

Babel

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Version 3.79
2022/08/29

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	7
1.4	Modifiers	8
1.5	Troubleshooting	8
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	10
1.10	Shorthands	12
1.11	Package options	15
1.12	The base option	17
1.13	ini files	17
1.14	Selecting fonts	25
1.15	Modifying a language	27
1.16	Creating a language	28
1.17	Digits and counters	31
1.18	Dates	33
1.19	Accessing language info	33
1.20	Hyphenation and line breaking	35
1.21	Transforms	37
1.22	Selection based on BCP 47 tags	39
1.23	Selecting scripts	40
1.24	Selecting directions	41
1.25	Language attributes	45
1.26	Hooks	45
1.27	Languages supported by babel with ldf files	47
1.28	Unicode character properties in luatex	48
1.29	Tweaking some features	48
1.30	Tips, workarounds, known issues and notes	48
1.31	Current and future work	50
1.32	Tentative and experimental code	50
2	Loading languages with language.dat	50
2.1	Format	51
3	The interface between the core of babel and the language definition files	51
3.1	Guidelines for contributed languages	52
3.2	Basic macros	53
3.3	Skeleton	54
3.4	Support for active characters	55
3.5	Support for saving macro definitions	55
3.6	Support for extending macros	56
3.7	Macros common to a number of languages	56
3.8	Encoding-dependent strings	56
3.9	Executing code based on the selector	59
II	Source code	60
4	Identification and loading of required files	60
5	locale directory	60

6	Tools	61
6.1	Multiple languages	65
6.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	66
6.3	<code>base</code>	67
6.4	<code>key=value</code> options and other general option	67
6.5	Conditional loading of shorthands	69
6.6	Interlude for Plain	70
7	Multiple languages	70
7.1	Selecting the language	73
7.2	Errors	81
7.3	Hooks	83
7.4	Setting up language files	85
7.5	Shorthands	87
7.6	Language attributes	96
7.7	Support for saving macro definitions	97
7.8	Short tags	99
7.9	Hyphens	99
7.10	Multiencoding strings	100
7.11	Macros common to a number of languages	107
7.12	Making glyphs available	107
7.12.1	Quotation marks	107
7.12.2	Letters	108
7.12.3	Shorthands for quotation marks	109
7.12.4	Umlauts and tremas	110
7.13	Layout	111
7.14	Load engine specific macros	112
7.15	Creating and modifying languages	112
8	Adjusting the Babel behavior	134
8.1	Cross referencing macros	135
8.2	Marks	138
8.3	Preventing clashes with other packages	139
8.3.1	<code>ifthen</code>	139
8.3.2	<code>varioref</code>	140
8.3.3	<code>hhline</code>	140
8.4	Encoding and fonts	141
8.5	Basic bidi support	142
8.6	Local Language Configuration	146
8.7	Language options	146
9	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	149
10	Loading hyphenation patterns	149
11	Font handling with <code>fontspec</code>	153
12	Hooks for XeTeX and LuaTeX	157
12.1	XeTeX	157
12.2	Layout	159
12.3	LuaTeX	160
12.4	Southeast Asian scripts	166
12.5	CJK line breaking	168
12.6	Arabic justification	170
12.7	Common stuff	173
12.8	Automatic fonts and ids switching	174
12.9	Bidi	178
12.10	Layout	180
12.11	Lua: transforms	185

12.12 Lua: Auto bidi with basic and basic-r	193
13 Data for CJK	203
14 The ‘nil’ language	203
15 Calendars	205
15.1 Islamic	205
16 Hebrew	206
17 Persian	210
18 Coptic	211
19 Buddhist	211
20 Support for Plain T_EX (plain.def)	212
20.1 Not renaming hyphen.tex	212
20.2 Emulating some L ^A T _E X features	212
20.3 General tools	213
20.4 Encoding related macros	216
21 Acknowledgements	219

Troubleshootoing

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	8
Unknown language ‘LANG’	8
Argument of \language@active@arg” has an extra }	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	26
Package babel Info: The following fonts are not babel standard families	26

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 `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

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

PDF \TeX

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

NOTE Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.22 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:


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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
! 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.

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

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

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

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.

NOTE Bear in mind `\selectlanguage` can be automatically executed, in some cases, in the auxiliary files, at heads and foots, and after the environment `otherlanguage*`.

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 There are a couple of issues related to the way the language information is written to the auxiliary files:

- `\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 `otherlanguage` instead.
- In addition, this macro inserts a `\write` in vertical mode, which may break the vertical spacing in some cases (for example, between lists). **New 3.64** The behavior can be adjusted with `\babeladjust{select.write=<mode>}`, where `<mode>` is `shift` (which shifts the skips down and adds a `\penalty`); `keep` (the default – with it the `\write` and the skips are kept in the order they are written), and `omit` (which may seem a too drastic solution, because nothing is written, but more often than not this command is applied to more or less short texts with no sectioning or similar commands and therefore no language synchronization is necessary).

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

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

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

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

Spaces after the environment are ignored.

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

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

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

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 `ruussian`, 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{ruussian}{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 of `\dag`). With `ini` files (see below), captions are ensured by default.

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

1.10 Shorthands

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

NOTE Keep in mind the following:

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

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).

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

The command `\useshortands` 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 `\useshortands*{\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 `\useshortands`. This restriction will be lifted in a future release.

`\defineshortand` [`\langle language \rangle`, `\langle language \rangle`, ...] `{\langle shorthand \rangle}{\langle code \rangle}`

The command `\defineshortand` 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 `\languageshortands{\langle lang \rangle}` to the corresponding `\extras\langle lang \rangle`, as explained below). By default, user shorthands are (re)defined.

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

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

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

```
\useshortands*{"}  
\defineshortand{"*}{\babelhyphen{soft}}  
\defineshortand{"-}{\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:

```
\defineshortand[*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.

`\languageshortands` `{\langle language \rangle}`

The command `\languageshortands` 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{\languageshortands{ngerman}}
```

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

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.

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

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

`\babelshorthand` $\langle shorthand \rangle$

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

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

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

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

⁶Thanks to Enrico Gregorio

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

character / over " in typing Polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`. For the reasons in the warning below, usage of this macro is not recommended.

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

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

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

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

1.11 Package options

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

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

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

activegrave Same for ```.

shorthands= `<char><char>... | off`

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

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

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

safe= `none | ref | bib`

Some \TeX macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from `varioref` and `ifthen`). With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in $\epsilon\TeX$ 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= *<file>*

Load *<file>*.cfg instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).

main= *<language>*

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 T_EX, 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 `\MakeUpper case` and the like (this feature misuses some internal L^AT_EX 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 `other language`;

other* also sets it at `other language*` 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

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.

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.

provide= *

New 3.49 An alternative to `\babelprovide` for languages passed as options. See section 1.13, which describes also the variants `provide+=` and `provide*=`.

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 $\langle\textit{option-name}\rangle\{\langle\textit{code}\rangle\}$

This command is currently the only provided by `base`. Executes $\langle\textit{code}\rangle$ 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 $\langle\textit{option-name}\rangle$ 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}
```

NOTE With a recent version of \LaTeX , an alternative method to execute some code just after an `ldf` file is loaded is with `\AddToHook` and the hook `file/<language>.ldf/after`. `Babel` does not predeclare it, and you have to do it yourself with `\ActivateGenericHook`.

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 250 of these files containing the basic data required for a locale, plus basic templates for 500 about locales.

`ini` files are not meant only for `babel`, and they have been devised as a resource for other packages. To easy interoperability between \TeX 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{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

Or also:

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

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

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

Arabic Monolingual documents mostly work in luatex, but it must be fine tuned, particularly math and 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).

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{lṇ lṃ lṁ lṅ lṇ lṅ} % 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 shorts texts the ini files should be fine, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX, etc.). This is what the class ltjbook does with luatex, which can be used in conjunction with the ldf for japanese, because the following piece of code loads luatexja:

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

Latin, Greek, Cyrillic Combining chars with the default luatex font renderer might be wrong; on the other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenations points are discarded (this bug is 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}	asa	Asu
agq	Aghem	ast	Asturian ^{ul}
ak	Akan	az-Cyrl	Azerbaijani
am	Amharic ^{ul}	az-Latn	Azerbaijani
ar	Arabic ^{ul}	az	Azerbaijani ^{ul}
ar-DZ	Arabic ^{ul}	bas	Basaa
ar-EG	Arabic ^{ul}	be	Belarusian ^{ul}
ar-IQ	Arabic ^{ul}	bem	Bemba
ar-JO	Arabic ^{ul}	bez	Bena
ar-LB	Arabic ^{ul}	bg	Bulgarian ^{ul}
ar-MA	Arabic ^{ul}	bm	Bambara
ar-PS	Arabic ^{ul}	bn	Bangla ^{ul}
ar-SA	Arabic ^{ul}	bo	Tibetan ^u
ar-SY	Arabic ^{ul}	brx	Bodo
ar-TN	Arabic ^{ul}	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian ^{ul}

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

mgo	Meta'	shi-Tfng	Tachelhit
mk	Macedonian ^{ul}	shi	Tachelhit
ml	Malayalam ^{ul}	si	Sinhala
mn	Mongolian	sk	Slovak ^{ul}
mr	Marathi ^{ul}	sl	Slovenian ^{ul}
ms-BN	Malay ^l	smn	Inari Sami
ms-SG	Malay ^l	sn	Shona
ms	Malay ^{ul}	so	Somali
mt	Maltese	sq	Albanian ^{ul}
mua	Mundang	sr-Cyrl-BA	Serbian ^{ul}
my	Burmese	sr-Cyrl-ME	Serbian ^{ul}
mzn	Mazanderani	sr-Cyrl-XK	Serbian ^{ul}
naq	Nama	sr-Cyrl	Serbian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Latn-BA	Serbian ^{ul}
nd	North Ndebele	sr-Latn-ME	Serbian ^{ul}
ne	Nepali	sr-Latn-XK	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Latn	Serbian ^{ul}
nmg	Kwasio	sr	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sv	Swedish ^{ul}
nnh	Ngiemboon	sw	Swahili
no	Norwegian	ta	Tamil ^u
nus	Nuer	te	Telugu ^{ul}
nyn	Nyankole	teo	Teso
om	Oromo	th	Thai ^{ul}
or	Odia	ti	Tigrinya
os	Ossetic	tk	Turkmen ^{ul}
pa-Arab	Punjabi	to	Tongan
pa-Guru	Punjabi	tr	Turkish ^{ul}
pa	Punjabi	twq	Tasawaq
pl	Polish ^{ul}	tzm	Central Atlas Tamazight
pms	Piedmontese ^{ul}	ug	Uyghur
ps	Pashto	uk	Ukrainian ^{ul}
pt-BR	Portuguese ^{ul}	ur	Urdu ^{ul}
pt-PT	Portuguese ^{ul}	uz-Arab	Uzbek
pt	Portuguese ^{ul}	uz-Cyrl	Uzbek
qu	Quechua	uz-Latn	Uzbek
rm	Romansh ^{ul}	uz	Uzbek
rn	Rundi	vai-Latn	Vai
ro	Romanian ^{ul}	vai-Vaii	Vai
ro-MD	Moldavian ^{ul}	vai	Vai
rof	Rombo	vi	Vietnamese ^{ul}
ru	Russian ^{ul}	vun	Vunjo
rw	Kinyarwanda	wae	Walser
rwk	Rwa	xog	Soga
sa-Beng	Sanskrit	yav	Yangben
sa-Deva	Sanskrit	yi	Yiddish
sa-Gujr	Sanskrit	yo	Yoruba
sa-Knda	Sanskrit	yue	Cantonese
sa-Mlym	Sanskrit	zgh	Standard Moroccan Tamazight
sa-Telu	Sanskrit	zh-Hans-HK	Chinese ^u
sa	Sanskrit	zh-Hans-MO	Chinese ^u
sah	Sakha	zh-Hans-SG	Chinese ^u
saq	Samburu	zh-Hans	Chinese ^u
sbp	Sangu	zh-Hant-HK	Chinese ^u
se	Northern Sami ^{ul}	zh-Hant-MO	Chinese ^u
seh	Sena	zh-Hant	Chinese ^u
ses	Koyraboro Senni	zh	Chinese ^u
sg	Sango	zu	Zulu
shi-Latn	Tachelhit		

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

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

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

filipino	kwasio
finnish	kyrgyz
french-be	lakota
french-belgium	langi
french-ca	lao
french-canada	latvian
french-ch	lingala
french-lu	lithuanian
french-luxembourg	lowersorbian
french-switzerland	lsorbian
french	lubakatanga
friulian	luo
fulah	luxembourgish
galician	luyia
ganda	macedonian
georgian	machame
german-at	makhuwameetto
german-austria	makonde
german-ch	malagasy
german-switzerland	malay-bn
german	malay-brunei
greek	malay-sg
gujarati	malay-singapore
gusii	malay
hausa-gh	malayalam
hausa-ghana	maltese
hausa-ne	manx
hausa-niger	marathi
hausa	masai
hawaiian	mazanderani
hebrew	meru
hindi	meta
hungarian	mexican
icelandic	mongolian
igbo	morisyen
inarisami	mundang
indonesian	nama
interlingua	nepali
irish	newzealand
italian	ngiemboon
japanese	ngomba
jolafonyi	norsk
kabuverdianu	northernluri
kabyle	northernsami
kako	northndebele
kalaallisut	norwegianbokmal
kalenjin	norwegiannynorsk
kamba	nswissgerman
kannada	nuer
kashmiri	nyankole
kazakh	nynorsk
khmer	occitan
kikuyu	oriya
kinyarwanda	oromo
konkani	ossetic
korean	pashto
koyraborosenni	persian
koyrachiini	piedmontese

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

Modifying and adding values to ini files

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

1.14 Selecting fonts

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

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}{עִבְרִית} svenska.

