHook{xetex}{encodedcommands}{%
4572 \def\bbl@tempa{#1}%
4573 \ifx\bbl@tempa\empty
4574 \XeTeXinputencoding"bytes"%
4575 \else
4576 \XeTeXinputencoding"#1"%
4577 \fi
4578 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4579 \AddBabelHook{xetex}{stopcommands}{%
4580 \xebbl@stop
4581 \let\xebbl@stop\relax}
4582 \def\bbl@intraspace#1 #2 #3\@@{%
4583 \bbl@csarg\gdef{xeisp@\language}%
4584 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4585 \def\bbl@intrapenalty#1\@@{%
4586 \bbl@csarg\gdef{xeipn@\language}%
4587 {\XeTeXlinebreakpenalty #1\relax}}
4588 \def\bbl@provide@intraspace{%
4589 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4590 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4591 \ifin@
4592 \bbl@ifunset{bbl@intsp@\language}{}%
4593 {\expandafter\ifx\csname bbl@intsp@\language\endcsname\empty\else
4594 \ifx\bbl@KVP@intraspace\@nil
4595 \bbl@exp{%
4596 \bbl@intraspace\bbl@cl{intsp}\@@}%
4597 \fi
4598 \ifx\bbl@KVP@intrapenalty\@nil
4599 \bbl@intrapenalty0\@@
4600 \fi
4601 \fi
4602 \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4603 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4604 \fi
4605 \ifx\bbl@KVP@intrapenalty\@nil\else
4606 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4607 \fi
4608 \bbl@exp{%
4609 % TODO. Execute only once (but redundant):
4610 \bbl@add\<extras\language>{%
4611 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4612 \<bbl@xeisp@\language>%
4613 \<bbl@xeipn@\language>%
4614 \bbl@tglobal\<extras\language>%
4615 \bbl@add\<noextras\language>%

```

```

4616 \XeTeXlinebreaklocale "en"%
4617 \\bbl@tglobal\<noextras\language>}%
4618 \ifx\bbl@ispace\undefined
4619 \gdef\bbl@ispace{\bbl@cl{xisp}}%
4620 \ifx\AtBeginDocument\@notprerr
4621 \expandafter\@secondoftwo % to execute right now
4622 \fi
4623 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4624 \fi}%
4625 \fi}
4626 \ifx\DisableBabelHook\@undefined\endinput\fi
4627 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4628 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4629 \DisableBabelHook{babel-fontspec}
4630 <<Font selection>>
4631 \input txtbabel.def
4632 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titles, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the \TeX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdfTeX and xetex.

```

4633 (*texxet)
4634 \providecommand\bbl@provide@intraspace{}
4635 \bbl@trace{Redefinitions for bidi layout}
4636 \def\bbl@sspre@caption{%
4637 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4638 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4639 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4640 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4641 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4642 \def\@hangfrom#1{%
4643 \setbox\@tempboxa\hbox{#1}}%
4644 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4645 \noindent\box\@tempboxa}
4646 \def\raggedright{%
4647 \let\@centercr
4648 \bbl@startskip\z@skip
4649 \@rightskip\@flushglue
4650 \bbl@endskip\@rightskip
4651 \parindent\z@
4652 \parfillskip\bbl@startskip}
4653 \def\raggedleft{%
4654 \let\@centercr
4655 \bbl@startskip\@flushglue
4656 \bbl@endskip\z@skip
4657 \parindent\z@
4658 \parfillskip\bbl@endskip}
4659 \fi
4660 \IfBabelLayout{lists}
4661 {\bbl@sreplace\list
4662 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4663 \def\bbl@listleftmargin{%

```

```

4664 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4665 \ifcase\bbl@engine
4666 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4667 \def\p@enumiii{\p@enumii}\theenumii{}\p@enumii{}
4668 \fi
4669 \bbl@sreplace\@verbatim
4670 {\leftskip\@totalleftmargin}%
4671 {\bbl@startskip\textwidth
4672 \advance\bbl@startskip-\linewidth}%
4673 \bbl@sreplace\@verbatim
4674 {\rightskip\z@skip}%
4675 {\bbl@endskip\z@skip}}%
4676 {}
4677 \IfBabelLayout{contents}
4678 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4679 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4680 {}
4681 \IfBabelLayout{columns}
4682 {\bbl@sreplace\@outputdblcol{\hb@xt\textwidth}{\bbl@outputbox}%
4683 \def\bbl@outputbox#1{%
4684 \hb@xt\textwidth{%
4685 \hskip\columnwidth
4686 \hfil
4687 {\normalcolor\vrule \@width\columnseprule}%
4688 \hfil
4689 \hb@xt\columnwidth{\box\@leftcolumn \hss}%
4690 \hskip-\textwidth
4691 \hb@xt\columnwidth{\box\@outputbox \hss}%
4692 \hskip\columnsep
4693 \hskip\columnwidth}}}%
4694 {}
4695 <<Footnote changes>>
4696 \IfBabelLayout{footnotes}%
4697 {\BabelFootnote\footnote\language\{}}%
4698 \BabelFootnote\localfootnote\language\{}}%
4699 \BabelFootnote\mainfootnote\{}}{}
4700 {}

```

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.

```

4701 \IfBabelLayout{counters}%
4702 {\let\bbl@latinarabic=\@arabic
4703 \def\@arabic#1{\bbl@sublr{\bbl@latinarabic#1}}%
4704 \let\bbl@asciroman=\@roman
4705 \def\@roman#1{\bbl@sublr{\ensureascii{\bbl@asciroman#1}}}%
4706 \let\bbl@asciiRoman=\@Roman
4707 \def\@Roman#1{\bbl@sublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4708 </texxet>

```

13.3 LuaTeX

The loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means

when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for ‘english’, so that it’s available without further intervention from the user. To avoid duplicating it, the following rule applies: if the “0th” language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won’t at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn’t happen very often – with luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn’t work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn’t true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4709 (*luatex)
4710 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4711 \bbl@trace{Read language.dat}
4712 \ifx\bbl@readstream\undefined
4713   \csname newread\endcsname\bbl@readstream
4714 \fi
4715 \begingroup
4716   \toks@{}
4717   \count@ \z@ % 0=start, 1=0th, 2=normal
4718   \def\bbl@process@line#1#2 #3 #4 {%
4719     \ifx=#1%
4720       \bbl@process@synonym{#2}%
4721     \else
4722       \bbl@process@language{#1#2}{#3}{#4}%
4723     \fi
4724     \ignorespaces}
4725   \def\bbl@manylang{%
4726     \ifnum\bbl@last>\@ne
4727       \bbl@info{Non-standard hyphenation setup}%
4728     \fi
4729     \let\bbl@manylang\relax}
4730   \def\bbl@process@language#1#2#3{%
4731     \ifcase\count@
4732       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4733     \or
4734       \count@\tw@
4735     \fi
4736     \ifnum\count@=\tw@
4737       \expandafter\addlanguage\csname l@#1\endcsname
4738       \language\allocationnumber
4739       \chardef\bbl@last\allocationnumber
4740       \bbl@manylang
4741       \let\bbl@elt\relax

```

```

4742 \xdef\bbl@languages{%
4743 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4744 \fi
4745 \the\toks@
4746 \toks@{}}
4747 \def\bbl@process@synonym@aux#1#2{%
4748 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4749 \let\bbl@elt\relax
4750 \xdef\bbl@languages{%
4751 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4752 \def\bbl@process@synonym#1{%
4753 \ifcase\count@
4754 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4755 \or
4756 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4757 \else
4758 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4759 \fi}
4760 \ifx\bbl@languages@\undefined % Just a (sensible?) guess
4761 \chardef\l@english\z@
4762 \chardef\l@USenglish\z@
4763 \chardef\bbl@last\z@
4764 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
4765 \gdef\bbl@languages{%
4766 \bbl@elt{english}{0}{\hyphen.tex}}%
4767 \bbl@elt{USenglish}{0}{}}
4768 \else
4769 \global\let\bbl@languages@format\bbl@languages
4770 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4771 \ifnum#2>\z@\else
4772 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4773 \fi}%
4774 \xdef\bbl@languages{\bbl@languages}%
4775 \fi
4776 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4777 \bbl@languages
4778 \openin\bbl@readstream=language.dat
4779 \ifeof\bbl@readstream
4780 \bbl@warning{I couldn't find language.dat. No additional\\%
4781 patterns loaded. Reported}%
4782 \else
4783 \loop
4784 \endlinechar\m@ne
4785 \read\bbl@readstream to \bbl@line
4786 \endlinechar``^^M
4787 \if T\ifeof\bbl@readstream F\fi T\relax
4788 \ifx\bbl@line\empty\else
4789 \edef\bbl@line{\bbl@line\space\space\space}%
4790 \expandafter\bbl@process@line\bbl@line\relax
4791 \fi
4792 \repeat
4793 \fi
4794 \endgroup
4795 \bbl@trace{Macros for reading patterns files}
4796 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
4797 \ifx\babelcatcodetablenum\undefined
4798 \ifx\newcatcodetable\undefined
4799 \def\babelcatcodetablenum{5211}
4800 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}

```

```

4801 \else
4802 \newcatcodetable\babelcatcodetablenum
4803 \newcatcodetable\bbl@pattcodes
4804 \fi
4805 \else
4806 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4807 \fi
4808 \def\bbl@luapatterns#1#2{%
4809 \bbl@get@enc#1::\@@@
4810 \setbox\z@\hbox\bgroup
4811 \begingroup
4812 \savecatcodetable\babelcatcodetablenum\relax
4813 \initcatcodetable\bbl@pattcodes\relax
4814 \catcodetable\bbl@pattcodes\relax
4815 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4816 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
4817 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4818 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4819 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4820 \catcode\`=12 \catcode\`=12 \catcode\"=12
4821 \input #1\relax
4822 \catcodetable\babelcatcodetablenum\relax
4823 \endgroup
4824 \def\bbl@tempa{#2}%
4825 \ifx\bbl@tempa@empty\else
4826 \input #2\relax
4827 \fi
4828 \egroup}%
4829 \def\bbl@patterns@lua#1{%
4830 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4831 \csname l@#1\endcsname
4832 \edef\bbl@tempa{#1}%
4833 \else
4834 \csname l@#1:\f@encoding\endcsname
4835 \edef\bbl@tempa{#1:\f@encoding}%
4836 \fi\relax
4837 \@namedef{luatexhyphen@loaded@the\language}{}% Temp
4838 \@ifundefined{bbl@hyphendata@the\language}%
4839 {\def\bbl@elt##1##2##3##4{%
4840 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4841 \def\bbl@tempb{##3}%
4842 \ifx\bbl@tempb@empty\else % if not a synonymous
4843 \def\bbl@tempc{##3}{##4}%
4844 \fi
4845 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4846 \fi}%
4847 \bbl@languages
4848 \@ifundefined{bbl@hyphendata@the\language}%
4849 {\bbl@info{No hyphenation patterns were set for\%
4850 language '\bbl@tempa'. Reported}}%
4851 {\expandafter\expandafter\expandafter\bbl@luapatterns
4852 \csname bbl@hyphendata@the\language\endcsname}}}%
4853 \endinput\fi
4854 % Here ends \ifx\AddBabelHook\@undefined
4855 % A few lines are only read by hyphen.cfg
4856 \ifx\DisableBabelHook\@undefined
4857 \AddBabelHook{luatex}{everylanguage}{%
4858 \def\process@language##1##2##3{%
4859 \def\process@line####1####2 ####3 ####4 {}}}

```

```

4860 \AddBabelHook{luatex}{loadpatterns}{%
4861   \input #1\relax
4862   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4863     {{#1}}}}
4864 \AddBabelHook{luatex}{loadexceptions}{%
4865   \input #1\relax
4866   \def\bbl@tempb##1##2{{##1}{##2}}%
4867   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4868     {\expandafter\expandafter\expandafter\bbl@tempb
4869       \csname bbl@hyphendata@the\language\endcsname}}
4870 \endinput\fi
4871 % Here stops reading code for hyphen.cfg
4872 % The following is read the 2nd time it's loaded
4873 \begingroup % TODO - to a lua file
4874 \catcode`\%=12
4875 \catcode`\'=12
4876 \catcode`\=12
4877 \catcode`\:=12
4878 \directlua{
4879   Babel = Babel or {}
4880   function Babel.bytes(line)
4881     return line:gsub(".",
4882       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4883   end
4884   function Babel.begin_process_input()
4885     if luatexbase and luatexbase.add_to_callback then
4886       luatexbase.add_to_callback('process_input_buffer',
4887         Babel.bytes, 'Babel.bytes')
4888     else
4889       Babel.callback = callback.find('process_input_buffer')
4890       callback.register('process_input_buffer', Babel.bytes)
4891     end
4892   end
4893   function Babel.end_process_input ()
4894     if luatexbase and luatexbase.remove_from_callback then
4895       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4896     else
4897       callback.register('process_input_buffer', Babel.callback)
4898     end
4899   end
4900   function Babel.addpatterns(pp, lg)
4901     local lg = lang.new(lg)
4902     local pats = lang.patterns(lg) or ''
4903     lang.clear_patterns(lg)
4904     for p in pp:gmatch('[^%s]+') do
4905       ss = ''
4906       for i in string.utfcharacters(p:gsub('%d', '')) do
4907         ss = ss .. '%d?' .. i
4908       end
4909       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4910       ss = ss:gsub('%.%%d%?$', '%%.')
4911       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4912       if n == 0 then
4913         tex.sprint(
4914           [[\string\csname\space bbl@info\endcsname{New pattern: }
4915             .. p .. [{}]])
4916         pats = pats .. ' ' .. p
4917       else
4918         tex.sprint(

```

```

4919         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
4920         .. p .. [{}]])
4921     end
4922 end
4923 lang.patterns(lg, pats)
4924 end
4925 }
4926 \endgroup
4927 \ifx\newattribute\undefined\else
4928   \newattribute\bbl@attr@locale
4929   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
4930   \AddBabelHook{luatex}{beforeextras}{%
4931     \setattribute\bbl@attr@locale\localeid}
4932 \fi
4933 \def\BabelStringsDefault{unicode}
4934 \let\luabbl@stop\relax
4935 \AddBabelHook{luatex}{encodedcommands}{%
4936   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4937   \ifx\bbl@tempa\bbl@tempb\else
4938     \directlua{Babel.begin_process_input()}%
4939     \def\luabbl@stop{%
4940       \directlua{Babel.end_process_input()}}%
4941   \fi}%
4942 \AddBabelHook{luatex}{stopcommands}{%
4943   \luabbl@stop
4944   \let\luabbl@stop\relax}
4945 \AddBabelHook{luatex}{patterns}{%
4946   \@ifundefined{bbl@hyphendata@the\language}%
4947   {\def\bbl@elt##1##2##3##4{%
4948     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
4949     \def\bbl@tempb{##3}%
4950     \ifx\bbl@tempb\@empty\else % if not a synonymous
4951       \def\bbl@tempc{##3}{##4}}%
4952     \fi
4953     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4954     \fi}%
4955   \bbl@languages
4956   \@ifundefined{bbl@hyphendata@the\language}%
4957   {\bbl@info{No hyphenation patterns were set for\%
4958     language '#2'. Reported}}%
4959   {\expandafter\expandafter\expandafter\bbl@luapatterns
4960     \csname bbl@hyphendata@the\language\endcsname}}}%
4961 \@ifundefined{bbl@patterns@}{}%
4962 \begingroup
4963   \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
4964   \ifin@else
4965     \ifx\bbl@patterns@\@empty\else
4966       \directlua{ Babel.addpatterns(
4967         [[\bbl@patterns@]], \number\language) }%
4968       \fi
4969       \@ifundefined{bbl@patterns@#1}%
4970       \@empty
4971       {\directlua{ Babel.addpatterns(
4972         [[\space\csname bbl@patterns@#1\endcsname]],
4973         \number\language) }}%
4974       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4975     \fi
4976   \endgroup}%
4977 \bbl@exp{%

```

```

4978 \bbl@ifunset{bbl@prehc@language}{}%
4979 {\bbl@ifblank{bbl@cs{prehc@language}}}%
4980 {\prehyphenchar=bbl@c1{prehc}\relax}}

\babelpatterns This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones
and \bbl@patterns@<lang> for language ones. We make sure there is a space between words when
multiple commands are used.

4981 \@onlypreamble\babelpatterns
4982 \AtEndOfPackage{%
4983 \newcommand\babelpatterns[2][\@empty]{%
4984 \ifx\bbl@patterns@relax
4985 \let\bbl@patterns@empty
4986 \fi
4987 \ifx\bbl@pttnlistemptyelse
4988 \bbl@warning{%
4989 You must not intermingle \string\selectlanguage\space and\%
4990 \string\babelpatterns\space or some patterns will not\%
4991 be taken into account. Reported}%
4992 \fi
4993 \ifx\@empty#1%
4994 \protected@edef\bbl@patterns@{\bbl@patterns@space#2}%
4995 \else
4996 \edef\bbl@tempb{\zap@space#1 \@empty}%
4997 \bbl@for\bbl@tempa\bbl@tempb{%
4998 \bbl@fixname\bbl@tempa
4999 \bbl@iflanguage\bbl@tempa{%
5000 \bbl@csarg\protected@edef{patterns@bbl@tempa}{%
5001 \ifundefined{bbl@patterns@bbl@tempa}%
5002 \@empty
5003 {\csname bbl@patterns@bbl@tempa\endcsname\space}%
5004 #2}}}%
5005 \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by \babelposthyphenation. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```

5006 % TODO - to a lua file
5007 \directlua{
5008 Babel = Babel or {}
5009 Babel.linebreaking = Babel.linebreaking or {}
5010 Babel.linebreaking.before = {}
5011 Babel.linebreaking.after = {}
5012 Babel.locale = {} % Free to use, indexed by \localeid
5013 function Babel.linebreaking.add_before(func)
5014 tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5015 table.insert(Babel.linebreaking.before, func)
5016 end
5017 function Babel.linebreaking.add_after(func)
5018 tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5019 table.insert(Babel.linebreaking.after, func)
5020 end
5021 }
5022 \def\bbl@intraspace#1 #2 #3\@{
5023 \directlua{
5024 Babel = Babel or {}

```

```

5025 Babel.intraspaces = Babel.intraspaces or {}
5026 Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
5027 {b = #1, p = #2, m = #3}
5028 Babel.locale_props[\the\localeid].intraspace = %
5029 {b = #1, p = #2, m = #3}
5030 }}
5031 \def\bbl@intrapenalty#1\@@{%
5032 \directlua{
5033 Babel = Babel or {}
5034 Babel.intrapenalties = Babel.intrapenalties or {}
5035 Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5036 Babel.locale_props[\the\localeid].intrapenalty = #1
5037 }}
5038 \begingroup
5039 \catcode`\%=12
5040 \catcode`\^=14
5041 \catcode`\'=12
5042 \catcode`\~=12
5043 \gdef\bbl@seaintraspace{^
5044 \let\bbl@seaintraspace\relax
5045 \directlua{
5046 Babel = Babel or {}
5047 Babel.sea_enabled = true
5048 Babel.sea_ranges = Babel.sea_ranges or {}
5049 function Babel.set_chranges (script, chrng)
5050 local c = 0
5051 for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5052 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5053 c = c + 1
5054 end
5055 end
5056 function Babel.sea_disc_to_space (head)
5057 local sea_ranges = Babel.sea_ranges
5058 local last_char = nil
5059 local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5060 for item in node.traverse(head) do
5061 local i = item.id
5062 if i == node.id'glyph' then
5063 last_char = item
5064 elseif i == 7 and item.subtype == 3 and last_char
5065 and last_char.char > 0x0C99 then
5066 quad = font.getfont(last_char.font).size
5067 for lg, rg in pairs(sea_ranges) do
5068 if last_char.char > rg[1] and last_char.char < rg[2] then
5069 lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyr11
5070 local intraspace = Babel.intraspaces[lg]
5071 local intrapenalty = Babel.intrapenalties[lg]
5072 local n
5073 if intrapenalty ~= 0 then
5074 n = node.new(14, 0) ^% penalty
5075 n.penalty = intrapenalty
5076 node.insert_before(head, item, n)
5077 end
5078 n = node.new(12, 13) ^% (glue, spaceskip)
5079 node.setglue(n, intraspace.b * quad,
5080 intraspace.p * quad,
5081 intraspace.m * quad)
5082 node.insert_before(head, item, n)
5083 node.remove(head, item)

```

```

5084         end
5085     end
5086 end
5087 end
5088 end
5089 }^^
5090 \bbl@luahyphenate}

```

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.

```

5091 \catcode`\%=14
5092 \gdef\bbl@cjkintraspacespace{%
5093   \let\bbl@cjkintraspacespace\relax
5094   \directlua{
5095     Babel = Babel or {}
5096     require('babel-data-cjk.lua')
5097     Babel.cjk_enabled = true
5098     function Babel.cjk_linebreak(head)
5099       local GLYPH = node.id'glyph'
5100       local last_char = nil
5101       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5102       local last_class = nil
5103       local last_lang = nil
5104
5105       for item in node.traverse(head) do
5106         if item.id == GLYPH then
5107
5108           local lang = item.lang
5109
5110           local LOCALE = node.get_attribute(item,
5111             Babel.attr_locale)
5112           local props = Babel.locale_props[LOCALE]
5113
5114           local class = Babel.cjk_class[item.char].c
5115
5116           if props.cjk_quotes and props.cjk_quotes[item.char] then
5117             class = props.cjk_quotes[item.char]
5118           end
5119
5120           if class == 'cp' then class = 'cl' end % )] as CL
5121           if class == 'id' then class = 'I' end
5122
5123           local br = 0
5124           if class and last_class and Babel.cjk_breaks[last_class][class] then
5125             br = Babel.cjk_breaks[last_class][class]
5126           end
5127
5128           if br == 1 and props.linebreak == 'c' and
5129             lang ~= \the\l@nohyphenation\space and
5130             last_lang ~= \the\l@nohyphenation then
5131             local intrapenalty = props.intrapenalty
5132             if intrapenalty ~= 0 then

```



```

5133         local n = node.new(14, 0)      % penalty
5134         n.penalty = intrapenalty
5135         node.insert_before(head, item, n)
5136     end
5137     local intraspace = props.intraspace
5138     local n = node.new(12, 13)          % (glue, spaceskip)
5139     node.setglue(n, intraspace.b * quad,
5140                  intraspace.p * quad,
5141                  intraspace.m * quad)
5142     node.insert_before(head, item, n)
5143 end
5144
5145     if font.getfont(item.font) then
5146         quad = font.getfont(item.font).size
5147     end
5148     last_class = class
5149     last_lang = lang
5150     else % if penalty, glue or anything else
5151         last_class = nil
5152     end
5153 end
5154 lang.hyphenate(head)
5155 end
5156 }%
5157 \bbl@luahyphenate}
5158 \gdef\bbl@luahyphenate{%
5159 \let\bbl@luahyphenate\relax
5160 \directlua{
5161     luatexbase.add_to_callback('hyphenate',
5162     function (head, tail)
5163         if Babel.linebreaking.before then
5164             for k, func in ipairs(Babel.linebreaking.before) do
5165                 func(head)
5166             end
5167         end
5168         if Babel.cjk_enabled then
5169             Babel.cjk_linebreak(head)
5170         end
5171         lang.hyphenate(head)
5172         if Babel.linebreaking.after then
5173             for k, func in ipairs(Babel.linebreaking.after) do
5174                 func(head)
5175             end
5176         end
5177         if Babel.sea_enabled then
5178             Babel.sea_disc_to_space(head)
5179         end
5180     end,
5181     'Babel.hyphenate')
5182 }
5183 }
5184 \endgroup
5185 \def\bbl@provide@intraspace{%
5186 \bbl@ifunset{bbl@intsp@language}{}%
5187 {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5188 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5189 \ifin@           % cjk
5190 \bbl@cjk@intraspace
5191 \directlua{

```

```

5192         Babel = Babel or {}
5193         Babel.locale_props = Babel.locale_props or {}
5194         Babel.locale_props[\the\localeid].linebreak = 'c'
5195     }%
5196     \bbl@exp{\bbbl@intraspace\bbl@cl{intsp}\@}%
5197     \ifx\bbbl@KVP@intrapenalty\@nil
5198         \bbl@intrapenalty0\@@
5199     \fi
5200 \else           % sea
5201     \bbl@seaintraspace
5202     \bbl@exp{\bbbl@intraspace\bbl@cl{intsp}\@}%
5203     \directlua{
5204         Babel = Babel or {}
5205         Babel.sea_ranges = Babel.sea_ranges or {}
5206         Babel.set_chranges('\bbl@cl{sbc}',
5207                             '\bbl@cl{chrng}')
5208     }%
5209     \ifx\bbbl@KVP@intrapenalty\@nil
5210         \bbl@intrapenalty0\@@
5211     \fi
5212 \fi
5213 \fi
5214 \ifx\bbbl@KVP@intrapenalty\@nil\else
5215     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5216 \fi}}

```

13.6 Arabic justification