\end{document}
```

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

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

This is *not* an 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* an 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.

NOTE `\babelfont` is a high level interface to `fontspec`, and therefore in `xetex` you can apply Mappings. For example, there is a set of [transliterations for Brahmic scripts](#) by Davis M. Jones. After installing them in you distribution, just set the map as you would do with `fontspec`.

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

NOTE Do not redefine a caption in the following way:

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

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

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

```
\addto\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= *<language-tag>*

Loads only the strings. For example:

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

hyphenrules= *<language-list>*

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

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

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

A special value is +, which allocates a new language (in the 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 (xetex or luatex) is mainly in Polytonic Greek with 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, polytonicgreek, provide=*]{babel}
```

Remember there is an alternative syntax for the latter:

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

Finally, also remember you might not need to load `italian` at all if there are only a few word in this language (see 1.3).

script= *<script-name>*

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

language= *<language-name>*

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= *<counter-name>*

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

Alph= *<counter-name>*

Same for `\Alph`.

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

onchar= `ids` | `fonts`

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

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

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

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

intrapenalty= $\langle penalty \rangle$

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

transforms= $\langle transform-list \rangle$

See section 1.21.

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

Languages providing native digits in all or some variants are:

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

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

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

`\localnumeral` $\{\langle style \rangle\}\{\langle number \rangle\}$
`\localecounter` $\{\langle style \rangle\}\{\langle counter \rangle\}$

New 3.41 Many ‘ini’ locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with `xetex` and `luatex` and are fully expendable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000. There are several ways to use them (for the available styles in each language, see the list below):

- `\localnumeral{\langle style \rangle}\{\langle number \rangle\}`, like `\localnumeral{abjad}{15}`
- `\localecounter{\langle style \rangle}\{\langle counter \rangle\}`, 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`
Armenian `lower.letter`, `upper.letter`
Belarusian, Bulgarian, Church Slavic, Macedonian, Serbian `lower`, `upper`
Bangla `alphabetic`
Central Kurdish `alphabetic`
Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
Church Slavic (Glagolitic) `letters`
Coptic `epact`, `lower.letters`
French `date.day` (mainly for internal use).
Georgian `letters`
Greek `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with `keraia`)
Hebrew `letters` (neither `geresh` nor `gershayim yet`)
Hindi `alphabetic`
Italian `lower.legal`, `upper.legal`
Japanese `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`, `informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

Khmer consonant
Korean consonant, syllable, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, 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

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=..., convert>*]{*<year>*}{*<month>*}{*<day>*}

By default the calendar is the Gregorian, but an ini file may define strings for other calendars (currently ar, ar-*, he, fa, hi). In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew and calendar=coptic). However, with the option convert it's converted (using internally the following command).

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

\babelcalendar [*<date>*]{*<calendar>*}{*<year-macro>*}{*<month-macro>*}{*<day-macro>*}

New 3.76 Although calendars aren't the primary concern of babel, the package should be able to, at least, generate correctly the current date in the way users would expect in their own culture. Currently, \localedate can print dates in a few calendars (provided the ini locale file has been imported), but year, month and day had to be entered by hand, which is very inconvenient. With this macro, the current date is converted and stored in the three last arguments, which must be macros: allowed calendars are buddhist, coptic, hebrew, islamic-civil, islamic-umalqura, persian. The optional argument converts the given date, in the form '*<year>*-'*<month>*-'*<day>*'. Please, refer to the page on the news for 3.76 in the babel site for further details.

1.19 Accessing language info

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

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

\iflanguage {*<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 T_EX 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). This is the value to be used for the ‘real’ provided tag (babel may fill other fields if they are considered necessary).

`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. This is a required field for the fonts to be correctly set up, and therefore it should be always defined.

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

`region.tag.bcp47` is the BCP 47 tag of the region or territory. Defined only if the locale loaded actually contains it (eg, `es-MX` does, but `es` doesn’t), which is how locales behave in the CLDR. **New 3.75**

`variant.tag.bcp47` is the BCP 47 tag of the variant (in the BCP 47 sense, like 1901 for German). **New 3.75**

`extension.⟨s⟩.tag.bcp47` is the BCP 47 value of the extension whose singleton is `⟨s⟩` (currently the recognized singletons are `x`, `t` and `u`). The internal syntax can be somewhat complex, and this feature is still somewhat tentative. An example is `classicalatin` which sets `extension.x.tag.bcp47` to `classic`. **New 3.75**

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.

New 3.75 Sometimes, it comes in handy to be able to use `\localeinfo` in an expandable way even if something went wrong (for example, the locale currently active is undefined). For these cases, `localeinfo*` just returns an empty string instead of raising an error. Bear in mind that babel, following the CLDR, may leave the region unset, which means `\getlanguageproperty*`, described below, is the preferred command, so that the existence of a field can be checked before. This also means building a string with the language and the region with `\localeinfo*{language.tab.bcp47}` - `\localeinfo*{region.tab.bcp47}` is not usually a good idea (because of the hyphen).

\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 פרק.

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

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

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 (when it makes sense) as an attribute, too.

`\LocaleForEach {<code>}`

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.

`ensureinfo=off` **New 3.75** Previously, ini files are loaded only with `\babelprovide` and also when languages are selected if there is a `\babelfont` or they have not been explicitly declared. Now the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met (in previous versions you had to enable it with `\BabelEnsureInfo` in the preamble). Because of the way this feature works, problems are very unlikely, but there is switch as a package option to turn the new behavior off (`ensureinfo=off`).

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. With luatex there are also tools for non-standard hyphenation rules, explained in the next section.

`\babelhyphen *` {<type>}

`\babelhyphen *` {<text>}

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

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

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

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

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

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Multiple declarations work much like `\hyphenation` (last wins), but 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 Use `\babelhyphenation` instead of `\hyphenation` to set hyphenation exceptions in the preamble before any language is explicitly set with a selector. 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 `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`).

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.

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}
```

New 3.67 Transforms predefined in the ini locale files can be made attribute-dependent, too. When an attribute between parenthesis is inserted subsequent transforms will be assigned to it (up to the list end or another attribute). For example, and provided an attribute called `\withsigmafinal` has been declared:

```
transforms = transliteration.omega (\withsigmafinal) sigma.final
```

This applies `transliteration.omega` always, but `sigma.final` only when `\withsigmafinal` is set.

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.
Finnish	<code>prehyphen.nobreak</code>	Line breaks just after hyphens prepended to words are prevented, like in “pakastekaapit ja -arkut”.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	<code>transliteration.omega</code>	Although the provided combinations are not the full set, this transform follows the syntax of Omega: = for the circumflex, v for digamma, and so on. For better compatibility with Levy’s system, ~ (as ‘string’) is an alternative to =. ' is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.

¹⁵They are similar in concept, but not the same, as those in Unicode. The main inspiration for this feature is the Omega transformation processes.

Greek	<code>sigma.final</code>	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transform does it. To prevent the conversion (an abbreviation, for example), write "s.
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: <code>!?:;</code> .
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs</i> , <i>ddz</i> , <i>ggy</i> , <i>lly</i> , <i>nny</i> , <i>ssz</i> , <i>tty</i> and <i>zzs</i> as <i>cs-cs</i> , <i>dz-dz</i> , etc.
Indic scripts	<code>danda.nobreak</code>	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.
Latin	<code>digraphs.ligatures</code>	Replaces the groups <i>ae</i> , <i>AE</i> , <i>oe</i> , <i>OE</i> with <i>æ</i> , <i>Æ</i> , <i>œ</i> , <i>Œ</i> .
Latin	<code>letters.noj</code>	Replaces <i>j</i> , <i>J</i> with <i>i</i> , <i>I</i> .
Latin	<code>letters.uv</code>	Replaces <i>v</i> , <i>U</i> with <i>u</i> , <i>V</i> .
Sanskrit	<code>transliteration.iast</code>	The IAST system to romanize Devanagari. ¹⁶
Serbian	<code>transliteration.gajica</code>	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	<code>kashida.plain</code>	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.

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

New 3.37-3.39 With *luatex* it is possible to define non-standard hyphenation rules, like *f-f* → *ff-f*, repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. 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}{([fmrtp]) | {1}}
{
  { no = {1}, pre = {1}{1}- }, % Replace first char with disc
  remove,                    % Remove automatic disc (2nd node)
  {}                          % Keep last char, untouched
}
```

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

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

New 3.67 With the optional argument you can associate a user defined transform to an attribute, so that it’s active only when it’s set (currently its attribute value is ignored). With this mechanism transforms can be set or unset even in the middle of paragraphs, and applied to single words. To define, set and unset the attribute, the LaTeX kernel provides the macros `\newattribute`, `\setattribute` and `\unsetattribute`. The following example shows how to use it, provided an attribute named `\latinnoj` has been declared:


```
\babelprehyphenation[attribute=\latinnoj]{latin}{ J }{ string = I }
```

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 [*\options*]{*\locale-name*}{*\lua-pattern*}{*\replacement*}

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.

See the description above for the optional argument.

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: $\text{fr-Latn-FR} \rightarrow \text{fr-Latn} \rightarrow \text{fr-FR} \rightarrow \text{fr}$. Languages with the same resolved name are considered the same. Case is normalized before, so that $\text{fr-latn-fr} \rightarrow \text{fr-Latn-FR}$. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

`autoload.bcp47` with values on and off.

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

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

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

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

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

1.23 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.¹⁷ 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.¹⁸

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

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

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

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

1.24 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 are progresses in the latter, including `amsmath` and `mathtools` too, but for example gathered 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

¹⁷The 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.

¹⁸But still defined for backwards compatibility.

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection`..`\section`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it 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.¹⁹

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** .

¹⁹Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

Digits in pdf_{tex} must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, xetex). This command is provided to set `{\langle lr-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-name \rangle}`

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

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

New 3.17 Something like:

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

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

```

(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}\language\language}%

```

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.

New 3.64 This is not the only way to inject code at those points. The events listed below can be used as a hook name in `\AddToHook` in the form `babel/⟨language-name⟩/⟨event-name⟩` (with `*` it's applied to all languages), but there is a limitation, because the parameters passed with the babel mechanism are not allowed. The `\AddToHook` mechanism does *not* replace the current one in 'babel'. Its main advantage is you can reconfigure 'babel' even before loading it. See the example below.

`\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 `\usesshortands*` to add a hook for the event `afterextras`).

New 3.33 They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE The generic unlocalized \TeX hooks are predefined, so that you can write:

```
\AddToHook{babel/*/afterextras}{\frenchspacing}
```

which is executed always after the extras for the language being selected (and just before the non-localized hooks defined with `\AddBabelHook`).

In addition, locale-specific hooks in the form `babel/<language-name>/<event-name>` are *recognized* (executed just before the localized babel hooks), but they are *not predefined*. You have to do it yourself. For example, to set `\frenchspacing` only in bengali:

```
\ActivateGenericHook{babel/bengali/afterextras}
\AddToHook{babel/bengali/afterextras}{\frenchspacing}
```


`\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)²⁰
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

²⁰The two last name comes from the times when they had to be shortened to 8 characters

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

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

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

1.28 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{`\z}{mirror}{`?}
\babelcharproperty{`-}{direction}{1l} % 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\extrasrussian{\inputencoding{koi8-r}}
```

- For the hyphenation to work correctly, `lccodes` cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²¹ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is 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 \TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

²¹This explains why \LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

1.31 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.²²

But that is the easy part, because they don't require modifying the \LaTeX internals.

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

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ból", but "from (3)" is "(3)-ból", in Spanish an item labelled "3." 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 babel site.

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 babel site for further details.

2 Loading languages with `language.dat`

\TeX and most engines based on it (pdf \TeX , xetex, $\epsilon\text{-}\TeX$, 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).²³ 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).²⁴

²²See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to \TeX because their aim is just to display information and not fine typesetting.

²³This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

²⁴The 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`.

2.1 Format

In that file the person who maintains a T_EX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored²⁵. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.²⁶ 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 T_EX users, so the files have to be coded so that they can be read by both L^AT_EX and plain T_EX. 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.

²⁵This is because different operating systems sometimes use very different file-naming conventions.

²⁶This is not a new feature, but in former versions it didn't work correctly.

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\langle lang \rangle hyphenmins`, `\captions\langle lang \rangle`, `\date\langle lang \rangle`, `\extras\langle lang \rangle` and `\noextras\langle lang \rangle` (the last two may be left empty); where `\langle lang \rangle` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date\langle lang \rangle` but not `\captions\langle lang \rangle` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@⟨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\langle lang \rangle` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁷
- 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.

²⁷But 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>`.

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

\main@language To postpone the activation of the definitions needed for a language until the beginning of a

document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.

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

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

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

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

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

\substitutefontfamily (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct \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\monthinname{<name of first month>}
% More strings

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

\StartBabelCommands*{<dialect>}{date}
\SetString\monthinname{<name of first month>}
```



```
% More strings

\EndBabelCommands

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

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

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

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

3.4 Support for active characters

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

- `\initiate@active@char` The internal macro `\initiate@active@char` is used in language definition files to instruct \TeX to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.
- `\bbl@activate` 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
- `\bbl@remove@special` macro, representing the set of all characters that have a special category code.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

Language definition files may want to *redefine* macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this²⁸.