```

5217 \ifnum\bbbl@bidimode>100 \ifnum\bbbl@bidimode<200
5218 \def\bblar@chars{%
5219     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5220     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5221     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5222 \def\bblar@elongated{%
5223     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5224     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5225     0649,064A}
5226 \begingroup
5227 \catcode\_ =11 \catcode`:=11
5228 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5229 \endgroup
5230 \gdef\bbl@arabicjust{%
5231     \let\bbl@arabicjust\relax
5232     \newattribute\bblar@kashida
5233     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5234     \bblar@kashida=\z@
5235     \bbl@patchfont{\bbl@parsejalt}}%
5236 \directlua{
5237     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5238     Babel.arabic.elong_map[\the\localeid] = {}
5239     luatexbase.add_to_callback('post_linebreak_filter',
5240         Babel.arabic.justify, 'Babel.arabic.justify')
5241     luatexbase.add_to_callback('hpack_filter',
5242         Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5243 }}%
5244 % Save both node lists to make replacement. TODO. Save also widths to
5245 % make computations
5246 \def\bblar@fetchjalt#1#2#3#4{%
5247     \bbl@exp{\bbbl@foreach{#1}}}%

```

```

5248 \bbl@ifunset{bblar@JE@##1}%
5249 {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5250 {\setbox\z@\hbox{^^^^200d\char"@nameuse{bblar@JE@##1}#2}}%
5251 \directlua{%
5252   local last = nil
5253   for item in node.traverse(tex.box[0].head) do
5254     if item.id == node.id'glyph' and item.char > 0x600 and
5255       not (item.char == 0x200D) then
5256       last = item
5257     end
5258   end
5259   Babel.arabic.#3['##1#4'] = last.char
5260 }}
5261 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5262 % perhaps other tables (falt?, cswb?). What about kaf? And diacritic
5263 % positioning?
5264 \gdef\bbl@parsejalt{%
5265   \ifx\addfontfeature\undefined\else
5266     \bbl@xin@{/e}{/\bbl@c{l}{lnbrk}}%
5267     \ifin@
5268       \directlua{%
5269         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5270           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5271           tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5272         end
5273       }%
5274     \fi
5275   \fi}
5276 \gdef\bbl@parsejalti{%
5277   \beginingroup
5278     \let\bbl@parsejalt\relax % To avoid infinite loop
5279     \edef\bbl@tempb{\fontid\font}%
5280     \bblar@nofswarn
5281     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5282     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5283     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5284     \addfontfeature{RawFeature+=jalt}%
5285     % \namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5286     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5287     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5288     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5289     \directlua{%
5290       for k, v in pairs(Babel.arabic.from) do
5291         if Babel.arabic.dest[k] and
5292           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5293           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5294             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5295         end
5296       end
5297     }%
5298   \endgroup}
5299 %
5300 \beginingroup
5301 \catcode`#=11
5302 \catcode`~=11
5303 \directlua{
5304
5305 Babel.arabic = Babel.arabic or {}
5306 Babel.arabic.from = {}

```

```

5307 Babel.arabic.dest = {}
5308 Babel.arabic.justify_factor = 0.95
5309 Babel.arabic.justify_enabled = true
5310
5311 function Babel.arabic.justify(head)
5312   if not Babel.arabic.justify_enabled then return head end
5313   for line in node.traverse_id(node.id'hlist', head) do
5314     Babel.arabic.justify_hlist(head, line)
5315   end
5316   return head
5317 end
5318
5319 function Babel.arabic.justify_hbox(head, gc, size, pack)
5320   local has_inf = false
5321   if Babel.arabic.justify_enabled and pack == 'exactly' then
5322     for n in node.traverse_id(12, head) do
5323       if n.stretch_order > 0 then has_inf = true end
5324     end
5325     if not has_inf then
5326       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5327     end
5328   end
5329   return head
5330 end
5331
5332 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5333   local d, new
5334   local k_list, k_item, pos_inline
5335   local width, width_new, full, k_curr, wt_pos, goal, shift
5336   local subst_done = false
5337   local elong_map = Babel.arabic.elong_map
5338   local last_line
5339   local GLYPH = node.id'glyph'
5340   local KASHIDA = Babel.attr_kashida
5341   local LOCALE = Babel.attr_locale
5342
5343   if line == nil then
5344     line = {}
5345     line.glue_sign = 1
5346     line.glue_order = 0
5347     line.head = head
5348     line.shift = 0
5349     line.width = size
5350   end
5351
5352   % Exclude last line. todo. But-- it discards one-word lines, too!
5353   % ? Look for glue = 12:15
5354   if (line.glue_sign == 1 and line.glue_order == 0) then
5355     elongs = {} % Stores elongated candidates of each line
5356     k_list = {} % And all letters with kashida
5357     pos_inline = 0 % Not yet used
5358
5359     for n in node.traverse_id(GLYPH, line.head) do
5360       pos_inline = pos_inline + 1 % To find where it is. Not used.
5361
5362       % Elongated glyphs
5363       if elong_map then
5364         local locale = node.get_attribute(n, LOCALE)
5365         if elong_map[locale] and elong_map[locale][n.font] and

```

```

5366         elong_map[locale][n.font][n.char] then
5367         table.insert(elongs, {node = n, locale = locale} )
5368         node.set_attribute(n.prev, KASHIDA, 0)
5369     end
5370 end
5371
5372 % Tatwil
5373 if Babel.kashida_wts then
5374     local k_wt = node.get_attribute(n, KASHIDA)
5375     if k_wt > 0 then % todo. parameter for multi inserts
5376         table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5377     end
5378 end
5379
5380 end % of node.traverse_id
5381
5382 if #elongs == 0 and #k_list == 0 then goto next_line end
5383 full = line.width
5384 shift = line.shift
5385 goal = full * Babel.arabic.justify_factor % A bit crude
5386 width = node.dimensions(line.head) % The 'natural' width
5387
5388 % == Elongated ==
5389 % Original idea taken from 'chickenize'
5390 while (#elongs > 0 and width < goal) do
5391     subst_done = true
5392     local x = #elongs
5393     local curr = elongs[x].node
5394     local oldchar = curr.char
5395     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5396     width = node.dimensions(line.head) % Check if the line is too wide
5397     % Substitute back if the line would be too wide and break:
5398     if width > goal then
5399         curr.char = oldchar
5400         break
5401     end
5402     % If continue, pop the just substituted node from the list:
5403     table.remove(elongs, x)
5404 end
5405
5406 % == Tatwil ==
5407 if #k_list == 0 then goto next_line end
5408
5409 width = node.dimensions(line.head) % The 'natural' width
5410 k_curr = #k_list
5411 wt_pos = 1
5412
5413 while width < goal do
5414     subst_done = true
5415     k_item = k_list[k_curr].node
5416     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5417         d = node.copy(k_item)
5418         d.char = 0x0640
5419         line.head, new = node.insert_after(line.head, k_item, d)
5420         width_new = node.dimensions(line.head)
5421         if width > goal or width == width_new then
5422             node.remove(line.head, new) % Better compute before
5423             break
5424         end

```

```

5425         width = width_new
5426     end
5427     if k_curr == 1 then
5428         k_curr = #k_list
5429         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5430     else
5431         k_curr = k_curr - 1
5432     end
5433 end
5434
5435 ::next_line::
5436
5437 % Must take into account marks and ins, see luatex manual.
5438 % Have to be executed only if there are changes. Investigate
5439 % what's going on exactly.
5440 if subst_done and not gc then
5441     d = node.hpack(line.head, full, 'exactly')
5442     d.shift = shift
5443     node.insert_before(head, line, d)
5444     node.remove(head, line)
5445 end
5446 end % if process line
5447 end
5448 }
5449 \endgroup
5450 \fi\fi % Arabic just block

```

13.7 Common stuff

```

5451 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5452 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
5453 \DisableBabelHook{babel-fontspec}
5454 <<Font selection>>

```

13.8 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.

```

5455 % TODO - to a lua file
5456 \directlua{
5457 Babel.script_blocks = {
5458   ['dflt'] = {},
5459   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5460              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5461   ['Armn'] = {{0x0530, 0x058F}},
5462   ['Beng'] = {{0x0980, 0x09FF}},
5463   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5464   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5465   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5466              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5467   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5468   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5469              {0xAB00, 0xAB2F}},
5470   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5471   % Don't follow strictly Unicode, which places some Coptic letters in

```

```

5472 % the 'Greek and Coptic' block
5473 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5474 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5475             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5476             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5477             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5478             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5479             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5480 ['Hebr'] = {{0x0590, 0x05FF}},
5481 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5482             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5483 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5484 ['Knda'] = {{0x0C80, 0x0CFF}},
5485 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5486             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5487             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5488 ['Lao'] = {{0x0E80, 0x0EFF}},
5489 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5490             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5491             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5492 ['Mahj'] = {{0x11150, 0x1117F}},
5493 ['Mlym'] = {{0x0D00, 0x0D7F}},
5494 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5495 ['Orya'] = {{0x0B00, 0x0B7F}},
5496 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5497 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5498 ['Taml'] = {{0x0B80, 0x0BFF}},
5499 ['Telu'] = {{0x0C00, 0x0C7F}},
5500 ['Tfng'] = {{0x2D30, 0x2D7F}},
5501 ['Thai'] = {{0x0E00, 0x0E7F}},
5502 ['Tibt'] = {{0x0F00, 0x0FFF}},
5503 ['Vaii'] = {{0xA500, 0xA63F}},
5504 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5505 }
5506
5507 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5508 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5509 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5510
5511 function Babel.locale_map(head)
5512   if not Babel.locale_mapped then return head end
5513
5514   local LOCALE = Babel.attr_locale
5515   local GLYPH = node.id('glyph')
5516   local inmath = false
5517   local toloc_save
5518   for item in node.traverse(head) do
5519     local toloc
5520     if not inmath and item.id == GLYPH then
5521       % Optimization: build a table with the chars found
5522       if Babel.chr_to_loc[item.char] then
5523         toloc = Babel.chr_to_loc[item.char]
5524       else
5525         for lc, maps in pairs(Babel.loc_to_scr) do
5526           for _, rg in pairs(maps) do
5527             if item.char >= rg[1] and item.char <= rg[2] then
5528               Babel.chr_to_loc[item.char] = lc
5529               toloc = lc
5530             break

```

```

5531         end
5532     end
5533 end
5534 end
5535 % Now, take action, but treat composite chars in a different
5536 % fashion, because they 'inherit' the previous locale. Not yet
5537 % optimized.
5538 if not toloc and
5539     (item.char >= 0x0300 and item.char <= 0x036F) or
5540     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5541     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5542     toloc = toloc_save
5543 end
5544 if toloc and toloc > -1 then
5545     if Babel.locale_props[toloc].lg then
5546         item.lang = Babel.locale_props[toloc].lg
5547         node.set_attribute(item, LOCALE, toloc)
5548     end
5549     if Babel.locale_props[toloc]['/'..item.font] then
5550         item.font = Babel.locale_props[toloc]['/'..item.font]
5551     end
5552     toloc_save = toloc
5553 end
5554 elseif not inmath and item.id == 7 then
5555     item.replace = item.replace and Babel.locale_map(item.replace)
5556     item.pre      = item.pre and Babel.locale_map(item.pre)
5557     item.post     = item.post and Babel.locale_map(item.post)
5558 elseif item.id == node.id'math' then
5559     inmath = (item.subtype == 0)
5560 end
5561 end
5562 return head
5563 end
5564 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

5565 \newcommand\babelcharproperty[1]{%
5566   \count@=#1\relax
5567   \ifvmode
5568     \expandafter\bbl@chprop
5569   \else
5570     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5571               vertical mode (preamble or between paragraphs)}%
5572     {See the manual for futher info}%
5573   \fi}
5574 \newcommand\bbl@chprop[3][\the\count@]{%
5575   \@tempcnta=#1\relax
5576   \bbl@ifunset\bbl@chprop@#2}%
5577   {\bbl@error{No property named '#2'. Allowed values are\\%
5578             direction (bc), mirror (bmg), and linebreak (lb)}%
5579    {See the manual for futher info}}%
5580   }%
5581 \loop
5582   \bbl@cs{chprop@#2}{#3}%
5583 \ifnum\count@<\@tempcnta
5584   \advance\count@\@ne
5585 \repeat}
5586 \def\bbl@chprop@direction#1{%

```



```

5587 \directlua{
5588   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5589   Babel.characters[\the\count@]['d'] = '#1'
5590 }}
5591 \let\bbl@chprop@bc\bbl@chprop@direction
5592 \def\bbl@chprop@mirror#1{%
5593   \directlua{
5594     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5595     Babel.characters[\the\count@]['m'] = '\number#1'
5596   }}
5597 \let\bbl@chprop@bmg\bbl@chprop@mirror
5598 \def\bbl@chprop@linebreak#1{%
5599   \directlua{
5600     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5601     Babel.cjk_characters[\the\count@]['c'] = '#1'
5602   }}
5603 \let\bbl@chprop@lb\bbl@chprop@linebreak
5604 \def\bbl@chprop@locale#1{%
5605   \directlua{
5606     Babel.chr_to_loc = Babel.chr_to_loc or {}
5607     Babel.chr_to_loc[\the\count@] =
5608       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5609   }}

```

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). The Lua code is below.

```

5610 \directlua{
5611   Babel.nohyphenation = \the\l@nohyphenation
5612 }

```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ becomes $\text{function}(m) \text{ return } m[1]..m[1]..'-' \text{ end}$, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m) \text{ return Babel.capt_map}(m[1],1) \text{ end}$, where the last argument identifies the mapping to be applied to $m[1]$. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of $@$, we just avoid this character in macro names (which explains the internal group, too).

```

5613 \begingroup
5614 \catcode`\~ = 12
5615 \catcode`\% = 12
5616 \catcode`\& = 14
5617 \gdef\babelposthyphenation#1#2#3{&%
5618   \bbl@activateposthyphen
5619   \begingroup
5620     \def\babeltempa{\bbl@add@list\babeltempb}&%
5621     \let\babeltempb\@empty
5622     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5623     \bbl@replace\bbl@tempa{,}{ ,}&%
5624     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5625       \bbl@ifsamestring{##1}{remove}&%
5626       {\bbl@add@list\babeltempb{nil}}&%
5627       {\directlua{
5628         local rep = {[#1]=]
5629         rep = rep:gsub('^s*(remove)s*$', 'remove = true')
5630         rep = rep:gsub('^s*(insert)s*', 'insert = true, ')
5631         rep = rep:gsub(' (no)s*=%s*([^\s,]*)', Babel.capture_func)
5632         rep = rep:gsub(' (pre)s*=%s*([^\s,]*)', Babel.capture_func)

```

```

5633         rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5634         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5635         tex.print([[\\string\\babeltempa{}}] .. rep .. [[{}]])
5636     }&&%
5637     \\directlua{
5638         local lbkr = Babel.linebreaking.replacements[1]
5639         local u = unicode.utf8
5640         local id = \\the\\csname l@#1\\endcsname
5641         &% Convert pattern:
5642         local patt = string.gsub([=[#2]=], '%s', '')
5643         if not u.find(patt, '()', nil, true) then
5644             patt = '()' .. patt .. '()'
5645         end
5646         patt = string.gsub(patt, '%(%)%^', '^()')
5647         patt = string.gsub(patt, '%$(%)', '()$')
5648         patt = u.gsub(patt, '{(.)}',
5649             function (n)
5650                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5651             end)
5652         patt = u.gsub(patt, '{(%x%x%x%x+)}',
5653             function (n)
5654                 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5655             end)
5656         lbkr[id] = lbkr[id] or {}
5657         table.insert(lbkr[id], { pattern = patt, replace = { \\babeltempb } })
5658     }&&%
5659     \\endgroup}
5660 % TODO. Copypaste pattern.
5661 \\gdef\\babelprehyphenation#1#2#3{&%
5662     \\bbl@activateprehyphen
5663     \\begingroup
5664     \\def\\babeltempa{\\bbl@add@list\\babeltempb}&%
5665     \\let\\babeltempb\\@empty
5666     \\def\\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5667     \\bbl@replace\\bbl@tempa{,}{ ,}&%
5668     \\expandafter\\bbl@foreach\\expandafter{\\bbl@tempa}{&%
5669         \\bbl@ifsamestring{##1}{remove}&%
5670         {\\bbl@add@list\\babeltempb{nil}}&%
5671     {\\directlua{
5672         local rep = [=[#1]=]
5673         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5674         rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
5675         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5676         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5677             'space = { ' .. '%2, %3, %4' .. ' }')
5678         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5679             'spacefactor = { ' .. '%2, %3, %4' .. ' }')
5680         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5681         tex.print([[\\string\\babeltempa{}}] .. rep .. [[{}]])
5682     }&&%
5683     \\directlua{
5684         local lbkr = Babel.linebreaking.replacements[0]
5685         local u = unicode.utf8
5686         local id = \\the\\csname bbl@id@#1\\endcsname
5687         &% Convert pattern:
5688         local patt = string.gsub([=[#2]=], '%s', '')
5689         local patt = string.gsub(patt, '|', ' ')
5690         if not u.find(patt, '()', nil, true) then
5691             patt = '()' .. patt .. '()'

```

```

5692     end
5693     &% patt = string.gsub(patt, '%(%)^', '^()')
5694     &% patt = string.gsub(patt, '([^^%])%$%$', '%1()$')
5695     patt = u.gsub(patt, '{(.)}',
5696         function (n)
5697             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5698         end)
5699     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5700         function (n)
5701             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5702         end)
5703     lbkr[id] = lbkr[id] or {}
5704     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5705 }&%
5706 \endgroup}
5707 \endgroup
5708 \def\bbbl@activateposthyphen{%
5709   \let\bbbl@activateposthyphen\relax
5710   \directlua{
5711     require('babel-transforms.lua')
5712     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5713   }}
5714 \def\bbbl@activateprehyphen{%
5715   \let\bbbl@activateprehyphen\relax
5716   \directlua{
5717     require('babel-transforms.lua')
5718     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5719   }}

```

13.9 Bidi

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.

```

5720 \def\bbbl@activate@preotf{%
5721   \let\bbbl@activate@preotf\relax % only once
5722   \directlua{
5723     Babel = Babel or {}
5724     %
5725     function Babel.pre_otfload_v(head)
5726       if Babel.numbers and Babel.digits_mapped then
5727         head = Babel.numbers(head)
5728       end
5729       if Babel.bidi_enabled then
5730         head = Babel.bidi(head, false, dir)
5731       end
5732       return head
5733     end
5734     %
5735     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5736       if Babel.numbers and Babel.digits_mapped then
5737         head = Babel.numbers(head)
5738       end
5739       if Babel.bidi_enabled then
5740         head = Babel.bidi(head, false, dir)
5741       end
5742       return head
5743     end

```

```

5744 %
5745 luatexbase.add_to_callback('pre_linebreak_filter',
5746   Babel.pre_otfload_v,
5747   'Babel.pre_otfload_v',
5748   luatexbase.priority_in_callback('pre_linebreak_filter',
5749     'luaotfload.node_processor') or nil)
5750 %
5751 luatexbase.add_to_callback('hpack_filter',
5752   Babel.pre_otfload_h,
5753   'Babel.pre_otfload_h',
5754   luatexbase.priority_in_callback('hpack_filter',
5755     'luaotfload.node_processor') or nil)
5756 }}

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option `bidir`.

```

5757 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5758   \let\bbl@beforeforeign\leavevmode
5759   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5760   \RequirePackage{luatexbase}
5761   \bbl@activate@preotf
5762   \directlua{
5763     require('babel-data-bidi.lua')
5764     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5765       require('babel-bidi-basic.lua')
5766     \or
5767       require('babel-bidi-basic-r.lua')
5768     \fi}
5769   % TODO - to locale_props, not as separate attribute
5770   \newattribute\bbl@attr@dir
5771   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5772   % TODO. I don't like it, hackish:
5773   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5774   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5775 \fi\fi
5776 \chardef\bbl@thetextdir\z@
5777 \chardef\bbl@thepardir\z@
5778 \def\bbl@getluadir#1{%
5779   \directlua{
5780     if tex.#1dir == 'TLT' then
5781       tex.sprint('0')
5782     elseif tex.#1dir == 'TRT' then
5783       tex.sprint('1')
5784     end}}
5785 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5786   \ifcase#3\relax
5787     \ifcase\bbl@getluadir{#1}\relax\else
5788       #2 TLT\relax
5789     \fi
5790   \else
5791     \ifcase\bbl@getluadir{#1}\relax
5792       #2 TRT\relax
5793     \fi
5794   \fi}
5795 \def\bbl@textdir#1{%
5796   \bbl@setluadir{text}\textdir{#1}%
5797   \chardef\bbl@thetextdir#1\relax
5798   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}

```

```

5799 \def\bbl@pdir#1{%
5800   \bbl@setluadir{par}\pdir{#1}%
5801   \chardef\bbl@thepdir#1\relax}
5802 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5803 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5804 \def\bbl@dirparastext{\pdir\the\textdir\relax}%    %%%
5805 %
5806 \ifnum\bbl@bidimode>\z@
5807   \def\bbl@mathboxdir{%
5808     \ifcase\bbl@thetextdir\relax
5809       \everyhbox{\bbl@mathboxdir@aux L}%
5810     \else
5811       \everyhbox{\bbl@mathboxdir@aux R}%
5812     \fi}
5813   \def\bbl@mathboxdir@aux#1{%
5814     \@ifnextchar\egroup{{}\textdir T#1T\relax}}
5815   \frozen@everymath\expandafter{%
5816     \expandafter\bbl@mathboxdir\the\frozen@everymath}
5817   \frozen@everydisplay\expandafter{%
5818     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5819 \fi

```

13.10 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.

```

5820 \bbl@trace{Redefinitions for bidi layout}
5821 \ifx\@eqnnum\@undefined\else
5822   \ifx\bbl@attr@dir\@undefined\else
5823     \edef\@eqnnum{%
5824       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5825       \unexpanded\expandafter{\@eqnnum}}
5826   \fi
5827 \fi
5828 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5829 \ifnum\bbl@bidimode>\z@
5830   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5831     \bbl@exp{%
5832       \mathdir\the\bodydir
5833       #1%           Once entered in math, set boxes to restore values
5834       \<ifmmode>%
5835       \everyvbox{%
5836         \the\everyvbox
5837         \bodydir\the\bodydir
5838         \mathdir\the\mathdir
5839         \everyhbox{\the\everyhbox}%
5840         \everyvbox{\the\everyvbox}}%
5841       \everyhbox{%

```

```

5842         \the\everyhbox
5843         \bodydir\the\bodydir
5844         \mathdir\the\mathdir
5845         \everyhbox{\the\everyhbox}%
5846         \everyvbox{\the\everyvbox}%
5847     \<fi>}}%
5848 \def\@hangfrom#1{%
5849     \setbox\@tempboxa\hbox{#{1}}%
5850     \hangindent\wd\@tempboxa
5851     \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
5852         \shapemode\@ne
5853     \fi
5854     \noindent\box\@tempboxa}
5855 \fi
5856 \IfBabelLayout{tabular}
5857 {\let\bbbl@OL@tabular\@tabular
5858  \bbbl@replace\@tabular{$}\{\bbbl@nextfake$}%
5859  \let\bbbl@NL@tabular\@tabular
5860  \AtBeginDocument{%
5861      \ifx\bbbl@NL@tabular\@tabular\else
5862          \bbbl@replace\@tabular{$}\{\bbbl@nextfake$}%
5863          \let\bbbl@NL@tabular\@tabular
5864      \fi}}
5865 {}
5866 \IfBabelLayout{lists}
5867 {\let\bbbl@OL@list\list
5868  \bbbl@sreplace\list{\parshape}\{\bbbl@listparshape}%
5869  \let\bbbl@NL@list\list
5870  \def\bbbl@listparshape#1#2#3{%
5871      \parshape #1 #2 #3 %
5872      \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
5873          \shapemode\tw@
5874      \fi}}
5875 {}
5876 \IfBabelLayout{graphics}
5877 {\let\bbbl@pictresetdir\relax
5878  \def\bbbl@pictsetdir#1{%
5879      \ifcase\bbbl@thetextdir
5880      \let\bbbl@pictresetdir\relax
5881      \else
5882          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5883              \or\textdir TLT
5884              \else\bodydir TLT \textdir TLT
5885          \fi
5886          % \(\text|par)dir required in pgf:
5887          \def\bbbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5888      \fi}%
5889  \ifx\AddToHook\undefined\else
5890      \AddToHook{env/picture/begin}\{\bbbl@pictsetdir\tw@}%
5891      \directlua{
5892          Babel.get_picture_dir = true
5893          Babel.picture_has_bidi = 0
5894          function Babel.picture_dir (head)
5895              if not Babel.get_picture_dir then return head end
5896              for item in node.traverse(head) do
5897                  if item.id == node.id'glyph' then
5898                      local itemchar = item.char
5899                      % TODO. Copypaste pattern from Babel.bidi (-r)
5900                      local chardata = Babel.characters[itemchar]