- `\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,

²⁸This mechanism was introduced by Bernd Raichle.

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

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

\allowhyphens Same as `\bbl@allowhyphens`, but does nothing if the encoding is `T1`. It is intended mainly for characters provided as real glyphs by this encoding but constructed with `\accent` in `OT1`. Note the previous command (`\bbl@allowhyphens`) has different applications (hyphens and discretionaries) than this one (composite chars). Note also prior to version 3.7, `\allowhyphens` had the behavior of `\bbl@allowhyphens`.

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

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

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

3.8 Encoding-dependent strings

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

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

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

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

\StartBabelCommands `{<language-list>}{<category>}[<selector>]`

The *<language-list>* specifies which languages the block is intended for. A block is taken into account only if the `\CurrentOption` is listed here. Alternatively, you can define `\BabelLanguages` to a comma-separated list of languages to be defined (if undefined,

`\StartBabelCommands` sets it to `\CurrentOption`). You may write `\CurrentOption` as the language, but this is discouraged – a explicit name (or names) is much better and clearer. A “selector” is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name `unicode` must be used for `xetex` and `luatex` (the key `strings` has also other two special values: `generic` and `encoded`). If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

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

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

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

The `<category>` is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.²⁹ It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiname{M\"a\"rz}
```

²⁹In future releases further categories may be added.

```

\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 `\date \langle language \rangle` exists).

\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.³⁰

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands `{\langle code \rangle}`

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

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

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

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

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

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

```

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

```

#1 is replaced by the roman numeral.

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

³⁰This replaces in 3.9g a short-lived `\UseStrings` which has been removed because it did not work.

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 `\@uc1clist` (eg, `\bb\BB\cc\CC`). The mandatory arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \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`I=\I\relax}
  {\lccode`i=\i\relax
   \lccode`I=\I\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 though the given uppercase codes, using the step, and assigns them the `\lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO` *{<uccode-from>}{<uccode-to>}{<step>}{<lccode>}* loops though the given uppercase codes, using the step, and assigns them the `\lccode`, which is fixed (MO stands for *many-to-one*).

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

```
\SetHyphenMap{\BabelLowerMM{"100}{\I1F}{2}{\I01}}
```

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.

3.9 Executing code based on the selector

`\IfBabelSelectorTF {<selectors>}{<true>}{<false>}`

New 3.67 Sometimes a different setup is desired depending on the selector used. Values allowed in `<selectors>` are `select`, `other`, `foreign`, `other*` (and also `foreign*` for the tentative starred version), and it can consist of a comma-separated list. For example:

```
\IfBabelSelectorTF{other, other*}{A}{B}
```

is true with these two environment selectors.
Its natural place of use is in hooks or in `\extras<language>`.

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>).

4 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 set options and load 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.

5 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, MMM for the month name) and anything outside is text. In addition, [] is a non breakable space and [.] is an abbreviation dot.

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

6 Tools

```
1 <<version=3.79>>
2 <<date=2022/08/29>>
```

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 L^AT_EX is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

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

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

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

\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the \else and \fi parts of an \if-statement³¹. 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, \<.> for \noexpand applied to a built macro name (which does not define the macro if undefined to \relax, because it is created locally), and \[...] for

³¹This code is based on code presented in TUGboat vol. 12, no2, June 1991 in "An expansion Power Lemma" by Sonja Maus.

one-level expansion (where . . is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```

29 \def\bbl@exp#1{%
30   \begingroup
31   \let\<\bbl@exp@en
32   \let\[\bbl@exp@ue
33   \edef\bbl@exp@aux{\endgroup#1}%
34   \bbl@exp@aux}
35 \def\bbl@exp@en#1>{\expandafter\<\noexpand\csname#1\endcsname}%
36 \def\bbl@exp@ue#1]{%
37   \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@ifsamestring` 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 it 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}> \equiv
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>

```

6.1 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}> \equiv
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}> \equiv
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.

6.2 The Package File (LaTeX, 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     \PackageNote{babel}{#1}%
240   \endgroup}
241 \def\bbl@info#1{%
242   \begingroup
243     \def\{\MessageBreak}%
244     \PackageInfo{babel}{#1}%
245   \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.

```
246 <<Basic macros>>
247 \@ifpackagewith{babel}{silent}
248   {\let\bbl@info\@gobble
249     \let\bbl@infowarn\@gobble
250     \let\bbl@warning\@gobble}
251   {}
252 %
253 \def\AfterBabelLanguage#1{%
254   \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.

```
255 \ifx\bbl@languages\@undefined\else
256   \begingroup
257     \catcode\^^I=12
258     \@ifpackagewith{babel}{showlanguages}{%
259       \begingroup
```

```

260     \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}}%
261     \wlog{<*languages>}%
262     \bbl@languages
263     \wlog{</languages>}%
264   \endgroup{}}
265 \endgroup
266 \def\bbl@elt#1#2#3#4{%
267   \ifnum#2=\z@
268     \gdef\bbl@nulllanguage{#1}%
269     \def\bbl@elt##1##2##3##4{%
270       \fi}%
271   \bbl@languages
272 \fi%

```

6.3 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`.

```

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

```

6.4 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.

```

293 \bbl@trace{key=value and another general options}
294 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
295 \def\bbl@tempb#1.#2{% Remove trailing dot
296   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
297 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
298   \ifx\@empty#2%
299     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
300   \else
301     \in@{,provide=}{, #1}%
302     \ifin@
303       \edef\bbl@tempc{%
304         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
305     \else
306       \in@{=}{#1}%
307     \ifin@

```

```

308     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
309     \else
310     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
311     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
312     \fi
313   \fi
314 \fi}
315 \let\bbl@tempc\@empty
316 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
317 \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.

```

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

```

338 \let\bbl@opt@shorthands\@nnil
339 \let\bbl@opt@config\@nnil
340 \let\bbl@opt@main\@nnil
341 \let\bbl@opt@headfoot\@nnil
342 \let\bbl@opt@layout\@nnil
343 \let\bbl@opt@provide\@nnil

```

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

```

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

```

355 \let\bbl@language@opts\@empty
356 \DeclareOption*{%

```

```

357 \bbl@xin@{\string=}{\CurrentOption}%
358 \ifin@
359 \expandafter\bbl@tempa\CurrentOption\bbl@tempa
360 \else
361 \bbl@add@list\bbl@language@opts{\CurrentOption}%
362 \fi}

```

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

```

363 \ProcessOptions*
364 \ifx\bbl@opt@provide\@nnil
365 \let\bbl@opt@provide\@empty %%% MOVE above
366 \else
367 \chardef\bbl@iniflag\@ne
368 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
369 \in@{,provide,}{, #1,}%
370 \ifin@
371 \def\bbl@opt@provide{#2}%
372 \bbl@replace\bbl@opt@provide{;}{,}%
373 \fi}
374 \fi
375 %

```

6.5 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=...

```

376 \bbl@trace{Conditional loading of shorthands}
377 \def\bbl@sh@string#1{%
378 \ifx#1\@empty\else
379 \ifx#1t\string~%
380 \else\ifx#1c\string,%
381 \else\string#1%
382 \fi\fi
383 \expandafter\bbl@sh@string
384 \fi}
385 \ifx\bbl@opt@shorthands\@nnil
386 \def\bbl@ifshorthand#1#2#3{#2}%
387 \else\ifx\bbl@opt@shorthands\@empty
388 \def\bbl@ifshorthand#1#2#3{#3}%
389 \else

```

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

```

390 \def\bbl@ifshorthand#1{%
391 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
392 \ifin@
393 \expandafter\@firstoftwo
394 \else
395 \expandafter\@secondoftwo
396 \fi}

```

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

```

397 \edef\bbl@opt@shorthands{%
398 \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

399 \bbl@ifshorthand{'}%
400 {\PassOptionsToPackage{activeacute}{babel}}{}
401 \bbl@ifshorthand{`}%
402 {\PassOptionsToPackage{activegrave}{babel}}{}
403 \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.

```
404 \ifx\bbbl@opt@headfoot\@nnil\else
405   \g@addto@macro\@resetactivechars{%
406     \set@typeset@protect
407     \expandafter\select@language@x\expandafter{\bbbl@opt@headfoot}%
408     \let\protect\noexpand}
409 \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.

```
410 \ifx\bbbl@opt@safe\@undefined
411   \def\bbbl@opt@safe{BR}
412   % \let\bbbl@opt@safe\empty % -- By September
413 \fi
```

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

```
414 \bbbl@trace{Defining IfBabelLayout}
415 \ifx\bbbl@opt@layout\@nnil
416   \newcommand\IfBabelLayout[3]{#3}%
417 \else
418   \newcommand\IfBabelLayout[1]{%
419     \@expandtwoargs\in@{.#1.}{.\bbbl@opt@layout.}%
420     \ifin@
421       \expandafter\@firstoftwo
422     \else
423       \expandafter\@secondoftwo
424     \fi}
425 \fi
426 \</package>
427 \<core>
```

6.6 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*.

```
428 \ifx\ldf@quit\@undefined\else
429 \endinput\fi % Same line!
430 \<Make sure ProvidesFile is defined>
431 \ProvidesFile{babel.def}[\<date>] [\<version>] Babel common definitions]
432 \ifx\AtBeginDocument\@undefined % TODO. change test.
433   \<Emulate LaTeX>
434 \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.

```
435 \</core>
436 \<package | core>
```

7 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.

```
437 \def\bbbl@version{\<version>}
438 \def\bbbl@date{\<date>}
439 \<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.

```

440 \def\adddialect#1#2{%
441   \global\chardef#1#2\relax
442   \bbl@usehooks{adddialect}{#1}{#2}}%
443   \begingroup
444     \count@#1\relax
445     \def\bbl@elt##1##2##3##4{%
446       \ifnum\count@=##2\relax
447         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
448         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
449           set to \expandafter\string\csname l@##1\endcsname\%
450           (\string\language\the\count@). Reported}%
451         \def\bbl@elt####1####2####3####4{%
452           \fi}%
453         \bbl@cs{languages}%
454       \endgroup

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s 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.

```

455 \def\bbl@fixname#1{%
456   \begingroup
457   \def\bbl@tempe{l@}%
458   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
459   \bbl@tempd
460     {\lowercase\expandafter{\bbl@tempd}%
461     {\uppercase\expandafter{\bbl@tempd}%
462     \@empty
463     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
464     {\uppercase\expandafter{\bbl@tempd}}}%
465     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
466     {\lowercase\expandafter{\bbl@tempd}}}%
467     \@empty
468     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
469   \bbl@tempd
470   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
471 \def\bbl@iflanguage#1{%
472   \@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`.

```

473 \def\bbl@bcpcase#1#2#3#4\@#5{%
474   \ifx\@empty#3%
475     \uppercase{\def#5{#1#2}}%
476   \else
477     \uppercase{\def#5{#1}}%
478     \lowercase{\edef#5{#5#2#3#4}}%
479   \fi}
480 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
481   \let\bbl@bcp\relax
482   \lowercase{\def\bbl@tempa{#1}}%
483   \ifx\@empty#2%
484     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
485   \else\ifx\@empty#3%
486     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
487     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%

```



```

488     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
489     }%
490     \ifx\bbl@bcp\relax
491       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
492     \fi
493   \else
494     \bbl@bcpcase#2\@empty\@empty\@empty\bbl@tempb
495     \bbl@bcpcase#3\@empty\@empty\@empty\bbl@tempc
496     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
497       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
498     }%
499     \ifx\bbl@bcp\relax
500       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
501       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
502     }%
503   \fi
504   \ifx\bbl@bcp\relax
505     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
506     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
507   }%
508 \fi
509 \ifx\bbl@bcp\relax
510   \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
511 \fi
512 \fi\fi}
513 \let\bbl@initoload\relax
514 \def\bbl@provide@locale{%
515   \ifx\babelprovide\undefined
516     \bbl@error{For a language to be defined on the fly 'base'\\%
517               is not enough, and the whole package must be\\%
518               loaded. Either delete the 'base' option or\\%
519               request the languages explicitly}%
520     {See the manual for further details.}%
521   \fi
522 % TODO. Option to search if loaded, with \LocaleForEach
523 \let\bbl@auxname\language % Still necessary. TODO
524 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
525 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
526 \ifbbl@bcp@allowed
527   \expandafter\ifx\csname date\language\endcsname\relax
528     \expandafter
529     \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@empty
530     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
531       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
532       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
533       \expandafter\ifx\csname date\language\endcsname\relax
534         \let\bbl@initoload\bbl@bcp
535         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
536         \let\bbl@initoload\relax
537       \fi
538       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
539     \fi
540   \fi
541 \fi
542 \expandafter\ifx\csname date\language\endcsname\relax
543   \IfFileExists{babel-\language.tex}%
544   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
545   {}%
546 \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.

```

547 \def\iflanguage#1{%
548   \bbl@iflanguage{#1}{%
549     \ifnum\csname l@#1\endcsname=\language
550       \expandafter\@firstoftwo
551     \else
552       \expandafter\@secondoftwo
553     \fi}}

```

7.1 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.

```

554 \let\bbl@select@type\z@
555 \edef\selectlanguage{%
556   \noexpand\protect
557   \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`.

```
558 \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.

```
559 \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.

```
560 \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`

```

561 \def\bbl@push@language{%
562   \ifx\languagename\undefined\else
563     \ifx\currentgrouplevel\undefined
564       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
565     \else
566       \ifnum\currentgrouplevel=\z@
567         \xdef\bbl@language@stack{\languagename+}%
568       \else
569         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
570       \fi
571     \fi
572   \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 `\languagename`. 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`.

```
573 \def\bbl@pop@lang#1+#2\@@{%
574   \edef\language{#1}%
575   \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).

```
576 \let\bbl@ifrestoring\@secondoftwo
577 \def\bbl@pop@language{%
578   \expandafter\bbl@pop@lang\bbl@language@stack\@@
579   \let\bbl@ifrestoring\@firstoftwo
580   \expandafter\bbl@set@language\expandafter{\language}%
581   \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).

```
582 \chardef\localeid\z@
583 \def\bbl@id@last{0} % No real need for a new counter
584 \def\bbl@id@assign{%
585   \bbl@ifunset\bbl@id@@\language}%
586   {\count@\bbl@id@last\relax
587   \advance\count@\@ne
588   \bbl@csarg\chardef{id@@\language}\count@
589   \edef\bbl@id@last{\the\count@}%
590   \ifcase\bbl@engine\or
591     \directlua{
592       Babel = Babel or {}
593       Babel.locale_props = Babel.locale_props or {}
594       Babel.locale_props[\bbl@id@last] = {}
595       Babel.locale_props[\bbl@id@last].name = '\language'
596     }%
597   \fi}%
598   {}%
599   \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of `\selectlanguage`.

```
600 \expandafter\def\csname selectlanguage \endcsname#1{%
601   \ifnum\bbl@hymapset=\@ccclv\let\bbl@hymapset\tw@\fi
602   \bbl@push@language
603   \aftergroup\bbl@pop@language
604   \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 `lualatex`, is to avoid the `\write` altogether when not needed).

```

605 \def\BabelContentsFiles{toc,lof,lot}
606 \def\bbl@set@language#1{% from selectlanguage, pop@
607 % The old buggy way. Preserved for compatibility.
608 \edef\language{%
609 \ifnum\escapechar=\expandafter`\string#1\@empty
610 \else\string#1\@empty\fi}%
611 \ifcat\relax\noexpand#1%
612 \expandafter\ifx\csname date\language\endcsname\relax
613 \edef\language{#1}%
614 \let\locale\language
615 \else
616 \bbl@info{Using '\string\language' instead of 'language' is\\%
617 deprecated. If what you want is to use a\\%
618 macro containing the actual locale, make\\%
619 sure it does not not match any language.\\%
620 Reported}%
621 \ifx\scantokens\@undefined
622 \def\locale{??}%
623 \else
624 \scantokens\expandafter{\expandafter
625 \def\expandafter\locale\expandafter{\language}}%
626 \fi
627 \fi
628 \else
629 \def\locale{#1}% This one has the correct catcodes
630 \fi
631 \select@language{\language}%
632 % write to aux
633 \expandafter\ifx\csname date\language\endcsname\relax\else
634 \if@files
635 \ifx\babel@aux\@gobbles\else % Set if single in the first, redundant
636 \bbl@savelastskip
637 \protected@write\@auxout{}\{\string\babel@aux{\bbl@auxname}\}%
638 \bbl@restorelastskip
639 \fi
640 \bbl@usehooks{write}\}%
641 \fi
642 \fi}
643 %
644 \let\bbl@restorelastskip\relax
645 \let\bbl@savelastskip\relax
646 %
647 \newif\ifbbl@bcpallowed
648 \bbl@bcpallowedfalse
649 \def\select@language#1{% from set@, babel@aux
650 \ifx\bbl@selectorname\@empty
651 \def\bbl@selectorname{select}%
652 % set hymap
653 \fi
654 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
655 % set name
656 \edef\language{#1}%
657 \bbl@fixname\language
658 % TODO. name@map must be here?
659 \bbl@provide@locale
660 \bbl@iflanguage\language{%
661 \expandafter\ifx\csname date\language\endcsname\relax
662 \bbl@error
663 {Unknown language '\language'. Either you have\\%
664 misspelled its name, it has not been installed,\\%
665 or you requested it in a previous run. Fix its name,\\%
666 install it or just rerun the file, respectively. In\\%
667 some cases, you may need to remove the aux file}%

```

```

668         {You may proceed, but expect wrong results}%
669     \else
670         % set type
671         \let\bbl@select@type\z@
672         \expandafter\bbl@switch\expandafter{\language}%
673     \fi}}
674 \def\babel@aux#1#2{%
675     \select@language{#1}%
676     \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
677         \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
678 \def\babel@toc#1#2{%
679     \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.

```

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

```

```

718 % > babel-bidi
719 % > babel-fontspec
720 % hyphenation - case mapping
721 \ifcase\bbbl@opt@hyphenmap\or
722   \def\BabelLower##1##2{\lccode##1=##2\relax}%
723   \ifnum\bbbl@hymapsel>4\else
724     \csname\language\name @bbbl@hyphenmap\endcsname
725   \fi
726   \chardef\bbbl@opt@hyphenmap\z@
727 \else
728   \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
729     \csname\language\name @bbbl@hyphenmap\endcsname
730   \fi
731 \fi
732 \let\bbbl@hymapsel\@cclv
733 % hyphenation - select rules
734 \ifnum\csname l@language\endcsname=\l@unhyphenated
735   \edef\bbbl@tempa{u}%
736 \else
737   \edef\bbbl@tempa{\bbbl@cl{lbrk}}%
738 \fi
739 % linebreaking - handle u, e, k (v in the future)
740 \bbbl@xin@{/u}{/\bbbl@tempa}%
741 \ifin@{\else\bbbl@xin@{/e}{/\bbbl@tempa}}\fi % elongated forms
742 \ifin@{\else\bbbl@xin@{/k}{/\bbbl@tempa}}\fi % only kashida
743 \ifin@{\else\bbbl@xin@{/v}{/\bbbl@tempa}}\fi % variable font
744 \ifin@
745   % unhyphenated/kashida/elongated = allow stretching
746   \language\l@unhyphenated
747   \babel@savevariable\emergencystretch
748   \emergencystretch\maxdimen
749   \babel@savevariable\hbadness
750   \hbadness\@M
751 \else
752   % other = select patterns
753   \bbbl@patterns{#1}%
754 \fi
755 % hyphenation - mins
756 \babel@savevariable\lefthyphenmin
757 \babel@savevariable\righthyphenmin
758 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
759   \set@hyphenmins\tw@\thr@@\relax
760 \else
761   \expandafter\expandafter\expandafter\set@hyphenmins
762     \csname #1hyphenmins\endcsname\relax
763 \fi
764 \let\bbbl@selectorname\@empty}

```

`otherlanguage (env.)` 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.

```

765 \long\def\otherlanguage#1{%
766   \def\bbbl@selectorname{other}%
767   \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@@\fi
768   \csname selectlanguage \endcsname{#1}%
769   \ignorespaces}

```

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

```

770 \long\def\endotherlanguage{%

```

```
771 \global\@ignoretrue\ignorespaces}
```

`otherlanguage*` (*env.*) 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`.

```
772 \expandafter\def\csname otherlanguage*\endcsname{%
773   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
774 \def\bbl@otherlanguage@s[#1]#2{%
775   \def\bbl@selectorname{other*}%
776   \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
777   \def\bbl@select@opts{#1}%
778   \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”.

```
779 \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`.

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

```

The following macro executes conditionally some code based on the selector being used.

```

832 \def\IfBabelSelectorTF#1{%
833   \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
834   \ifin@
835     \expandafter\@firstoftwo
836   \else
837     \expandafter\@secondoftwo
838   \fi}

```

`\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` 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.

```

839 \let\bbl@hyphlist\@empty
840 \let\bbl@hyphenation@\relax
841 \let\bbl@pttnlist\@empty
842 \let\bbl@patterns@\relax
843 \let\bbl@hymapsel=\cclv
844 \def\bbl@patterns#1{%
845   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
846     \csname l@#1\endcsname
847     \edef\bbl@tempa{#1}%
848   \else
849     \csname l@#1:\f@encoding\endcsname
850     \edef\bbl@tempa{#1:\f@encoding}%
851   \fi
852   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%

```



```

853 % > luatex
854 \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
855 \begingroup
856 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
857 \ifin@else
858 \expandafter\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
859 \hyphenation{%
860 \bbl@hyphenation@
861 \@ifundefined{bbl@hyphenation@#1}%
862 \@empty
863 {\space\csname bbl@hyphenation@#1\endcsname}}%
864 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
865 \fi
866 \endgroup}}

```

`hyphenrules` (*env.*) 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, `\lcode`'s and font encodings are not set at all, so in most cases you should use `otherlanguage*`.

```

867 \def\hyphenrules#1{%
868 \edef\bbl@tempf{#1}%
869 \bbl@fixname\bbl@tempf
870 \bbl@iflanguage\bbl@tempf{%
871 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
872 \ifx\languageshorthands\undefined\else
873 \languageshorthands{none}%
874 \fi
875 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
876 \set@hyphenmins\tw@\thr@@\relax
877 \else
878 \expandafter\expandafter\expandafter\set@hyphenmins
879 \csname\bbl@tempf hyphenmins\endcsname\relax
880 \fi}}
881 \let\endhyphenrules\@empty

```

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

```

882 \def\providehyphenmins#1#2{%
883 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
884 \@namedef{#1hyphenmins}{#2}%
885 \fi}

```

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

```

886 \def\set@hyphenmins#1#2{%
887 \lefthyphenmin#1\relax
888 \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.

```

889 \ifx\ProvidesFile\undefined
890 \def\ProvidesLanguage#1[#2 #3 #4]{%
891 \wlog{Language: #1 #4 #3 <#2>}%
892 }
893 \else
894 \def\ProvidesLanguage#1{%
895 \begingroup
896 \catcode`\ 10 %
897 \@makeother\/%

```

```

898     \ifnextchar[%]
899     {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
900 \def\@provideslanguage#1[#2]{%
901   \wlog{Language: #1 #2}%
902   \expandafter\edef\csname ver@#1.ldf\endcsname{#2}%
903   \endgroup}
904 \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`.

```
905 \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.

```
906 \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’:

```

907 \providecommand\setlocale{%
908   \bbl@error
909   {Not yet available}%
910   {Find an armchair, sit down and wait}}
911 \let\uselocale\setlocale
912 \let\locale\setlocale
913 \let\selectlocale\setlocale
914 \let\textlocale\setlocale
915 \let\textlanguage\setlocale
916 \let\languagegettext\setlocale

```

7.2 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.

```

917 \edef\bbl@nulllanguage{\string\language=0}
918 \def\bbl@nocaption{\protect\bbl@nocaption@i}
919 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
920   \global\@namedef{#2}{\textbf{?#1?}}}%
921   \@nameuse{#2}%
922   \edef\bbl@tempa{#1}%
923   \bbl@sreplace\bbl@tempa{name}{}}%
924   \bbl@warning{% TODO.
925     \@backslashchar#1 not set for '\language'. Please,\\%
926     define it after the language has been loaded\\%
927     (typically in the preamble) with:\\%
928     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
929     Feel free to contribute on github.com/latex3/babel.\\%
930     Reported}}
931 \def\bbl@tentative{\protect\bbl@tentative@i}
932 \def\bbl@tentative@i#1{%
933   \bbl@warning{%
934     Some functions for '#1' are tentative.\\%
935     They might not work as expected and their behavior\\%
936     could change in the future.\\%
937     Reported}}
938 \def\@nolanerr#1{%

```

```

939 \bbl@error
940 {You haven't defined the language '#1' yet.\\%
941   Perhaps you misspelled it or your installation\\%
942   is not complete}%
943 {Your command will be ignored, type <return> to proceed}}
944 \def\bbl@nopatterns#1{%
945   \bbl@warning
946   {No hyphenation patterns were preloaded for\\%
947     the language '#1' into the format.\\%
948     Please, configure your TeX system to add them and\\%
949     rebuild the format. Now I will use the patterns\\%
950     preloaded for \bbl@nulllanguage\space instead}}
951 \let\bbl@usehooks\@gobbletwo
952 \ifx\bbl@onlyswitch\@empty\endinput\fi
953 % Here ended switch.def

```

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

```

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

```

`\addto` It takes two arguments, a *<control sequence>* and TeX-code to be added to the *<control sequence>*. If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

982 \def\addto#1#2{%
983   \ifx#1\@undefined
984     \def#1{#2}%
985   \else
986     \ifx#1\relax
987       \def#1{#2}%
988     \else
989       {\toks@\expandafter{#1#2}%
990        \xdef#1{\the\toks@}}%
991     \fi
992   \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?

```
993 \def\bbl@withactive#1#2{%
994   \begingroup
995     \lcode`~=`#2\relax
996     \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`.

```
997 \def\bbl@redefine#1{%
998   \edef\bbl@tempa{\bbl@stripslash#1}%
999   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1000   \expandafter\def\csname\bbl@tempa\endcsname{
1001     \@onlypreamble\bbl@redefine
```

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

```
1002 \def\bbl@redefine@long#1{%
1003   \edef\bbl@tempa{\bbl@stripslash#1}%
1004   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1005   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1006     \@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_`.

```
1007 \def\bbl@redefineroobust#1{%
1008   \edef\bbl@tempa{\bbl@stripslash#1}%
1009   \bbl@ifunset{\bbl@tempa\space}%
1010     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1011       \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1012     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1013     \@namedef{\bbl@tempa\space}%
1014     \@onlypreamble\bbl@redefineroobust
```

7.3 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.