```

```

5901         local dir = chardata and chardata.d or nil
5902         if not dir then
5903             for nn, et in ipairs(Babel.ranges) do
5904                 if itemchar < et[1] then
5905                     break
5906                 elseif itemchar <= et[2] then
5907                     dir = et[3]
5908                     break
5909                 end
5910             end
5911         end
5912         if dir and (dir == 'al' or dir == 'r') then
5913             Babel.picture_has_bidi = 1
5914         end
5915     end
5916 end
5917 return head
5918 end
5919 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5920     "Babel.picture_dir")
5921 }%
5922 \AtBeginDocument{%
5923     \long\def\put(#1,#2)#3{%
5924         \@killglue
5925         % Try:
5926         \ifx\bbl@pictresetdir\relax
5927             \def\bbl@tempc{0}%
5928         \else
5929             \directlua{
5930                 Babel.get_picture_dir = true
5931                 Babel.picture_has_bidi = 0
5932             }%
5933             \setbox\z@\hb@xt@\z@{%
5934                 \@defaultunitsset\@tempdimc{#1}\unitlength
5935                 \kern\@tempdimc
5936                 #3\hss}%
5937             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
5938         \fi
5939         % Do:
5940         \@defaultunitsset\@tempdimc{#2}\unitlength
5941         \raise\@tempdimc\hb@xt@\z@{%
5942             \@defaultunitsset\@tempdimc{#1}\unitlength
5943             \kern\@tempdimc
5944             {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
5945         \ignorespaces}%
5946         \MakeRobust\put}%
5947 \fi
5948 \AtBeginDocument
5949 {
5950     \ifx\tikz@atbegin@node\undefined\else
5951         \ifx\AddToHook\undefined\else % TODO. Still tentative.
5952             \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
5953             \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
5954         \fi
5955         \let\bbl@OL\pgfpicture\pgfpicture
5956         \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
5957         {\bbl@pictsetdir\z@\pgfpicturetrue}%
5958         \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
5959         \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5960         \bbl@sreplace\tikz{\beginpgroup}%

```

```

5960      {\begingroup\bbl@pictsetdir\tw}%
5961      \fi
5962      \ifx\AddToHook\@undefined\else
5963        \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
5964        \fi
5965      }}
5966  {}

```

```

5967 \IfBabelLayout{counters}%
5968   {\let\bbl@OL@@textsuperscript\@textsuperscript
5969     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5970     \let\bbl@latinarabic=\@arabic
5971     \let\bbl@OL@@arabic\@arabic
5972     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5973     \ifpackagewith{babel}{\bidi=default}%
5974     {\let\bbl@asciroman=\@roman
5975       \let\bbl@OL@@roman\@roman
5976       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5977       \let\bbl@asciRoman=\@Roman
5978       \let\bbl@OL@@roman\@Roman
5979       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
5980       \let\bbl@OL@labelenumii\labelenumii
5981       \def\labelenumii{}\theenumii{}%
5982       \let\bbl@OL@p@enumiii\p@enumiii
5983       \def\p@enumiii{\p@enumii}\theenumii{}}{}{}
5984 \langle\langle Footnote changes\rangle\rangle
5985 \IfBabelLayout{footnotes}%
5986   {\let\bbl@OL@footnote\footnote
5987     \BabelFootnote\footnote\language{}{}}%
5988     \BabelFootnote\localfootnote\language{}{}}%
5989     \BabelFootnote\mainfootnote{}{}}{}
5990   {}

```

```

5991 \IfBabelLayout{extras}%
5992   {\let\bb1@0L@underline\underline
5993    \bb1@sreplace\underline{$\@@underline}\bb1@nextfake$\@@underline}%
5994    \let\bb1@0L@LaTeX2e\LaTeX2e
5995    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5996      \if b\expandafter\@car\@f@series\@nil\boldmath\fi
5997      \babselsublr}%
5998      \LaTeX\kern.15em2\bb1@nextfake$_{\textstyle\varepsilon}$}}
5999  {}
6000 </luatex>

```

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).

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.

```

6001 (*transforms)
6002 Babel.linebreaking.replacements = {}
6003 Babel.linebreaking.replacements[0] = {} -- pre
6004 Babel.linebreaking.replacements[1] = {} -- post
6005
6006 -- Discretionaries contain strings as nodes
6007 function Babel.str_to_nodes(fn, matches, base)
6008   local n, head, last
6009   if fn == nil then return nil end
6010   for s in string.utfvalues(fn(matches)) do
6011     if base.id == 7 then
6012       base = base.replace
6013     end
6014     n = node.copy(base)
6015     n.char = s
6016     if not head then
6017       head = n
6018     else
6019       last.next = n
6020     end
6021     last = n
6022   end
6023   return head
6024 end
6025
6026 Babel.fetch_subtext = {}
6027
6028 Babel.ignore_pre_char = function(node)
6029   return (node.lang == Babel.nohyphenation)
6030 end
6031
6032 -- Merging both functions doesn't seem feasible, because there are too
6033 -- many differences.
6034 Babel.fetch_subtext[0] = function(head)
6035   local word_string = ''
6036   local word_nodes = {}
6037   local lang
6038   local item = head
6039   local inmath = false
6040
6041   while item do
6042
6043     if item.id == 11 then
6044       inmath = (item.subtype == 0)
6045     end
6046
6047     if inmath then
6048       -- pass
6049     elseif item.id == 29 then
6050       local locale = node.get_attribute(item, Babel.attr_locale)
6051
6052       if lang == locale or lang == nil then
6053         lang = lang or locale
6054         if Babel.ignore_pre_char(item) then

```

```

6056         word_string = word_string .. Babel.us_char
6057     else
6058         word_string = word_string .. unicode.utf8.char(item.char)
6059     end
6060     word_nodes[#word_nodes+1] = item
6061 else
6062     break
6063 end
6064
6065 elseif item.id == 12 and item.subtype == 13 then
6066     word_string = word_string .. ' '
6067     word_nodes[#word_nodes+1] = item
6068
6069 -- Ignore leading unrecognized nodes, too.
6070 elseif word_string ~= '' then
6071     word_string = word_string .. Babel.us_char
6072     word_nodes[#word_nodes+1] = item -- Will be ignored
6073 end
6074
6075 item = item.next
6076 end
6077
6078 -- Here and above we remove some trailing chars but not the
6079 -- corresponding nodes. But they aren't accessed.
6080 if word_string:sub(-1) == ' ' then
6081     word_string = word_string:sub(1,-2)
6082 end
6083 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6084 return word_string, word_nodes, item, lang
6085 end
6086
6087 Babel.fetch_subtext[1] = function(head)
6088     local word_string = ''
6089     local word_nodes = {}
6090     local lang
6091     local item = head
6092     local inmath = false
6093
6094     while item do
6095
6096         if item.id == 11 then
6097             inmath = (item.subtype == 0)
6098         end
6099
6100         if inmath then
6101             -- pass
6102
6103         elseif item.id == 29 then
6104             if item.lang == lang or lang == nil then
6105                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6106                     lang = lang or item.lang
6107                     word_string = word_string .. unicode.utf8.char(item.char)
6108                     word_nodes[#word_nodes+1] = item
6109                 end
6110             else
6111                 break
6112             end
6113
6114         elseif item.id == 7 and item.subtype == 2 then

```

```

6115     word_string = word_string .. '='
6116     word_nodes[#word_nodes+1] = item
6117
6118     elseif item.id == 7 and item.subtype == 3 then
6119         word_string = word_string .. '|'
6120         word_nodes[#word_nodes+1] = item
6121
6122     -- (1) Go to next word if nothing was found, and (2) implicitly
6123     -- remove leading USs.
6124     elseif word_string == '' then
6125         -- pass
6126
6127     -- This is the responsible for splitting by words.
6128     elseif (item.id == 12 and item.subtype == 13) then
6129         break
6130
6131     else
6132         word_string = word_string .. Babel.us_char
6133         word_nodes[#word_nodes+1] = item -- Will be ignored
6134     end
6135
6136     item = item.next
6137 end
6138
6139 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6140 return word_string, word_nodes, item, lang
6141 end
6142
6143 function Babel.pre_hyphenate_replace(head)
6144     Babel.hyphenate_replace(head, 0)
6145 end
6146
6147 function Babel.post_hyphenate_replace(head)
6148     Babel.hyphenate_replace(head, 1)
6149 end
6150
6151 Babel.us_char = string.char(31)
6152
6153 function Babel.hyphenate_replace(head, mode)
6154     local u = unicode.utf8
6155     local lbkr = Babel.linebreaking.replacements[mode]
6156
6157     local word_head = head
6158
6159     while true do -- for each subtext block
6160
6161         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6162
6163         if Babel.debug then
6164             print()
6165             print((mode == 0) and '@@@<' or '@@@>', w)
6166         end
6167
6168         if nw == nil and w == '' then break end
6169
6170         if not lang then goto next end
6171         if not lbkr[lang] then goto next end
6172
6173         -- For each saved (pre|post)hyphenation. TODO. Reconsider how

```

```

6174 -- loops are nested.
6175 for k=1, #lbr[lang] do
6176     local p = lbr[lang][k].pattern
6177     local r = lbr[lang][k].replace
6178
6179     if Babel.debug then
6180         print('*****', p, mode)
6181     end
6182
6183     -- This variable is set in some cases below to the first *byte*
6184     -- after the match, either as found by u.match (faster) or the
6185     -- computed position based on sc if w has changed.
6186     local last_match = 0
6187     local step = 0
6188
6189     -- For every match.
6190     while true do
6191         if Babel.debug then
6192             print('====')
6193         end
6194         local new -- used when inserting and removing nodes
6195
6196         local matches = { u.match(w, p, last_match) }
6197
6198         if #matches < 2 then break end
6199
6200         -- Get and remove empty captures (with ()'s, which return a
6201         -- number with the position), and keep actual captures
6202         -- (from (...)), if any, in matches.
6203         local first = table.remove(matches, 1)
6204         local last = table.remove(matches, #matches)
6205         -- Non re-fetched substrings may contain \31, which separates
6206         -- subsubstrings.
6207         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6208
6209         local save_last = last -- with A()BC()D, points to D
6210
6211         -- Fix offsets, from bytes to unicode. Explained above.
6212         first = u.len(w:sub(1, first-1)) + 1
6213         last = u.len(w:sub(1, last-1)) -- now last points to C
6214
6215         -- This loop stores in n small table the nodes
6216         -- corresponding to the pattern. Used by 'data' to provide a
6217         -- predictable behavior with 'insert' (now w_nodes is modified on
6218         -- the fly), and also access to 'remove'd nodes.
6219         local sc = first-1 -- Used below, too
6220         local data_nodes = {}
6221
6222         for q = 1, last-first+1 do
6223             data_nodes[q] = w_nodes[sc+q]
6224         end
6225
6226         -- This loop traverses the matched substring and takes the
6227         -- corresponding action stored in the replacement list.
6228         -- sc = the position in substr nodes / string
6229         -- rc = the replacement table index
6230         local rc = 0
6231
6232         while rc < last-first+1 do -- for each replacement

```

```

6233     if Babel.debug then
6234         print('.....', rc + 1)
6235     end
6236     sc = sc + 1
6237     rc = rc + 1
6238
6239     if Babel.debug then
6240         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6241         local ss = ''
6242         for itt in node.traverse(head) do
6243             if itt.id == 29 then
6244                 ss = ss .. unicode.utf8.char(itt.char)
6245             else
6246                 ss = ss .. '{' .. itt.id .. '}'
6247             end
6248         end
6249         print('*****', ss)
6250
6251     end
6252
6253     local crep = r[rc]
6254     local item = w_nodes[sc]
6255     local item_base = item
6256     local placeholder = Babel.us_char
6257     local d
6258
6259     if crep and crep.data then
6260         item_base = data_nodes[crep.data]
6261     end
6262
6263     if crep then
6264         step = crep.step or 0
6265     end
6266
6267     if crep and next(crep) == nil then -- = {}
6268         last_match = save_last -- Optimization
6269         goto next
6270
6271     elseif crep == nil or crep.remove then
6272         node.remove(head, item)
6273         table.remove(w_nodes, sc)
6274         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6275         sc = sc - 1 -- Nothing has been inserted.
6276         last_match = utf8.offset(w, sc+1+step)
6277         goto next
6278
6279     elseif crep and crep.kashida then -- Experimental
6280         node.set_attribute(item,
6281             Babel.attr_kashida,
6282             crep.kashida)
6283         last_match = utf8.offset(w, sc+1+step)
6284         goto next
6285
6286     elseif crep and crep.string then
6287         local str = crep.string(matches)
6288         if str == '' then -- Gather with nil
6289             node.remove(head, item)
6290             table.remove(w_nodes, sc)
6291             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)