```
1015 \bbl@trace{Hooks}
1016 \newcommand\AddBabelHook[3][]{%
1017   \bbl@ifunset{\bbl@hk##2}{\EnableBabelHook{##2}}}%
1018   \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1019   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1020   \bbl@ifunset{\bbl@ev##2@##3@##1}%
1021     {\bbl@csarg\bbl@add{ev##3@##1}{\bbl@elth{##2}}}%
1022     {\bbl@csarg\let{ev##2@##3@##1}\relax}%
1023   \bbl@csarg\newcommand{ev##2@##3@##1}{\bbl@tempb}}
1024 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk##1}\@firstofone}
1025 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk##1}\@gobble}
1026 \def\bbl@usehooks#1#2{%
1027   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1028   \def\bbl@elth##1{%
1029     \bbl@cs{hk##1}{\bbl@cs{ev##1@##1@##2}}%
1030     \bbl@cs{ev##1@##1}%
1031     \ifx\language\@undefined\else % Test required for Plain (?)
1032       \ifx\UseHook\@undefined\else\UseHook{babel/\language/#1}\fi
1033       \def\bbl@elth##1{%
1034         \bbl@cs{hk##1}{\bbl@c1{ev##1@##1@##2}}%
```

```

1035 \bbl@c1{ev@#1}%
1036 \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).

```

1037 \def\bbl@evargs{,% <- don't delete this comma
1038 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1039 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1040 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1041 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1042 beforestart=0,languagename=2}
1043 \ifx\NewHook\@undefined\else
1044 \def\bbl@tempa#1=#2\@{ \NewHook{babel/#1}}
1045 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1046 \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.

```

1047 \bbl@trace{Defining babelensure}
1048 \newcommand\babelensure[2][{}% TODO - revise test files
1049 \AddBabelHook{babel-ensure}{afterextras}{%
1050 \ifcase\bbl@select@type
1051 \bbl@c1{e}%
1052 \fi}%
1053 \begingroup
1054 \let\bbl@ens@include\@empty
1055 \let\bbl@ens@exclude\@empty
1056 \def\bbl@ens@fontenc{\relax}%
1057 \def\bbl@tempb##1{%
1058 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1059 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1060 \def\bbl@tempb##1=#2\@{ \@namedef{\bbl@ens@##1}{##2}}%
1061 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1062 \def\bbl@tempc{\bbl@ensure}%
1063 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1064 \expandafter{\bbl@ens@include}}%
1065 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1066 \expandafter{\bbl@ens@exclude}}%
1067 \toks@\expandafter{\bbl@tempc}%
1068 \bbl@exp{%
1069 \endgroup
1070 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1071 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1072 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist
1073 \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1074 \edef##1{\noexpand\bbl@nocaption
1075 {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1076 \fi
1077 \ifx##1\@empty\else
1078 \in@{##1}{#2}%
1079 \ifin@\else
1080 \bbl@ifunset{\bbl@ensure@\languagename}%
1081 {\bbl@exp{%
1082 \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1083 \\\foreignlanguage{\languagename}%

```

```

1084         {\ifx\relax#3\else
1085           \\fontencoding{#3}\\selectfont
1086           \fi
1087           #####1}}}%
1088     }%
1089     \toks@{\expandafter{##1}%
1090     \edef##1{%
1091       \bbl@csarg\noexpand{ensure@\language}\expandafter{##1}%
1092       {\the\toks@}}%
1093     \fi
1094     \expandafter\bbl@tempb
1095     \fi}%
1096 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1097 \def\bbl@tempa##1{% elt for include list
1098   \ifx##1\@empty\else
1099     \bbl@csarg\in@{ensure@\language}\expandafter}\expandafter{##1}%
1100     \ifin@\else
1101       \bbl@tempb##1\@empty
1102     \fi
1103     \expandafter\bbl@tempa
1104     \fi}%
1105 \bbl@tempa#1\@empty}
1106 \def\bbl@captionslist{%
1107   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1108   \contentsname\listfigurename\listtablename\indexname\figurename
1109   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1110   \alsoname\proofname\glossaryname}

```

7.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1111 \bbl@trace{Macros for setting language files up}
1112 \def\bbl@ldfinit{%
1113   \let\bbl@screset\@empty
1114   \let\BabelStrings\bbl@opt@string
1115   \let\BabelOptions\@empty
1116   \let\BabelLanguages\relax
1117   \ifx\originalTeX\@undefined
1118     \let\originalTeX\@empty
1119   \else
1120     \originalTeX
1121   \fi}
1122 \def\LdfInit#1#2{%
1123   \chardef\atcatcode=\catcode`\@
1124   \catcode`\@=11\relax
1125   \chardef\eqcatcode=\catcode`\=

```

```

1126 \catcode`\==12\relax
1127 \expandafter\if\expandafter\@backslashchar
1128         \expandafter\@car\string#2\@nil
1129     \ifx#2\@undefined\else
1130         \ldf@quit{#1}%
1131     \fi
1132 \else
1133     \expandafter\ifx\csname#2\endcsname\relax\else
1134         \ldf@quit{#1}%
1135     \fi
1136 \fi
1137 \bbl@ldfinit}

```

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

```

1138 \def\ldf@quit#1{%
1139     \expandafter\main@language\expandafter{#1}%
1140     \catcode`\@=\atcatcode \let\atcatcode\relax
1141     \catcode`\==\eqcatcode \let\eqcatcode\relax
1142     \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.

```

1143 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1144     \bbl@afterlang
1145     \let\bbl@afterlang\relax
1146     \let\BabelModifiers\relax
1147     \let\bbl@screset\relax}%
1148 \def\ldf@finish#1{%
1149     \loadlocalcfg{#1}%
1150     \bbl@afterldf{#1}%
1151     \expandafter\main@language\expandafter{#1}%
1152     \catcode`\@=\atcatcode \let\atcatcode\relax
1153     \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1154 \@onlypreamble\LdfInit
1155 \@onlypreamble\ldf@quit
1156 \@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.

```

1157 \def\main@language#1{%
1158     \def\bbl@main@language{#1}%
1159     \let\language\bbl@main@language % TODO. Set locale name
1160     \bbl@id@assign
1161     \bbl@patterns{\language}

```

We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the `\AtBeginDocument` is executed. Languages do not set `\pagedir`, so we set here for the whole document to the main `\bodydir`.

```

1162 \def\bbl@beforestart{%
1163     \def\@nolanerr##1{%
1164         \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1165     \bbl@usehooks{beforestart}{}%
1166     \global\let\bbl@beforestart\relax}
1167 \AtBeginDocument{%
1168     {\@nameuse{bbl@beforestart}}% Group!
1169     \if@filesw

```

```

1170 \providecommand\babel@aux[2]{}%
1171 \immediate\write\@mainaux{%
1172   \string\providecommand\string\babel@aux[2]{}%
1173   \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1174 \fi
1175 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1176 \ifbbl@single % must go after the line above.
1177   \renewcommand\selectlanguage[1]{}%
1178   \renewcommand\foreignlanguage[2]{#2}%
1179   \global\let\babel@aux\@gobbletwo % Also as flag
1180 \fi
1181 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1182 \def\select@language@x#1{%
1183   \ifcase\bbl@select@type
1184     \bbl@ifsamestring\language#1\fi\select@language{#1}%
1185   \else
1186     \select@language{#1}%
1187   \fi}

```

7.5 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.

```

1188 \bbl@trace{Shorhands}
1189 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1190   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1191   \bbl@ifunset{@sanitize}\fi\bbl@add\@sanitize{\@makeother#1}%
1192   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1193     \begingroup
1194       \catcode`#1\active
1195       \nfss@catcodes
1196       \ifnum\catcode`#1=\active
1197         \endgroup
1198         \bbl@add\nfss@catcodes{\@makeother#1}%
1199       \else
1200         \endgroup
1201       \fi
1202   \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.

```

1203 \def\bbl@remove@special#1{%
1204   \begingroup
1205     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1206       \else\noexpand##1\noexpand##2\fi}%
1207     \def\do{\x\do}%
1208     \def\@makeother{\x\@makeother}%
1209   \edef\x{\endgroup
1210     \def\noexpand\dospecials{\dospecials}%
1211     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1212       \def\noexpand\@sanitize{\@sanitize}%
1213     \fi}%
1214   \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).

```

1215 \def\bbl@active@def#1#2#3#4{%
1216   \@namedef{#3#1}{%
1217     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1218       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1219     \else
1220       \bbl@afterfi\csname#2@sh@#1\endcsname
1221     \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.

```

1222 \long\@namedef{#3@arg#1}##1{%
1223   \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1224     \bbl@afterelse\csname#4#1\endcsname##1%
1225   \else
1226     \bbl@afterfi\csname#2@sh@#1@string##1\endcsname
1227   \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.

```

1228 \def\initiate@active@char#1{%
1229   \bbl@ifunset{active@char\string#1}%
1230   {\bbl@withactive
1231     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1232   {}}

```

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).

```

1233 \def\@initiate@active@char#1#2#3{%
1234   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1235   \ifx#1\undefined
1236     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1237   \else
1238     \bbl@csarg\let{oridef@#2}#1%
1239     \bbl@csarg\edef{oridef@#2}{%
1240       \let\noexpand#1%
1241       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1242   \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*").

```

1243   \ifx#1#3\relax
1244     \expandafter\let\csname normal@char#2\endcsname#3%

```

```

1245 \else
1246 \bbl@info{Making #2 an active character}%
1247 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1248 \namedef{normal@char#2}{%
1249 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1250 \else
1251 \namedef{normal@char#2}{#3}%
1252 \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).

```

1253 \bbl@restoreactive{#2}%
1254 \AtBeginDocument{%
1255 \catcode`#2\active
1256 \if@files
1257 \immediate\write\@mainaux{\catcode`\string#2\active}%
1258 \fi}%
1259 \expandafter\bbl@add@special\csname#2\endcsname
1260 \catcode`#2\active
1261 \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}).

```

1262 \let\bbl@tempa\@firstoftwo
1263 \if\string^#2%
1264 \def\bbl@tempa{\noexpand\textormath}%
1265 \else
1266 \ifx\bbl@mathnormal\@undefined\else
1267 \let\bbl@tempa\bbl@mathnormal
1268 \fi
1269 \fi
1270 \expandafter\edef\csname active@char#2\endcsname{%
1271 \bbl@tempa
1272 {\noexpand\if@safe@actives
1273 \noexpand\expandafter
1274 \expandafter\noexpand\csname normal@char#2\endcsname
1275 \noexpand\else
1276 \noexpand\expandafter
1277 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1278 \noexpand\fi}%
1279 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1280 \bbl@csarg\edef{doactive#2}{%
1281 \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!).

```

1282 \bbl@csarg\edef{active@#2}{%
1283 \noexpand\active@prefix\noexpand#1%
1284 \expandafter\noexpand\csname active@char#2\endcsname}%
1285 \bbl@csarg\edef{normal@#2}{%
1286 \noexpand\active@prefix\noexpand#1%
1287 \expandafter\noexpand\csname normal@char#2\endcsname}%
1288 \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.

```
1289 \bbl@active@def#2\user@group{user@active}{language@active}%
1290 \bbl@active@def#2\language@group{language@active}{system@active}%
1291 \bbl@active@def#2\system@group{system@active}{normal@char}%
```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as ' ' ends up in a heading T_EX would see \protect '\protect '. To prevent this from happening a couple of shorthand needs to be defined at user level.

```
1292 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1293 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1294 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1295 {\expandafter\noexpand\csname user@active#2\endcsname}%
```

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

```
1296 \if\string'#2%
1297 \let\prim@s\bbl@prim@s
1298 \let\active@math@prime#1%
1299 \fi
1300 \bbl@usehooks{initiateactive}{#{1}{#2}{#3}}}
```

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

```
1301 <<{*More package options}>> ≡
1302 \DeclareOption{math=active}{%
1303 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1304 <</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.

```
1305 \ifpackagewith{babel}{KeepShorthandsActive}%
1306 {\let\bbl@restoreactive\@gobble}%
1307 {\def\bbl@restoreactive#1{%
1308 \bbl@exp{%
1309 \\\AfterBabelLanguage\\CurrentOption
1310 {\catcode`#1=\the\catcode`#1\relax}%
1311 \\\AtEndOfPackage
1312 {\catcode`#1=\the\catcode`#1\relax}}}%
1313 \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.

```
1314 \def\bbl@sh@select#1#2{%
1315 \expandafter\ifx\csname#1sh#2@sel\endcsname\relax
1316 \bbl@afterelse\bbl@scndcs
1317 \else
1318 \bbl@afterfi\csname#1sh#2@sel\endcsname
1319 \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.

```

1320 \begingroup
1321 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1322 {\gdef\active@prefix#1{%
1323   \ifx\protect\@typeset@protect
1324   \else
1325     \ifx\protect\@unexpandable@protect
1326       \noexpand#1%
1327     \else
1328       \protect#1%
1329     \fi
1330   \expandafter\@gobble
1331   \fi}}
1332 {\gdef\active@prefix#1{%
1333   \ifincsname
1334     \string#1%
1335     \expandafter\@gobble
1336   \else
1337     \ifx\protect\@typeset@protect
1338     \else
1339       \ifx\protect\@unexpandable@protect
1340         \noexpand#1%
1341       \else
1342         \protect#1%
1343       \fi
1344       \expandafter\expandafter\expandafter\@gobble
1345     \fi
1346   \fi}}
1347 \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⟩`.

```

1348 \newif\if@safe@actives
1349 \@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.

```

1350 \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`.

```

1351 \chardef\bbl@activated\z@
1352 \def\bbl@activate#1{%
1353   \chardef\bbl@activated\@ne
1354   \bbl@withactive{\expandafter\let\expandafter}#1%
1355   \csname bbl@active@\string#1\endcsname}
1356 \def\bbl@deactivate#1{%
1357   \chardef\bbl@activated\tw@
1358   \bbl@withactive{\expandafter\let\expandafter}#1%
1359   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1360 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1361 \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.

```

1362 \def\babel@texpdf#1#2#3#4{%
1363   \ifx\texorpdfstring\undefined
1364     \textormath{#1}{#3}%
1365   \else
1366     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1367     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1368   \fi}
1369 %
1370 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1371 \def\@decl@short#1#2#3\@nil#4{%
1372   \def\bbl@tempa{#3}%
1373   \ifx\bbl@tempa\empty
1374     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1375     \bbl@ifunset{#1@sh@\string#2@}{}%
1376     {\def\bbl@tempa{#4}%
1377      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1378      \else
1379        \bbl@info
1380        {Redefining #1 shorthand \string#2\%
1381         in language \CurrentOption}%
1382      \fi}%
1383     \@namedef{#1@sh@\string#2@}{#4}%
1384   \else
1385     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1386     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1387     {\def\bbl@tempa{#4}%
1388      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1389      \else
1390        \bbl@info
1391        {Redefining #1 shorthand \string#2\string#3\%
1392         in language \CurrentOption}%
1393      \fi}%
1394     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1395   \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.

```

1396 \def\textormath{%
1397   \ifmmode
1398     \expandafter\@secondoftwo
1399   \else
1400     \expandafter\@firstoftwo
1401   \fi}

```

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

```

1402 \def\user@group{user}
1403 \def\language@group{english} % TODO. I don't like defaults
1404 \def\system@group{system}

```

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

```

1405 \def\useshorthands{%
1406   \@ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1407 \def\bbl@usesesh@s#1{%
1408   \bbl@usesesh@x
1409   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1410   {#1}}
1411 \def\bbl@usesesh@x#1#2{%
1412   \bbl@ifshorthand{#2}%
1413   {\def\user@group{user}%
1414     \initiate@active@char{#2}%
1415     #1%
1416     \bbl@activate{#2}}%
1417   {\bbl@error
1418     {I can't declare a shorthand turned off (\string#2)}
1419     {Sorry, but you can't use shorthands which have been\\
1420       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.

```

1421 \def\user@language@group{user@\language@group}
1422 \def\bbl@set@user@generic#1#2{%
1423   \bbl@ifunset{user@generic@active#1}%
1424   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1425     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1426     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1427       \expandafter\noexpand\csname normal@char#1\endcsname}%
1428     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1429       \expandafter\noexpand\csname user@active#1\endcsname}}%
1430   \@empty}
1431 \newcommand\defineshorthand[3][user]{%
1432   \edef\bbl@tempa{\zap@space#1 \@empty}%
1433   \bbl@for\bbl@tempb\bbl@tempa{%
1434     \if*\expandafter\@car\bbl@tempb\@nil
1435       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1436       \@expandtwoargs
1437       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1438     \fi
1439     \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].

```

1440 \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".

```

1441 \def\aliasshorthand#1#2{%
1442   \bbl@ifshorthand{#2}%
1443   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1444     \ifx\document\@notprerr
1445       \@notshorthand{#2}%
1446     \else
1447       \initiate@active@char{#2}%
1448       \expandafter\let\csname active@char\string#2\expandafter\endcsname
1449         \csname active@char\string#1\endcsname
1450       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1451         \csname normal@char\string#1\endcsname
1452       \bbl@activate{#2}%
1453     \fi

```

```

1454 \fi}%
1455 {\bbl@error
1456 {Cannot declare a shorthand turned off (\string#2)}
1457 {Sorry, but you cannot use shorthands which have been\\%
1458 turned off in the package options}}}
```

\@notshorthand

```

1459 \def\@notshorthand#1{%
1460 \bbl@error{%
1461 The character '\string #1' should be made a shorthand character;\\%
1462 add the command \string\usesshorthands\string{#1\string} to
1463 the preamble.\\%
1464 I will ignore your instruction}%
1465 {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.

```

1466 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1467 \DeclareRobustCommand*\shorthandoff{%
1468 \ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1469 \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.

```

1470 \def\bbl@switch@sh#1#2{%
1471 \ifx#2\@nnil\else
1472 \bbl@ifunset{\bbl@active@\string#2}%
1473 {\bbl@error
1474 {I can't switch '\string#2' on or off--not a shorthand}%
1475 {This character is not a shorthand. Maybe you made\\%
1476 a typing mistake? I will ignore your instruction.}}%
1477 {\ifcase#1% off, on, off*
1478 \catcode`#2\relax
1479 \or
1480 \catcode`#2\active
1481 \bbl@ifunset{\bbl@shdef@\string#2}%
1482 {}%
1483 {\bbl@withactive{\expandafter\let\expandafter}#2%
1484 \csname bbl@shdef@\string#2\endcsname
1485 \bbl@csarg\let{shdef@\string#2}\relax}%
1486 \ifcase\bbl@activated\or
1487 \bbl@activate{#2}%
1488 \else
1489 \bbl@deactivate{#2}%
1490 \fi
1491 \or
1492 \bbl@ifunset{\bbl@shdef@\string#2}%
1493 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1494 {}%
1495 \csname bbl@oricat@\string#2\endcsname
1496 \csname bbl@oridef@\string#2\endcsname
1497 \fi}%
1498 \bbl@afterfi\bbl@switch@sh#1%
1499 \fi}
```

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

```

1500 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1501 \def\bbl@putsh#1{%
```

```

1502 \bbl@ifunset{bbl@active@\string#1}%
1503   {\bbl@putsh@i#1\@empty\@nnil}%
1504   {\csname bbl@active@\string#1\endcsname}}
1505 \def\bbl@putsh@i#1#2\@nnil{%
1506   \csname\language@group @sh@\string#1@%
1507     \ifx\@empty#2\else\string#2@\fi\endcsname}
1508 \ifx\bbl@opt@shorthands\@nnil\else
1509   \let\bbl@s@initiate@active@char\initiate@active@char
1510   \def\initiate@active@char#1{%
1511     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1512   \let\bbl@s@switch@sh\bbl@switch@sh
1513   \def\bbl@switch@sh#1#2{%
1514     \ifx#2\@nnil\else
1515       \bbl@afterfi
1516       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1517     \fi}
1518   \let\bbl@s@activate\bbl@activate
1519   \def\bbl@activate#1{%
1520     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1521   \let\bbl@s@deactivate\bbl@deactivate
1522   \def\bbl@deactivate#1{%
1523     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1524 \fi

```

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

```

1525 \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.

```

1526 \def\bbl@prim@s{%
1527   \prime\futurelet\@let@token\bbl@pr@m@s}
1528 \def\bbl@if@primes#1#2{%
1529   \ifx#1\@let@token
1530     \expandafter\@firstoftwo
1531   \else\ifx#2\@let@token
1532     \bbl@afterelse\expandafter\@firstoftwo
1533   \else
1534     \bbl@afterfi\expandafter\@secondoftwo
1535   \fi\fi}
1536 \begingroup
1537   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1538   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1539   \lowercase{%
1540     \gdef\bbl@pr@m@s{%
1541       \bbl@if@primes""%
1542       \pr@@@s
1543       {\bbl@if@primes*^\pr@@@t\egroup}}
1544 \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).

```

1545 \initiate@active@char{~}
1546 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1547 \bbl@activate{~}

```


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

```
1548 \expandafter\def\csname OT1dqpos\endcsname{127}
1549 \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

```
1550 \ifx\f@encoding\undefined
1551   \def\f@encoding{OT1}
1552 \fi
```

7.6 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.

```
1553 \bbl@trace{Language attributes}
1554 \newcommand\languageattribute[2]{%
1555   \def\bbl@tempc{#1}%
1556   \bbl@fixname\bbl@tempc
1557   \bbl@iflanguage\bbl@tempc{%
1558     \bbl@vforeach{#2}{%
```

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

```
1559     \ifx\bbl@known@attrs\undefined
1560       \in@false
1561     \else
1562       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1563     \fi
1564     \ifin@
1565       \bbl@warning{%
1566         You have more than once selected the attribute '##1'\%
1567         for language #1. Reported}%
1568     \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.

```
1569       \bbl@exp{%
1570         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1571       \edef\bbl@tempa{\bbl@tempc-##1}%
1572       \expandafter\bbl@ifknown@ttrrib\expandafter{\bbl@tempa}\bbl@attributes%
1573       {\csname\bbl@tempc @attr@##1\endcsname}%
1574       {\@attrerr{\bbl@tempc}{##1}}%
1575     \fi}}
1576 \@onlypreamble\languageattribute
```

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

```
1577 \newcommand*{\@attrerr}[2]{%
1578   \bbl@error
1579   {The attribute #2 is unknown for language #1.}%
1580   {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}`.

```
1581 \def\bbl@declare@ttribute#1#2#3{%
1582   \bbl@xin@{,#2,}{,\BabelModifiers,}%
```

```

1583 \ifin@
1584 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1585 \fi
1586 \bbl@add@list\bbl@attributes{#1-#2}%
1587 \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

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

```

1588 \def\bbl@ifattributeset#1#2#3#4{%
1589 \ifx\bbl@known@attribs\undefined
1590 \in@false
1591 \else
1592 \bbl@xin@{, #1-#2, }{, \bbl@known@attribs,}%
1593 \fi
1594 \ifin@
1595 \bbl@afterelse#3%
1596 \else
1597 \bbl@afterfi#4%
1598 \fi}

```

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

```

1599 \def\bbl@ifknown@ttrib#1#2{%
1600 \let\bbl@tempa\@secondoftwo
1601 \bbl@loopx\bbl@tempb{#2}{%
1602 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1603 \ifin@
1604 \let\bbl@tempa\@firstoftwo
1605 \else
1606 \fi}%
1607 \bbl@tempa}

```

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

```

1608 \def\bbl@clear@ttribs{%
1609 \ifx\bbl@attributes\undefined\else
1610 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1611 \expandafter\bbl@clear@ttrib\bbl@tempa.
1612 }%
1613 \let\bbl@attributes\undefined
1614 \fi}
1615 \def\bbl@clear@ttrib#1-#2.{%
1616 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1617 \AtBeginDocument{\bbl@clear@ttribs}

```

7.7 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`

```

1618 \bbl@trace{Macros for saving definitions}
1619 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1620 \newcount\babel@savecnt
1621 \babel@beginsave

```

`\babel@save` The macro `\babel@save⟨csmame⟩` saves the current meaning of the control sequence `⟨csmame⟩` to `\originalTeX`³². 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.

```

1622 \def\babel@save#1{%
1623   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1624   \toks@\expandafter{\originalTeX\let#1=}%
1625   \bbl@exp{%
1626     \def\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1627   \advance\babel@savecnt\@ne}
1628 \def\babel@savevariable#1{%
1629   \toks@\expandafter{\originalTeX #1=}%
1630   \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.

```

1631 \def\bbl@frenchspacing{%
1632   \ifnum\the\sfcode`\.=\@m
1633     \let\bbl@nonfrenchspacing\relax
1634   \else
1635     \frenchspacing
1636     \let\bbl@nonfrenchspacing\nonfrenchspacing
1637   \fi}
1638 \let\bbl@nonfrenchspacing\nonfrenchspacing
1639 \let\bbl@elt\relax
1640 \edef\bbl@fs@chars{%
1641   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1642   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1643   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1644 \def\bbl@pre@fs{%
1645   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1646   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1647 \def\bbl@post@fs{%
1648   \bbl@save@sfcodes
1649   \edef\bbl@tempa{\bbl@cl{frspc}}%
1650   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1651   \if u\bbl@tempa      % do nothing
1652   \else\if n\bbl@tempa % non french
1653     \def\bbl@elt##1##2##3{%
1654       \ifnum\sfcode`##1=##2\relax
1655         \babel@savevariable{\sfcode`##1}%
1656       \sfcode`##1=##3\relax
1657     \fi}%
1658     \bbl@fs@chars
1659   \else\if y\bbl@tempa % french
1660     \def\bbl@elt##1##2##3{%
1661       \ifnum\sfcode`##1=##3\relax
1662         \babel@savevariable{\sfcode`##1}%
1663       \sfcode`##1=##2\relax
1664     \fi}%

```

³²`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

```

1665 \bbl@fs@chars
1666 \fi\fi\fi}

```

7.8 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.

```

1667 \bbl@trace{Short tags}
1668 \def\babeltags#1{%
1669   \edef\bbl@tempa{\zap@space#1 \@empty}%
1670   \def\bbl@tempb##1=##2\@{%
1671     \edef\bbl@tempc{%
1672       \noexpand\newcommand
1673       \expandafter\noexpand\csname ##1\endcsname{%
1674         \noexpand\protect
1675         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1676       \noexpand\newcommand
1677       \expandafter\noexpand\csname text##1\endcsname{%
1678         \noexpand\foreignlanguage{##2}}}
1679   \bbl@tempc}%
1680 \bbl@for\bbl@tempa\bbl@tempa{%
1681   \expandafter\bbl@tempb\bbl@tempa\@}%

```

7.9 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.