```

```

6292         sc = sc - 1 -- Nothing has been inserted.
6293     else
6294         local loop_first = true
6295         for s in string.utfvalues(str) do
6296             d = node.copy(item_base)
6297             d.char = s
6298             if loop_first then
6299                 loop_first = false
6300                 head, new = node.insert_before(head, item, d)
6301                 if sc == 1 then
6302                     word_head = head
6303                 end
6304                 w_nodes[sc] = d
6305                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6306             else
6307                 sc = sc + 1
6308                 head, new = node.insert_before(head, item, d)
6309                 table.insert(w_nodes, sc, new)
6310                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6311             end
6312             if Babel.debug then
6313                 print('.....', 'str')
6314                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6315             end
6316             end -- for
6317             node.remove(head, item)
6318         end -- if ''
6319         last_match = utf8.offset(w, sc+1+step)
6320         goto next
6321
6322     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6323         d = node.new(7, 0) -- (disc, discretionary)
6324         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6325         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6326         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6327         d.attr = item_base.attr
6328         if crep.pre == nil then -- TeXbook p96
6329             d.penalty = crep.penalty or tex.hyphenpenalty
6330         else
6331             d.penalty = crep.penalty or tex.exhyphenpenalty
6332         end
6333         placeholder = '|'
6334         head, new = node.insert_before(head, item, d)
6335
6336     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6337         -- ERROR
6338
6339     elseif crep and crep.penalty then
6340         d = node.new(14, 0) -- (penalty, userpenalty)
6341         d.attr = item_base.attr
6342         d.penalty = crep.penalty
6343         head, new = node.insert_before(head, item, d)
6344
6345     elseif crep and crep.space then
6346         -- 655360 = 10 pt = 10 * 65536 sp
6347         d = node.new(12, 13) -- (glue, spaceskip)
6348         local quad = font.getfont(item_base.font).size or 655360
6349         node.setglue(d, crep.space[1] * quad,
6350                     crep.space[2] * quad,

```

```

6351             crep.space[3] * quad)
6352         if mode == 0 then
6353             placeholder = ' '
6354         end
6355         head, new = node.insert_before(head, item, d)
6356
6357     elseif crep and crep.spacefactor then
6358         d = node.new(12, 13)      -- (glue, spaceskip)
6359         local base_font = font.getfont(item_base.font)
6360         node.setglue(d,
6361             crep.spacefactor[1] * base_font.parameters['space'],
6362             crep.spacefactor[2] * base_font.parameters['space_stretch'],
6363             crep.spacefactor[3] * base_font.parameters['space_shrink'])
6364         if mode == 0 then
6365             placeholder = ' '
6366         end
6367         head, new = node.insert_before(head, item, d)
6368
6369     elseif mode == 0 and crep and crep.space then
6370         -- ERROR
6371
6372     end -- ie replacement cases
6373
6374     -- Shared by disc, space and penalty.
6375     if sc == 1 then
6376         word_head = head
6377     end
6378     if crep.insert then
6379         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6380         table.insert(w_nodes, sc, new)
6381         last = last + 1
6382     else
6383         w_nodes[sc] = d
6384         node.remove(head, item)
6385         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6386     end
6387
6388     last_match = utf8.offset(w, sc+1+step)
6389
6390     ::next::
6391
6392     end -- for each replacement
6393
6394     if Babel.debug then
6395         print('.....', '/')
6396         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6397     end
6398
6399     end -- for match
6400
6401     end -- for patterns
6402
6403     ::next::
6404     word_head = nw
6405 end -- for substring
6406 return head
6407 end
6408
6409 -- This table stores capture maps, numbered consecutively

```

```

6410 Babel.capture_maps = {}
6411
6412 -- The following functions belong to the next macro
6413 function Babel.capture_func(key, cap)
6414   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6415   local cnt
6416   local u = unicode.utf8
6417   ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
6418   if cnt == 0 then
6419     ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6420       function (n)
6421         return u.char(tonumber(n, 16))
6422       end)
6423   end
6424   ret = ret:gsub("%[%[%]]%.", '')
6425   ret = ret:gsub("%.%.%[%[%]]%", '')
6426   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6427 end
6428
6429 function Babel.capt_map(from, mapno)
6430   return Babel.capture_maps[mapno][from] or from
6431 end
6432
6433 -- Handle the {n|abc|ABC} syntax in captures
6434 function Babel.capture_func_map(capno, from, to)
6435   local u = unicode.utf8
6436   from = u.gsub(from, '{{(%x%x%x%x+)}',
6437     function (n)
6438       return u.char(tonumber(n, 16))
6439     end)
6440   to = u.gsub(to, '{{(%x%x%x%x+)}',
6441     function (n)
6442       return u.char(tonumber(n, 16))
6443     end)
6444   local froms = {}
6445   for s in string.utfcharacters(from) do
6446     table.insert(froms, s)
6447   end
6448   local cnt = 1
6449   table.insert(Babel.capture_maps, {})
6450   local mlen = table.getn(Babel.capture_maps)
6451   for s in string.utfcharacters(to) do
6452     Babel.capture_maps[mlen][froms[cnt]] = s
6453     cnt = cnt + 1
6454   end
6455   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6456     (mlen) .. ").." .. "["
6457 end
6458
6459 -- Create/Extend reversed sorted list of kashida weights:
6460 function Babel.capture_kashida(key, wt)
6461   wt = tonumber(wt)
6462   if Babel.kashida_wts then
6463     for p, q in ipairs(Babel.kashida_wts) do
6464       if wt == q then
6465         break
6466       elseif wt > q then
6467         table.insert(Babel.kashida_wts, p, wt)
6468         break

```



```

6469     elseif table.getn(Babel.kashida_wts) == p then
6470         table.insert(Babel.kashida_wts, wt)
6471     end
6472 end
6473 else
6474     Babel.kashida_wts = { wt }
6475 end
6476 return 'kashida = ' .. wt
6477 end
6478 </transforms>

```

13.12 Lua: 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.

```

6479 <(*basic-r)
6480 Babel = Babel or {}
6481
6482 Babel.bidi_enabled = true
6483
6484 require('babel-data-bidi.lua')
6485
6486 local characters = Babel.characters
6487 local ranges = Babel.ranges

```

```

6488
6489 local DIR = node.id("dir")
6490
6491 local function dir_mark(head, from, to, outer)
6492   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6493   local d = node.new(DIR)
6494   d.dir = '+' .. dir
6495   node.insert_before(head, from, d)
6496   d = node.new(DIR)
6497   d.dir = '-' .. dir
6498   node.insert_after(head, to, d)
6499 end
6500
6501 function Babel.bidi(head, ispar)
6502   local first_n, last_n          -- first and last char with nums
6503   local last_es                  -- an auxiliary 'last' used with nums
6504   local first_d, last_d          -- first and last char in L/R block
6505   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):

```

6506   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6507   local strong_lr = (strong == 'l') and 'l' or 'r'
6508   local outer = strong
6509
6510   local new_dir = false
6511   local first_dir = false
6512   local inmath = false
6513
6514   local last_lr
6515
6516   local type_n = ''
6517
6518   for item in node.traverse(head) do
6519
6520     -- three cases: glyph, dir, otherwise
6521     if item.id == node.id'glyph'
6522       or (item.id == 7 and item.subtype == 2) then
6523
6524       local itemchar
6525       if item.id == 7 and item.subtype == 2 then
6526         itemchar = item.replace.char
6527       else
6528         itemchar = item.char
6529       end
6530       local chardata = characters[itemchar]
6531       dir = chardata and chardata.d or nil
6532       if not dir then
6533         for nn, et in ipairs(ranges) do
6534           if itemchar < et[1] then
6535             break
6536           elseif itemchar <= et[2] then
6537             dir = et[3]
6538             break
6539           end
6540         end
6541       end
6542       dir = dir or 'l'

```

```
6543     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).

```
6544     if new_dir then
6545         attr_dir = 0
6546         for at in node.traverse(item.attr) do
6547             if at.number == Babel.attr_dir then
6548                 attr_dir = at.value % 3
6549             end
6550         end
6551         if attr_dir == 1 then
6552             strong = 'r'
6553         elseif attr_dir == 2 then
6554             strong = 'al'
6555         else
6556             strong = 'l'
6557         end
6558         strong_lr = (strong == 'l') and 'l' or 'r'
6559         outer = strong_lr
6560         new_dir = false
6561     end
6562
6563     if dir == 'nsm' then dir = strong end          -- W1
```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```
6564     dir_real = dir          -- We need dir_real to set strong below
6565     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:

```
6566     if strong == 'al' then
6567         if dir == 'en' then dir = 'an' end          -- W2
6568         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6569         strong_lr = 'r'          -- W3
6570     end
```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```
6571     elseif item.id == node.id'dir' and not inmath then
6572         new_dir = true
6573         dir = nil
6574     elseif item.id == node.id'math' then
6575         inmath = (item.subtype == 0)
6576     else
6577         dir = nil          -- Not a char
6578     end
```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```
6579     if dir == 'en' or dir == 'an' or dir == 'et' then
6580         if dir ~= 'et' then
6581             type_n = dir
6582         end
6583         first_n = first_n or item
```

```

6584     last_n = last_es or item
6585     last_es = nil
6586 elseif dir == 'es' and last_n then -- W3+W6
6587     last_es = item
6588 elseif dir == 'cs' then           -- it's right - do nothing
6589 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6590     if strong_lr == 'r' and type_n ~= '' then
6591         dir_mark(head, first_n, last_n, 'r')
6592     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6593         dir_mark(head, first_n, last_n, 'r')
6594         dir_mark(head, first_d, last_d, outer)
6595         first_d, last_d = nil, nil
6596     elseif strong_lr == 'l' and type_n ~= '' then
6597         last_d = last_n
6598     end
6599     type_n = ''
6600     first_n, last_n = nil, nil
6601 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:

```

6602     if dir == 'l' or dir == 'r' then
6603         if dir ~= outer then
6604             first_d = first_d or item
6605             last_d = item
6606         elseif first_d and dir ~= strong_lr then
6607             dir_mark(head, first_d, last_d, outer)
6608             first_d, last_d = nil, nil
6609         end
6610     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.

```

6611     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6612         item.char = characters[item.char] and
6613             characters[item.char].m or item.char
6614     elseif (dir or new_dir) and last_lr ~= item then
6615         local mir = outer .. strong_lr .. (dir or outer)
6616         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6617             for ch in node.traverse(node.next(last_lr)) do
6618                 if ch == item then break end
6619                 if ch.id == node.id'glyph' and characters[ch.char] then
6620                     ch.char = characters[ch.char].m or ch.char
6621                 end
6622             end
6623         end
6624     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).

```

6625     if dir == 'l' or dir == 'r' then
6626         last_lr = item
6627         strong = dir_real           -- Don't search back - best save now
6628         strong_lr = (strong == 'l') and 'l' or 'r'

```

```

6629     elseif new_dir then
6630         last_lr = nil
6631     end
6632 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

6633 if last_lr and outer == 'r' then
6634     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6635         if characters[ch.char] then
6636             ch.char = characters[ch.char].m or ch.char
6637         end
6638     end
6639 end
6640 if first_n then
6641     dir_mark(head, first_n, last_n, outer)
6642 end
6643 if first_d then
6644     dir_mark(head, first_d, last_d, outer)
6645 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

6646 return node.prev(head) or head
6647 end
6648 </basic-r>

```

And here the Lua code for bidi=basic:

```

6649 <(*basic)
6650 Babel = Babel or {}
6651
6652 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6653
6654 Babel.fontmap = Babel.fontmap or {}
6655 Babel.fontmap[0] = {}      -- l
6656 Babel.fontmap[1] = {}      -- r
6657 Babel.fontmap[2] = {}      -- al/an
6658
6659 Babel.bidi_enabled = true
6660 Babel.mirroring_enabled = true
6661
6662 require('babel-data-bidi.lua')
6663
6664 local characters = Babel.characters
6665 local ranges = Babel.ranges
6666
6667 local DIR = node.id('dir')
6668 local GLYPH = node.id('glyph')
6669
6670 local function insert_implicit(head, state, outer)
6671     local new_state = state
6672     if state.sim and state.eim and state.sim ~= state.eim then
6673         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6674         local d = node.new(DIR)
6675         d.dir = '+' .. dir
6676         node.insert_before(head, state.sim, d)
6677         local d = node.new(DIR)
6678         d.dir = '-' .. dir
6679         node.insert_after(head, state.eim, d)
6680     end

```

```

6681 new_state.sim, new_state.eim = nil, nil
6682 return head, new_state
6683 end
6684
6685 local function insert_numeric(head, state)
6686   local new
6687   local new_state = state
6688   if state.san and state.ean and state.san ~= state.ean then
6689     local d = node.new(DIR)
6690     d.dir = '+TLT'
6691     _, new = node.insert_before(head, state.san, d)
6692     if state.san == state.sim then state.sim = new end
6693     local d = node.new(DIR)
6694     d.dir = '-TLT'
6695     _, new = node.insert_after(head, state.ean, d)
6696     if state.ean == state.eim then state.eim = new end
6697   end
6698   new_state.san, new_state.ean = nil, nil
6699   return head, new_state
6700 end
6701
6702 -- TODO - \hbox with an explicit dir can lead to wrong results
6703 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6704 -- was made to improve the situation, but the problem is the 3-dir
6705 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6706 -- well.
6707
6708 function Babel.bidi(head, ispar, hdir)
6709   local d -- d is used mainly for computations in a loop
6710   local prev_d = ''
6711   local new_d = false
6712
6713   local nodes = {}
6714   local outer_first = nil
6715   local inmath = false
6716
6717   local glue_d = nil
6718   local glue_i = nil
6719
6720   local has_en = false
6721   local first_et = nil
6722
6723   local ATDIR = Babel.attr_dir
6724
6725   local save_outer
6726   local temp = node.get_attribute(head, ATDIR)
6727   if temp then
6728     temp = temp % 3
6729     save_outer = (temp == 0 and 'l') or
6730                  (temp == 1 and 'r') or
6731                  (temp == 2 and 'al')
6732   elseif ispar then -- Or error? Shouldn't happen
6733     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6734   else -- Or error? Shouldn't happen
6735     save_outer = ('TRT' == hdir) and 'r' or 'l'
6736   end
6737   -- when the callback is called, we are just _after_ the box,
6738   -- and the textdir is that of the surrounding text
6739   -- if not ispar and hdir ~= tex.textdir then

```

```

6740 -- save_outer = ('TRT' == hdir) and 'r' or 'l'
6741 -- end
6742 local outer = save_outer
6743 local last = outer
6744 -- 'al' is only taken into account in the first, current loop
6745 if save_outer == 'al' then save_outer = 'r' end
6746
6747 local fontmap = Babel.fontmap
6748
6749 for item in node.traverse(head) do
6750
6751     -- In what follows, #node is the last (previous) node, because the
6752     -- current one is not added until we start processing the neutrals.
6753
6754     -- three cases: glyph, dir, otherwise
6755     if item.id == GLYPH
6756         or (item.id == 7 and item.subtype == 2) then
6757
6758         local d_font = nil
6759         local item_r
6760         if item.id == 7 and item.subtype == 2 then
6761             item_r = item.replace -- automatic discs have just 1 glyph
6762         else
6763             item_r = item
6764         end
6765         local chardata = characters[item_r.char]
6766         d = chardata and chardata.d or nil
6767         if not d or d == 'nsm' then
6768             for nn, et in ipairs(ranges) do
6769                 if item_r.char < et[1] then
6770                     break
6771                 elseif item_r.char <= et[2] then
6772                     if not d then d = et[3]
6773                     elseif d == 'nsm' then d_font = et[3]
6774                     end
6775                     break
6776                 end
6777             end
6778         end
6779         d = d or 'l'
6780
6781         -- A short 'pause' in bidi for mapfont
6782         d_font = d_font or d
6783         d_font = (d_font == 'l' and 0) or
6784             (d_font == 'nsm' and 0) or
6785             (d_font == 'r' and 1) or
6786             (d_font == 'al' and 2) or
6787             (d_font == 'an' and 2) or nil
6788         if d_font and fontmap and fontmap[d_font][item_r.font] then
6789             item_r.font = fontmap[d_font][item_r.font]
6790         end
6791
6792         if new_d then
6793             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6794             if inmath then
6795                 attr_d = 0
6796             else
6797                 attr_d = node.get_attribute(item, ATDIR)
6798                 attr_d = attr_d % 3

```

```

6799         end
6800         if attr_d == 1 then
6801             outer_first = 'r'
6802             last = 'r'
6803         elseif attr_d == 2 then
6804             outer_first = 'r'
6805             last = 'al'
6806         else
6807             outer_first = 'l'
6808             last = 'l'
6809         end
6810         outer = last
6811         has_en = false
6812         first_et = nil
6813         new_d = false
6814     end
6815
6816     if glue_d then
6817         if (d == 'l' and 'l' or 'r') ~= glue_d then
6818             table.insert(nodes, {glue_i, 'on', nil})
6819         end
6820         glue_d = nil
6821         glue_i = nil
6822     end
6823
6824     elseif item.id == DIR then
6825         d = nil
6826         new_d = true
6827
6828     elseif item.id == node.id'glue' and item.subtype == 13 then
6829         glue_d = d
6830         glue_i = item
6831         d = nil
6832
6833     elseif item.id == node.id'math' then
6834         inmath = (item.subtype == 0)
6835
6836     else
6837         d = nil
6838     end
6839
6840     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6841     if last == 'al' and d == 'en' then
6842         d = 'an'          -- W3
6843     elseif last == 'al' and (d == 'et' or d == 'es') then
6844         d = 'on'          -- W6
6845     end
6846
6847     -- EN + CS/ES + EN      -- W4
6848     if d == 'en' and #nodes >= 2 then
6849         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6850             and nodes[#nodes-1][2] == 'en' then
6851             nodes[#nodes][2] = 'en'
6852         end
6853     end
6854
6855     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
6856     if d == 'an' and #nodes >= 2 then
6857         if (nodes[#nodes][2] == 'cs')

```



```

6858         and nodes[#nodes-1][2] == 'an' then
6859             nodes[#nodes][2] = 'an'
6860         end
6861     end
6862
6863     -- ET/EN          -- W5 + W7->l / W6->on
6864     if d == 'et' then
6865         first_et = first_et or (#nodes + 1)
6866     elseif d == 'en' then
6867         has_en = true
6868         first_et = first_et or (#nodes + 1)
6869     elseif first_et then      -- d may be nil here !
6870         if has_en then
6871             if last == 'l' then
6872                 temp = 'l'      -- W7
6873             else
6874                 temp = 'en'     -- W5
6875             end
6876         else
6877             temp = 'on'        -- W6
6878         end
6879         for e = first_et, #nodes do
6880             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6881         end
6882         first_et = nil
6883         has_en = false
6884     end
6885
6886     -- Force mathdir in math if ON (currently works as expected only
6887     -- with 'l')
6888     if inmath and d == 'on' then
6889         d = ('TRT' == tex.mathdir) and 'r' or 'l'
6890     end
6891
6892     if d then
6893         if d == 'al' then
6894             d = 'r'
6895             last = 'al'
6896         elseif d == 'l' or d == 'r' then
6897             last = d
6898         end
6899         prev_d = d
6900         table.insert(nodes, {item, d, outer_first})
6901     end
6902
6903     outer_first = nil
6904
6905 end
6906
6907 -- TODO -- repeated here in case EN/ET is the last node. Find a
6908 -- better way of doing things:
6909 if first_et then      -- dir may be nil here !
6910     if has_en then
6911         if last == 'l' then
6912             temp = 'l'      -- W7
6913         else
6914             temp = 'en'     -- W5
6915         end
6916     else

```

```

6917     temp = 'on'      -- W6
6918   end
6919   for e = first_et, #nodes do
6920     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6921   end
6922 end
6923
6924 -- dummy node, to close things
6925 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6926
6927 ----- NEUTRAL -----
6928
6929 outer = save_outer
6930 last = outer
6931
6932 local first_on = nil
6933
6934 for q = 1, #nodes do
6935   local item
6936
6937   local outer_first = nodes[q][3]
6938   outer = outer_first or outer
6939   last = outer_first or last
6940
6941   local d = nodes[q][2]
6942   if d == 'an' or d == 'en' then d = 'r' end
6943   if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6944
6945   if d == 'on' then
6946     first_on = first_on or q
6947   elseif first_on then
6948     if last == d then
6949       temp = d
6950     else
6951       temp = outer
6952     end
6953     for r = first_on, q - 1 do
6954       nodes[r][2] = temp
6955       item = nodes[r][1] -- MIRRORING
6956       if Babel.mirroring_enabled and item.id == GLYPH
6957         and temp == 'r' and characters[item.char] then
6958         local font_mode = font.fonts[item.font].properties.mode
6959         if font_mode ~= 'harf' and font_mode ~= 'plug' then
6960           item.char = characters[item.char].m or item.char
6961         end
6962       end
6963     end
6964     first_on = nil
6965   end
6966
6967   if d == 'r' or d == 'l' then last = d end
6968 end
6969
6970 ----- IMPLICIT, REORDER -----
6971
6972 outer = save_outer
6973 last = outer
6974
6975 local state = {}

```

```

6976 state.has_r = false
6977
6978 for q = 1, #nodes do
6979     local item = nodes[q][1]
6980
6981     outer = nodes[q][3] or outer
6982
6983     local d = nodes[q][2]
6984
6985     if d == 'nsm' then d = last end          -- W1
6986     if d == 'en' then d = 'an' end
6987     local isdir = (d == 'r' or d == 'l')
6988
6989     if outer == 'l' and d == 'an' then
6990         state.san = state.san or item
6991         state.ean = item
6992     elseif state.san then
6993         head, state = insert_numeric(head, state)
6994     end
6995
6996     if outer == 'l' then
6997         if d == 'an' or d == 'r' then      -- im -> implicit
6998             if d == 'r' then state.has_r = true end
6999             state.sim = state.sim or item
7000             state.eim = item
7001         elseif d == 'l' and state.sim and state.has_r then
7002             head, state = insert_implicit(head, state, outer)
7003         elseif d == 'l' then
7004             state.sim, state.eim, state.has_r = nil, nil, false
7005         end
7006     else
7007         if d == 'an' or d == 'l' then
7008             if nodes[q][3] then -- nil except after an explicit dir
7009                 state.sim = item -- so we move sim 'inside' the group
7010             else
7011                 state.sim = state.sim or item
7012             end
7013             state.eim = item
7014         elseif d == 'r' and state.sim then
7015             head, state = insert_implicit(head, state, outer)
7016         elseif d == 'r' then
7017             state.sim, state.eim = nil, nil
7018         end
7019     end
7020 end
7021
7022 if isdir then
7023     last = d          -- Don't search back - best save now
7024 elseif d == 'on' and state.san then
7025     state.san = state.san or item
7026     state.ean = item
7027 end
7028
7029 end
7030
7031 return node.prev(head) or head
7032 end
7033 </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.

```
7034 <*nil>
7035 \ProvidesLanguage{nil}[<<date>> <<version>> Nil language]
7036 \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.

```
7037 \ifx\l@nil\undefined
7038   \newlanguage\l@nil
7039   \@namedef{bbl@hyphendata@the\l@nil}{}{}% Remove warning
7040   \let\bbl@elt\relax
7041   \edef\bbl@languages{% Add it to the list of languages
7042     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
7043 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
7044 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil
7045 \let\captionnil\@empty
7046 \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.

```
7047 \ldf@finish{nil}
7048 </nil>
```

16 Support for Plain \TeX (plain.def)

16.1 Not renaming `hyphen.tex`

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based \TeX -format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTeX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTeX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```
7049 <(*bplain | blplain)
7050 \catcode`\{=1 % left brace is begin-group character
7051 \catcode`\}=2 % right brace is end-group character
7052 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called `hyphen.cfg` can be found, we make sure that it will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
7053 \openin 0 hyphen.cfg
7054 \ifeof0
7055 \else
7056   \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that’s done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
7057   \def\input #1 {%
7058     \let\input\input
7059     \a hyphen.cfg
7060     \let\input\undefined
7061   }
7062 \fi
7063 </bplain | blplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
7064 <bplain>\a plain.tex
7065 <blplain>\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
7066 <bplain>\def\fmtname{babel-plain}
7067 <blplain>\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some \LaTeX features

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

```
7068 <(*Emulate LaTeX)> \equiv
7069 \def\@empty{}
7070 \def\loadlocalcfg#1{%
7071   \openin0#1.cfg
7072   \ifeof0
```

```

7073 \closein0
7074 \else
7075 \closein0
7076 {\immediate\write16{*****}%}
7077 \immediate\write16{* Local config file #1.cfg used}%
7078 \immediate\write16{*}%
7079 }
7080 \input #1.cfg\relax
7081 \fi
7082 \@endoflfd}

```

16.3 General tools

A number of \LaTeX macro's that are needed later on.

```

7083 \long\def\@firstofone#1{#1}
7084 \long\def\@firstoftwo#1#2{#1}
7085 \long\def\@secondoftwo#1#2{#2}
7086 \def\@nnil{\@nil}
7087 \def\@gobbletwo#1#2{}
7088 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7089 \def\@star@or@long#1{%
7090 \@ifstar
7091 {\let\l@ngrel@x\relax#1}%
7092 {\let\l@ngrel@x\long#1}}
7093 \let\l@ngrel@x\relax
7094 \def\@car#1#2\@nil{#1}
7095 \def\@cdr#1#2\@nil{#2}
7096 \let\@typeset@protect\relax
7097 \let\protected@edef\edef
7098 \long\def\@gobble#1{}
7099 \edef\@backslashchar{\expandafter\@gobble\string\}
7100 \def\strip@prefix#1>{}
7101 \def\g@addto@macro#1#2{%
7102 \toks@{\expandafter{#1#2}}%
7103 \xdef#1{\the\toks@}}
7104 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7105 \def\@nameuse#1{\csname #1\endcsname}
7106 \def\@ifundefined#1{%
7107 \expandafter\ifx\csname#1\endcsname\relax
7108 \expandafter\@firstoftwo
7109 \else
7110 \expandafter\@secondoftwo
7111 \fi}
7112 \def\@expandtwoargs#1#2#3{%
7113 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7114 \def\zap@space#1 #2{%
7115 #1%
7116 \ifx#2\@empty\else\expandafter\zap@space\fi
7117 #2}
7118 \let\bbl@trace\@gobble
7119 \def\bbl@error#1#2{%
7120 \begingroup
7121 \newlinechar=^^J
7122 \def\{^^J(babel) }%
7123 \errhelp{#2}\errmessage{\#1}%
7124 \endgroup}
7125 \def\bbl@warning#1{%
7126 \begingroup