```

1682 \bbl@trace{Hyphens}
1683 \@onlypreamble\babelhyphenation
1684 \AtEndOfPackage{%
1685   \newcommand\babelhyphenation[2][\@empty]{%
1686     \ifx\bbl@hyphenation@relax
1687       \let\bbl@hyphenation@\@empty
1688     \fi
1689     \ifx\bbl@hyphlist\@empty\else
1690       \bbl@warning{%
1691         You must not intermingle \string\selectlanguage\space and\%
1692         \string\babelhyphenation\space or some exceptions will not\%
1693         be taken into account. Reported}%
1694     \fi
1695     \ifx\@empty#1%
1696       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1697     \else
1698       \bbl@vforeach{#1}{%
1699         \def\bbl@tempa{##1}%
1700         \bbl@fixname\bbl@tempa
1701         \bbl@iflanguage\bbl@tempa{%
1702           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1703             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1704             {}%
1705             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1706             #2}}}%
1707       \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip Opt plus Opt`³³.

```

1708 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1709 \def\bbl@t@one{T1}
1710 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

³³`TEX` begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

`\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`.

```

1711 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1712 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1713 \def\bbl@hyphen{%
1714   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1715 \def\bbl@hyphen@i#1#2{%
1716   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
1717   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1718   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. `\nobreak` is always preceded by `\leavevmode`, in case the shorthand starts a paragraph.

```

1719 \def\bbl@usehyphen#1{%
1720   \leavevmode
1721   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1722   \nobreak\hskip\z@skip}
1723 \def\bbl@@usehyphen#1{%
1724   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1725 \def\bbl@hyphenchar{%
1726   \ifnum\hyphenchar\font=\m@ne
1727     \babelnullhyphen
1728   \else
1729     \char\hyphenchar\font
1730   \fi}

```

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

```

1731 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1732 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1733 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1734 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1735 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1736 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1737 \def\bbl@hy@repeat{%
1738   \bbl@usehyphen{%
1739     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1740 \def\bbl@hy@@repeat{%
1741   \bbl@usehyphen{%
1742     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1743 \def\bbl@hy@empty{\hskip\z@skip}
1744 \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.

```

1745 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

```

7.10 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.

```

1746 \bbl@trace{Multiencoding strings}
1747 \def\bbl@tglobal#1{\global\let#1#1}
1748 \def\bbl@recatcode#1{% TODO. Used only once?
1749   \@tempcnta="7F
1750   \def\bbl@tempa{%
1751     \ifnum\@tempcnta>"FF\else
1752       \catcode\@tempcnta=#1\relax
1753       \advance\@tempcnta\@ne
1754       \expandafter\bbl@tempa
1755     \fi}%
1756   \bbl@tempa}

```

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

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

and starts over (and similarly when lowercasing).

```

1757 \ifpackagewith{babel}{nocase}%
1758   {\let\bbl@patchuclc\relax}%
1759   {\def\bbl@patchuclc{%
1760     \global\let\bbl@patchuclc\relax
1761     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1762     \gdef\bbl@uclc##1{%
1763       \let\bbl@encoded\bbl@encoded@uclc
1764       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1765       {##1}%
1766       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1767        \csname\language @bbl@uclc\endcsname}%
1768        {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1769     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1770     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1771 <<More package options>> ≡
1772 \DeclareOption{nocase}{}
1773 <</More package options>>

```

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

```

1774 <<More package options>> ≡
1775 \let\bbl@opt@strings\@nnil % accept strings=value
1776 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1777 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1778 \def\BabelStringsDefault{generic}
1779 <</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.

```

1780 \@onlypreamble\StartBabelCommands
1781 \def\StartBabelCommands{%
1782   \begingroup
1783   \bbl@recatcode{11}%
1784   <<Macros local to BabelCommands>>
1785   \def\bbl@provstring##1##2{%
1786     \providecommand##1{##2}%

```

```

1787 \bbl@tglobal##1}%
1788 \global\let\bbl@scafter\@empty
1789 \let\StartBabelCommands\bbl@startcmds
1790 \ifx\BabelLanguages\relax
1791 \let\BabelLanguages\CurrentOption
1792 \fi
1793 \begingroup
1794 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1795 \StartBabelCommands}
1796 \def\bbl@startcmds{%
1797 \ifx\bbl@screset\@nnil\else
1798 \bbl@usehooks{stopcommands}{}%
1799 \fi
1800 \endgroup
1801 \begingroup
1802 \@ifstar
1803 {\ifx\bbl@opt@strings\@nnil
1804 \let\bbl@opt@strings\BabelStringsDefault
1805 \fi
1806 \bbl@startcmds@i}%
1807 \bbl@startcmds@i}
1808 \def\bbl@startcmds@i#1#2{%
1809 \edef\bbl@L{\zap@space#1 \@empty}%
1810 \edef\bbl@G{\zap@space#2 \@empty}%
1811 \bbl@startcmds@ii}
1812 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1813 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1814 \let\SetString\@gobbletwo
1815 \let\bbl@stringdef\@gobbletwo
1816 \let\AfterBabelCommands\@gobble
1817 \ifx\@empty#1%
1818 \def\bbl@sc@label{generic}%
1819 \def\bbl@encstring##1##2{%
1820 \ProvideTextCommandDefault##1{##2}%
1821 \bbl@tglobal##1%
1822 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1823 \let\bbl@sctest\in@true
1824 \else
1825 \let\bbl@sc@charset\space % <- zapped below
1826 \let\bbl@sc@fontenc\space % <- " "
1827 \def\bbl@tempa##1=##2\@nil{%
1828 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1829 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1830 \def\bbl@tempa##1 ##2{% space -> comma
1831 ##1%
1832 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1833 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1834 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1835 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1836 \def\bbl@encstring##1##2{%
1837 \bbl@foreach\bbl@sc@fontenc{%
1838 \bbl@ifunset{T@###1}%
1839 }%

```

```

1840         {\ProvideTextCommand##1{####1}{##2}%
1841         \bbl@toglobal##1%
1842         \expandafter
1843         \bbl@toglobal\csname####1\string##1\endcsname}}}%
1844     \def\bbl@sctest{%
1845         \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
1846     \fi
1847     \ifx\bbl@opt@strings\@nnil          % ie, no strings key -> defaults
1848     \else\ifx\bbl@opt@strings\relax      % ie, strings=encoded
1849         \let\AfterBabelCommands\bbl@aftercmds
1850         \let\SetString\bbl@setstring
1851         \let\bbl@stringdef\bbl@encstring
1852     \else          % ie, strings=value
1853     \bbl@sctest
1854     \ifin@
1855         \let\AfterBabelCommands\bbl@aftercmds
1856         \let\SetString\bbl@setstring
1857         \let\bbl@stringdef\bbl@provstring
1858     \fi\fi\fi
1859     \bbl@scswitch
1860     \ifx\bbl@G\@empty
1861         \def\SetString##1##2{%
1862             \bbl@error{Missing group for string \string##1}%
1863             {You must assign strings to some category, typically\\%
1864             captions or extras, but you set none}}}%
1865     \fi
1866     \ifx\@empty#1%
1867         \bbl@usehooks{defaultcommands}{}%
1868     \else
1869         \@expandtwoargs
1870         \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1871     \fi}

```

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

```

1872 \def\bbl@forlang#1#2{%
1873     \bbl@for#1\bbl@L{%
1874         \bbl@xin@{, #1,}{\BabelLanguages,}%
1875         \ifin#2\relax\fi}}
1876 \def\bbl@scswitch{%
1877     \bbl@forlang\bbl@tempa{%
1878         \ifx\bbl@G\@empty\else
1879             \ifx\SetString@gobbletwo\else
1880                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1881                 \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1882             \ifin@ \else
1883                 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1884                 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1885             \fi
1886         \fi
1887     \fi}}
1888 \AtEndOfPackage{%
1889     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1890     \let\bbl@scswitch\relax}
1891 \onlypreamble\EndBabelCommands
1892 \def\EndBabelCommands{%
1893     \bbl@usehooks{stopcommands}{}%

```



```

1894 \endgroup
1895 \endgroup
1896 \bbl@scafter}
1897 \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.

```

1898 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1899 \bbl@forlang\bbl@tempa{%
1900 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1901 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1902 {\bbl@exp{%
1903 \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
1904 }%
1905 \def\BabelString{#2}%
1906 \bbl@usehooks{stringprocess}{}%
1907 \expandafter\bbl@stringdef
1908 \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.

```

1909 \ifx\bbl@opt@strings\relax
1910 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1911 \bbl@patchuclc
1912 \let\bbl@encoded\relax
1913 \def\bbl@encoded@uclc#1{%
1914 \@inmathwarn#1%
1915 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1916 \expandafter\ifx\csname ?\string#1\endcsname\relax
1917 \TextSymbolUnavailable#1%
1918 \else
1919 \csname ?\string#1\endcsname
1920 \fi
1921 \else
1922 \csname\cf@encoding\string#1\endcsname
1923 \fi}
1924 \else
1925 \def\bbl@scset#1#2{\def#1{#2}}
1926 \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.

```

1927 << *Macros local to BabelCommands >> ≡
1928 \def\SetStringLoop##1##2{%
1929 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1930 \count@\z@
1931 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1932 \advance\count@\@ne
1933 \toks@\expandafter{\bbl@tempa}%
1934 \bbl@exp{%
1935 \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1936 \count@=\the\count@\relax}}}%
1937 <</Macros local to BabelCommands >>

```

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

```

1938 \def\bbl@aftercmds#1{%

```

```

1939 \toks@\expandafter{\bbl@scafter#1}%
1940 \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 `\uc\lclist` to the parsing command.

```

1941 <<(*Macros local to BabelCommands)>> ≡
1942 \newcommand\SetCase[3][]{%
1943   \bbl@patchuc\l
1944   \bbl@forlang\bbl@tempa{%
1945     \expandafter\bbl@encstring
1946     \csname\bbl@tempa @bbl@uc\l\endcsname{\bbl@tempa##1}%
1947     \expandafter\bbl@encstring
1948     \csname\bbl@tempa @bbl@uc\l\endcsname{##2}%
1949     \expandafter\bbl@encstring
1950     \csname\bbl@tempa @bbl@lc\l\endcsname{##3}}}%
1951 <</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.

```

1952 <<(*Macros local to BabelCommands)>> ≡
1953 \newcommand\SetHyphenMap[1]{%
1954   \bbl@forlang\bbl@tempa{%
1955     \expandafter\bbl@stringdef
1956     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1957 <</Macros local to BabelCommands>>

```

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

```

1958 \newcommand\BabelLower[2]{% one to one.
1959   \ifnum\lccode#1=#2\else
1960     \babel@savevariable{\lccode#1}%
1961     \lccode#1=#2\relax
1962   \fi}
1963 \newcommand\BabelLowerMM[4]{% many-to-many
1964   \@tempcnta=#1\relax
1965   \@tempcntb=#4\relax
1966   \def\bbl@tempa{%
1967     \ifnum\@tempcnta>#2\else
1968       \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1969       \advance\@tempcnta#3\relax
1970       \advance\@tempcntb#3\relax
1971       \expandafter\bbl@tempa
1972     \fi}%
1973   \bbl@tempa}
1974 \newcommand\BabelLowerMO[4]{% many-to-one
1975   \@tempcnta=#1\relax
1976   \def\bbl@tempa{%
1977     \ifnum\@tempcnta>#2\else
1978       \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1979       \advance\@tempcnta#3
1980       \expandafter\bbl@tempa
1981     \fi}%
1982   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1983 <<(*More package options)>> ≡
1984 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1985 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1986 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1987 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@}
1988 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1989 <</More package options>>

```

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

```

1990 \AtEndOfPackage{%
1991   \ifx\bbbl@opt@hyphenmap\undefined
1992     \bbbl@xin@{,}{\bbbl@language@opts}%
1993     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1994   \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.

```

1995 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
1996   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1997 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1998   \bbbl@trim@def\bbbl@tempa{#2}%
1999   \bbbl@xin@{.template}{\bbbl@tempa}%
2000   \ifin@
2001     \bbbl@ini@captions@template{#3}{#1}%
2002   \else
2003     \edef\bbbl@tempd{%
2004       \expandafter\expandafter\expandafter
2005       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2006     \bbbl@xin@
2007       {\expandafter\string\csname #2name\endcsname}%
2008       {\bbbl@tempd}%
2009     \ifin@ % Renew caption
2010       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
2011       \ifin@
2012         \bbbl@exp{%
2013           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2014           {\bbbl@scset\<#2name>\<#1#2name>}%
2015           {}}%
2016         \else % Old way converts to new way
2017           \bbbl@ifunset{#1#2name}%
2018             {\bbbl@exp{%
2019               \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2020               \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2021               {\def\<#2name>{\<#1#2name>}}%
2022               {}}}%
2023             {}%
2024         \fi
2025       \else
2026         \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2027         \ifin@ % New way
2028           \bbbl@exp{%
2029             \\bbbl@add\<captions#1>{\bbbl@scset\<#2name>\<#1#2name>}%
2030             \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2031             {\bbbl@scset\<#2name>\<#1#2name>}%
2032             {}}%
2033           \else % Old way, but defined in the new way
2034             \bbbl@exp{%
2035               \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2036               \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2037               {\def\<#2name>{\<#1#2name>}}%
2038               {}}%
2039             \fi%
2040           \fi
2041           \@namedef{#1#2name}{#3}%
2042           \toks@ \expandafter{\bbbl@captionslist}%
2043           \bbbl@exp{\in@{\<#2name>}{\the\toks@}}%
2044         \ifin@ \else
2045           \bbbl@exp{\bbbl@add\bbbl@captionslist{\<#2name>}}%
2046           \bbbl@tglobal\bbbl@captionslist
2047         \fi

```

```

2048 \fi}
2049 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

7.11 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.