```

```

7127 \newlinechar=`^^J
7128 \def\{^^J(babel) }%
7129 \message{\#1}%
7130 \endgroup}
7131 \let\bbl@infowarn\bbl@warning
7132 \def\bbl@info#1{%
7133 \begingroup
7134 \newlinechar=`^^J
7135 \def\{^^J}%
7136 \wlog{#1}%
7137 \endgroup}

```

$\LaTeX_2\epsilon$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

7138 \ifx\@preamblecmds\@undefined
7139 \def\@preamblecmds{}
7140 \fi
7141 \def\@onlypreamble#1{%
7142 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7143 \@preamblecmds\do#1}}
7144 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

7145 \def\begindocument{%
7146 \@begindocumenthook
7147 \global\let\@begindocumenthook\@undefined
7148 \def\do##1{\global\let##1\@undefined}%
7149 \@preamblecmds
7150 \global\let\do\noexpand}
7151 \ifx\@begindocumenthook\@undefined
7152 \def\@begindocumenthook{}
7153 \fi
7154 \@onlypreamble\@begindocumenthook
7155 \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`.

```

7156 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7157 \@onlypreamble\AtEndOfPackage
7158 \def\@endofldf{}
7159 \@onlypreamble\@endofldf
7160 \let\bbl@afterlang\@empty
7161 \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.

```

7162 \catcode`\&=\z@
7163 \ifx&\if@files\@undefined
7164 \expandafter\let\csname if@files\expandafter\endcsname
7165 \csname iffalse\endcsname
7166 \fi
7167 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7168 \def\newcommand{\@star@or@long\new@command}
7169 \def\new@command#1{%
7170 \@testopt{\@newcommand#1}0}
7171 \def\@newcommand#1[#2]{%

```

```

7172 \@ifnextchar [{\@argdef#1[#2]]%
7173         {\@argdef#1[#2]}}
7174 \long\def\@argdef#1[#2]#3{%
7175     \@yargdef#1\@ne{#2}{#3}}
7176 \long\def\@argdef#1[#2][#3]#4{%
7177     \expandafter\def\expandafter#1\expandafter{%
7178         \expandafter\@protected@testopt\expandafter #1%
7179         \csname\string#1\expandafter\endcsname{#3}}}%
7180 \expandafter\@yargdef \csname\string#1\endcsname
7181 \tw@{#2}{#4}}
7182 \long\def\@yargdef#1#2#3{%
7183     \@tempcnta#3\relax
7184     \advance \@tempcnta \@ne
7185     \let\@hash@\relax
7186     \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7187     \@tempcntb #2%
7188     \@whilenum\@tempcntb <\@tempcnta
7189     \do{%
7190         \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7191         \advance\@tempcntb \@ne}%
7192     \let\@hash@###
7193     \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
7194 \def\providecommand{\@star@or@long\provide@command}
7195 \def\provide@command#1{%
7196     \begingroup
7197     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7198     \endgroup
7199     \expandafter\ifundefined\@gtempa
7200     {\def\reserved@a{\new@command#1}}%
7201     {\let\reserved@a\relax
7202     \def\reserved@a{\new@command\reserved@a}}%
7203     \reserved@a}%
7204 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7205 \def\declare@robustcommand#1{%
7206     \edef\reserved@a{\string#1}%
7207     \def\reserved@b{#1}%
7208     \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7209     \edef#1{%
7210         \ifx\reserved@a\reserved@b
7211             \noexpand\x@protect
7212             \noexpand#1%
7213         \fi
7214         \noexpand\protect
7215         \expandafter\noexpand\csname
7216             \expandafter\@gobble\string#1 \endcsname
7217     }%
7218     \expandafter\new@command\csname
7219         \expandafter\@gobble\string#1 \endcsname
7220 }
7221 \def\x@protect#1{%
7222     \ifx\protect\@typeset@protect\else
7223         \@x@protect#1%
7224     \fi
7225 }
7226 \catcode`\&=\z@ % Trick to hide conditionals
7227 \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`.

```

7228 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7229 \catcode`\&=4
7230 \ifx\in@\undefined
7231 \def\in@#1#2{%
7232   \def\in@@##1##2##3\in@{%
7233     \ifx\in@@##2\in@false\else\in@true\fi}%
7234   \in@@#2#1\in@\in@@}
7235 \else
7236   \let\bbl@tempa\@empty
7237 \fi
7238 \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).

```

7239 \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.

```

7240 \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.

```

7241 \ifx\@tempcnta\@undefined
7242   \csname newcount\endcsname\@tempcnta\relax
7243 \fi
7244 \ifx\@tempcntb\@undefined
7245   \csname newcount\endcsname\@tempcntb\relax
7246 \fi

```

To prevent wasting two counters in \LaTeX (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

7247 \ifx\bye\@undefined
7248   \advance\count10 by -2\relax
7249 \fi
7250 \ifx\@ifnextchar\@undefined
7251   \def\@ifnextchar#1#2#3{%
7252     \let\reserved@d=#1%
7253     \def\reserved@a{#2}\def\reserved@b{#3}%
7254     \futurelet\@let@token\@ifnch}
7255   \def\@ifnch{%
7256     \ifx\@let@token\@sptoken
7257       \let\reserved@c\@xifnch
7258     \else
7259       \ifx\@let@token\reserved@d
7260         \let\reserved@c\reserved@a
7261       \else
7262         \let\reserved@c\reserved@b
7263       \fi
7264     \fi
7265     \reserved@c}
7266   \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
7267   \def\;\@xifnch\expandafter\def\:{\futurelet\@let@token\@ifnch}
7268 \fi
7269 \def\@testopt#1#2{%

```

```

7270 \@ifnextchar[{\#1}{\#1[\#2]}}
7271 \def\@protected@testopt#1{%
7272 \ifx\protect\@typeset@protect
7273 \expandafter\@testopt
7274 \else
7275 \@@x@protect#1%
7276 \fi}
7277 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7278 #2\relax}\fi}
7279 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7280 \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

7281 \def\DeclareTextCommand{%
7282 \@dec@text@cmd\providecommand
7283 }
7284 \def\ProvideTextCommand{%
7285 \@dec@text@cmd\providecommand
7286 }
7287 \def\DeclareTextSymbol#1#2#3{%
7288 \@dec@text@cmd\chardef#1{#2}#3\relax
7289 }
7290 \def\@dec@text@cmd#1#2#3{%
7291 \expandafter\def\expandafter#2%
7292 \expandafter{%
7293 \csname#3-cmd\expandafter\endcsname
7294 \expandafter#2%
7295 \csname#3\string#2\endcsname
7296 }%
7297 % \let\@ifdefinable\@rc@ifdefinable
7298 \expandafter#1\csname#3\string#2\endcsname
7299 }
7300 \def\@current@cmd#1{%
7301 \ifx\protect\@typeset@protect\else
7302 \noexpand#1\expandafter\@gobble
7303 \fi
7304 }
7305 \def\@changed@cmd#1#2{%
7306 \ifx\protect\@typeset@protect
7307 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7308 \expandafter\ifx\csname ?\string#1\endcsname\relax
7309 \expandafter\def\csname ?\string#1\endcsname{%
7310 \@changed@x@err{#1}%
7311 }%
7312 \fi
7313 \global\expandafter\let
7314 \csname\cf@encoding\string#1\expandafter\endcsname
7315 \csname ?\string#1\endcsname
7316 \fi
7317 \csname\cf@encoding\string#1%
7318 \expandafter\endcsname
7319 \else
7320 \noexpand#1%
7321 \fi
7322 }
7323 \def\@changed@x@err#1{%

```

```

7324 \errhelp{Your command will be ignored, type <return> to proceed}%
7325 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7326 \def\DeclareTextCommandDefault#1{%
7327 \DeclareTextCommand#1?%
7328 }
7329 \def\ProvideTextCommandDefault#1{%
7330 \ProvideTextCommand#1?%
7331 }
7332 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7333 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7334 \def\DeclareTextAccent#1#2#3{%
7335 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7336 }
7337 \def\DeclareTextCompositeCommand#1#2#3#4{%
7338 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7339 \edef\reserved@b{\string##1}%
7340 \edef\reserved@c{%
7341 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7342 \ifx\reserved@b\reserved@c
7343 \expandafter\expandafter\expandafter\ifx
7344 \expandafter\@car\reserved@a\relax\relax\@nil
7345 \@text@composite
7346 \else
7347 \edef\reserved@b##1{%
7348 \def\expandafter\noexpand
7349 \csname#2\string#1\endcsname####1{%
7350 \noexpand\@text@composite
7351 \expandafter\noexpand\csname#2\string#1\endcsname
7352 ####1\noexpand\@empty\noexpand\@text@composite
7353 {##1}%
7354 }%
7355 }%
7356 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7357 \fi
7358 \expandafter\def\csname\expandafter\string\csname
7359 #2\endcsname\string#1-\string#3\endcsname{#4}
7360 \else
7361 \errhelp{Your command will be ignored, type <return> to proceed}%
7362 \errmessage{\string\DeclareTextCompositeCommand\space used on
7363 inappropriate command \protect#1}
7364 \fi
7365 }
7366 \def\@text@composite#1#2#3\@text@composite{%
7367 \expandafter\@text@composite@x
7368 \csname\string#1-\string#2\endcsname
7369 }
7370 \def\@text@composite@x#1#2{%
7371 \ifx#1\relax
7372 #2%
7373 \else
7374 #1%
7375 \fi
7376 }
7377 %
7378 \def\@strip@args#1:#2-#3\@strip@args{#2}
7379 \def\DeclareTextComposite#1#2#3#4{%
7380 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7381 \bgroup
7382 \lcode`\@=#4%

```

```

7383 \lowercase{%
7384 \egroup
7385 \reserved@a @%
7386 }%
7387 }
7388 %
7389 \def\UseTextSymbol#1#2{#2}
7390 \def\UseTextAccent#1#2#3{}
7391 \def\@use@text@encoding#1{}
7392 \def\DeclareTextSymbolDefault#1#2{%
7393 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7394 }
7395 \def\DeclareTextAccentDefault#1#2{%
7396 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7397 }
7398 \def\cf@encoding{OT1}

```

Currently we only use the \TeX method for accents for those that are known to be made active in *some* language definition file.

```

7399 \DeclareTextAccent{"}{OT1}{127}
7400 \DeclareTextAccent{'}{OT1}{19}
7401 \DeclareTextAccent{^}{OT1}{94}
7402 \DeclareTextAccent{\`}{OT1}{18}
7403 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for `PLAIN \TeX` .

```

7404 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7405 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7406 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
7407 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7408 \DeclareTextSymbol{\i}{OT1}{16}
7409 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \TeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \TeX has, we just `\let` it to `\sevenrm`.

```

7410 \ifx\scriptsize\undefined
7411 \let\scriptsize\sevenrm
7412 \fi

```

And a few more “dummy” definitions.

```

7413 \def\language{english}%
7414 \let\bbl@opt@shorthands\@nnil
7415 \def\bbl@ifshorthand#1#2#3{#2}%
7416 \let\bbl@language@opts\@empty
7417 \ifx\babeloptionstrings\undefined
7418 \let\bbl@opt@strings\@nnil
7419 \else
7420 \let\bbl@opt@strings\babeloptionstrings
7421 \fi
7422 \def\BabelStringsDefault{generic}
7423 \def\bbl@tempa{normal}
7424 \ifx\babeloptionmath\bbl@tempa
7425 \def\bbl@mathnormal{\noexpand\textormath}
7426 \fi
7427 \def\AfterBabelLanguage#1#2{}
7428 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
7429 \let\bbl@afterlang\relax
7430 \def\bbl@opt@safe{BR}
7431 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi

```

```

7432 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
7433 \expandafter\newif\csname ifbbl@single\endcsname
7434 \chardef\bbl@bidimode\z@
7435 <</Emulate LaTeX>>

```

A proxy file:

```

7436 <*plain>
7437 \input babel.def
7438 </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 $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$ styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The $\mathcal{T}\mathcal{E}\mathcal{X}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *$\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: $\mathcal{T}\mathcal{E}\mathcal{X}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German $\mathcal{T}\mathcal{E}\mathcal{X}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$ is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).