```

2050 \bbl@trace{Macros related to glyphs}
2051 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2052 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2053 \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.

```

2054 \def\save@sf@q#1{\leavevmode
2055 \begingroup
2056 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2057 \endgroup}

```

7.12 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`.

7.12.1 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.

```

2058 \ProvideTextCommand{\quotedblbase}{OT1}{%
2059 \save@sf@q{\set@low@box{\textquotedblright\}}%
2060 \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.

```

2061 \ProvideTextCommandDefault{\quotedblbase}{%
2062 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2063 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2064 \save@sf@q{\set@low@box{\textquoteright\}}%
2065 \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.

```

2066 \ProvideTextCommandDefault{\quotesinglbase}{%
2067 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2068 \ProvideTextCommand{\guillemetleft}{OT1}{%
2069 \ifmmode
2070 \ll
2071 \else
2072 \save@sf@q{\nobreak
2073 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2074 \fi}
2075 \ProvideTextCommand{\guillemetright}{OT1}{%
2076 \ifmmode
2077 \gg
2078 \else
2079 \save@sf@q{\nobreak
2080 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2081 \fi}
2082 \ProvideTextCommand{\guillemotleft}{OT1}{%

```

```

2083 \ifmmode
2084 \ll
2085 \else
2086 \save@sf@q{\nobreak
2087 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2088 \fi}
2089 \ProvideTextCommand{\guillemotright}{OT1}{%
2090 \ifmmode
2091 \gg
2092 \else
2093 \save@sf@q{\nobreak
2094 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2095 \fi}

```

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

```

2096 \ProvideTextCommandDefault{\guillemetleft}{%
2097 \UseTextSymbol{OT1}{\guillemetleft}}
2098 \ProvideTextCommandDefault{\guillemetright}{%
2099 \UseTextSymbol{OT1}{\guillemetright}}
2100 \ProvideTextCommandDefault{\guillemotleft}{%
2101 \UseTextSymbol{OT1}{\guillemotleft}}
2102 \ProvideTextCommandDefault{\guillemotright}{%
2103 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2104 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2105 \ifmmode
2106 <%
2107 \else
2108 \save@sf@q{\nobreak
2109 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2110 \fi}
2111 \ProvideTextCommand{\guilsinglright}{OT1}{%
2112 \ifmmode
2113 >%
2114 \else
2115 \save@sf@q{\nobreak
2116 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2117 \fi}

```

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

```

2118 \ProvideTextCommandDefault{\guilsinglleft}{%
2119 \UseTextSymbol{OT1}{\guilsinglleft}}
2120 \ProvideTextCommandDefault{\guilsinglright}{%
2121 \UseTextSymbol{OT1}{\guilsinglright}}

```

7.12.2 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.

```

2122 \DeclareTextCommand{\ij}{OT1}{%
2123 i\kern-0.02em\bbl@allowhyphens j}
2124 \DeclareTextCommand{\IJ}{OT1}{%
2125 I\kern-0.02em\bbl@allowhyphens J}
2126 \DeclareTextCommand{\ij}{T1}{\char188}
2127 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2128 \ProvideTextCommandDefault{\ij}{%
2129 \UseTextSymbol{OT1}{\ij}}
2130 \ProvideTextCommandDefault{\IJ}{%
2131 \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).

```

2132 \def\crrtic@{\hrule height0.1ex width0.3em}
2133 \def\crttic@{\hrule height0.1ex width0.33em}
2134 \def\ddj@{%
2135   \setbox0\hbox{d}\dimen@=\ht0
2136   \advance\dimen@1ex
2137   \dimen@.45\dimen@
2138   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2139   \advance\dimen@ii.5ex
2140   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2141 \def\DDJ@{%
2142   \setbox0\hbox{D}\dimen@=.55\ht0
2143   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2144   \advance\dimen@ii.15ex % correction for the dash position
2145   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2146   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2147   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2148 %
2149 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2150 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2151 \ProvideTextCommandDefault{\dj}{%
2152   \UseTextSymbol{OT1}{\dj}}
2153 \ProvideTextCommandDefault{\DJ}{%
2154   \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.

```

2155 \DeclareTextCommand{\SS}{OT1}{\SS}
2156 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

7.12.3 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 2157 \ProvideTextCommandDefault{\glq}{%
2158   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2159 \ProvideTextCommand{\grq}{T1}{%
2160   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2161 \ProvideTextCommand{\grq}{TU}{%
2162   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2163 \ProvideTextCommand{\grq}{OT1}{%
2164   \save@sf@q{\kern-.0125em
2165     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2166     \kern.07em\relax}}
2167 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 2168 \ProvideTextCommandDefault{\glqq}{%
2169   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2170 \ProvideTextCommand{\grqq}{T1}{%
2171   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}

```

```

2172 \ProvideTextCommand{\grqq}{TU}{%
2173   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2174 \ProvideTextCommand{\grqq}{OT1}{%
2175   \save@sf@q{\kern-.07em
2176     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2177     \kern.07em\relax}}
2178 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

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).

```

2210 \AtBeginDocument{%
2211   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2212   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2213   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2214   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
2215   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2216   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2217   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2218   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2219   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2220   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2221   \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.

```

2222 \ifx\l@english\@undefined
2223   \chardef\l@english\z@
2224 \fi
2225 % The following is used to cancel rules in ini files (see Amharic).
2226 \ifx\l@unhyphenated\@undefined
2227   \newlanguage\l@unhyphenated
2228 \fi

```

7.13 Layout

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

```

2229 \bbl@trace{Bidi layout}
2230 \providecommand\IfBabelLayout[3]{#3}%
2231 \newcommand\BabelPatchSection[1]{%
2232   \@ifundefined{#1}{}{%
2233     \bbl@exp{\let<bbl@ss@#1>\<#1>}%
2234     \@namedef{#1}{%
2235       \ifstar{\bbl@presec@#1}%
2236       {\@dblarg{\bbl@presec@x{#1}}}}%
2237 \def\bbl@presec@x#1[#2]#3{%
2238   \bbl@exp{%
2239     \\\select@language@x{\bbl@main@language}%
2240     \\\bbl@cs{sspre@#1}%
2241     \\\bbl@cs{ss@#1}%
2242     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2243     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2244     \\\select@language@x{\language}}%
2245 \def\bbl@presec@s#1#2{%
2246   \bbl@exp{%
2247     \\\select@language@x{\bbl@main@language}%
2248     \\\bbl@cs{sspre@#1}%
2249     \\\bbl@cs{ss@#1}*%
2250     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2251     \\\select@language@x{\language}}%
2252 \IfBabelLayout{sectioning}%
2253   {\BabelPatchSection{part}%
2254    \BabelPatchSection{chapter}%
2255    \BabelPatchSection{section}%
2256    \BabelPatchSection{subsection}%
2257    \BabelPatchSection{subsubsection}%
2258    \BabelPatchSection{paragraph}%

```



```

2259 \BabelPatchSection{subparagraph}%
2260 \def\babel@toc#1{%
2261   \select@language{x{\bbl@main@language}}{}
2262 \IfBabelLayout{captions}%
2263   {\BabelPatchSection{caption}}{}

```

7.14 Load engine specific macros

Some macros are not defined in all engines, so, after loading the files define them if necessary to raise an error.

```

2264 \bbl@trace{Input engine specific macros}
2265 \ifcase\bbl@engine
2266   \input txtbabel.def
2267 \or
2268   \input luababel.def
2269 \or
2270   \input xebabel.def
2271 \fi
2272 \providecommand\babelfont{%
2273   \bbl@error
2274   {This macro is available only in LuaLaTeX and XeLaTeX.}%
2275   {Consider switching to these engines.}}
2276 \providecommand\babelprehyphenation{%
2277   \bbl@error
2278   {This macro is available only in LuaLaTeX.}%
2279   {Consider switching to that engine.}}
2280 \ifx\babelposthyphenation\@undefined
2281   \let\babelposthyphenation\babelprehyphenation
2282   \let\babelpatterns\babelprehyphenation
2283   \let\babelcharproperty\babelprehyphenation
2284 \fi

```

7.15 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.

```

2285 \bbl@trace{Creating languages and reading ini files}
2286 \let\bbl@extend@ini\@gobble
2287 \newcommand\babelprovide[2][]{%
2288   \let\bbl@savelangname\language
2289   \edef\bbl@savlocaleid{\the\localeid}%
2290   % Set name and locale id
2291   \edef\language{#2}%
2292   \bbl@id@assign
2293   % Initialize keys
2294   \bbl@vforeach{captions,date,import,main,script,language,%
2295     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2296     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2297     Alph,labels,labels*,calendar}%
2298     {\bbl@csarg\let{KVP@##1}\@nnil}%
2299   \global\let\bbl@release@transforms\@empty
2300   \let\bbl@calendars\@empty
2301   \global\let\bbl@inidata\@empty
2302   \global\let\bbl@extend@ini\@gobble
2303   \gdef\bbl@key@list{;}%
2304   \bbl@forkv{#1}{% TODO - error handling
2305     \in@{/}{##1}%
2306     \ifin@
2307       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2308       \bbl@renewinikey##1\@{##2}%
2309     \else

```

```

2310 \bbl@csarg\ifx{KVP@##1}\@nnil\else
2311 \bbl@error
2312 {Unknown key '##1' in \string\babelprovide}%
2313 {See the manual for valid keys}%
2314 \fi
2315 \bbl@csarg\def{KVP@##1}{##2}%
2316 \fi}%
2317 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2318 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2319 % == init ==
2320 \ifx\bbl@screset\@undefined
2321 \bbl@ldfinit
2322 \fi
2323 % ==
2324 \let\bbl@lbfkflag\relax % \@empty = do setup linebreak
2325 \ifcase\bbl@howloaded
2326 \let\bbl@lbfkflag\@empty % new
2327 \else
2328 \ifx\bbl@KVP@hyphenrules\@nnil\else
2329 \let\bbl@lbfkflag\@empty
2330 \fi
2331 \ifx\bbl@KVP@import\@nnil\else
2332 \let\bbl@lbfkflag\@empty
2333 \fi
2334 \fi
2335 % == import, captions ==
2336 \ifx\bbl@KVP@import\@nnil\else
2337 \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
2338 {\ifx\bbl@initoload\relax
2339 \begin{group}
2340 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2341 \bbl@input@texini{##2}%
2342 \end{group}
2343 \else
2344 \xdef\bbl@KVP@import{\bbl@initoload}%
2345 \fi}%
2346 {}%
2347 \fi
2348 \ifx\bbl@KVP@captions\@nnil
2349 \let\bbl@KVP@captions\bbl@KVP@import
2350 \fi
2351 % ==
2352 \ifx\bbl@KVP@transforms\@nnil\else
2353 \bbl@replace\bbl@KVP@transforms{ }{,}%
2354 \fi
2355 % == Load ini ==
2356 \ifcase\bbl@howloaded
2357 \bbl@provide@new{##2}%
2358 \else
2359 \bbl@ifblank{##1}%
2360 {}% With \bbl@load@basic below
2361 {\bbl@provide@renew{##2}}%
2362 \fi
2363 % Post tasks
2364 % -----
2365 % == subsequent calls after the first provide for a locale ==
2366 \ifx\bbl@inidata\@empty\else
2367 \bbl@extend@ini{##2}%
2368 \fi
2369 % == ensure captions ==
2370 \ifx\bbl@KVP@captions\@nnil\else
2371 \bbl@ifunset{\bbl@extracaps@#2}%
2372 {\bbl@exp{\@babelensure[exclude=\@today]{##2}}}%

```

```

2373     {\bbl@exp{\babelensure[exclude=\today,
2374               include=\bbl@extracaps@#2]}{#2}}%
2375 \bbl@ifunset{\bbl@ensure@language}%
2376 {\bbl@exp%
2377   \DeclareRobustCommand\<bbl@ensure@language>[1]{%
2378     \foreignlanguage{language}%
2379     {###1}}}%
2380 }%
2381 \bbl@exp%
2382   \bbl@tglobal\<bbl@ensure@language>%
2383   \bbl@tglobal\<bbl@ensure@language\space>%
2384 \fi
2385 % ==
2386 % At this point all parameters are defined if 'import'. Now we
2387 % execute some code depending on them. But what about if nothing was
2388 % imported? We just set the basic parameters, but still loading the
2389 % whole ini file.
2390 \bbl@load@basic{#2}%
2391 % == script, language ==
2392 % Override the values from ini or defines them
2393 \ifx\bbl@KVP@script\@nnil\else
2394   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2395 \fi
2396 \ifx\bbl@KVP@language\@nnil\else
2397   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2398 \fi
2399 \ifcase\bbl@engine\or
2400   \bbl@ifunset{\bbl@chrng@language}{}%
2401   {\directlua{
2402     Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2403 \fi
2404 % == onchar ==
2405 \ifx\bbl@KVP@onchar\@nnil\else
2406   \bbl@luahyphenate
2407   \bbl@exp%
2408     \AddToHook{env/document/before}{\select@language{#2}}}%
2409   \directlua{
2410     if Babel.locale_mapped == nil then
2411       Babel.locale_mapped = true
2412       Babel.linebreaking.add_before(Babel.locale_map)
2413       Babel.loc_to_scr = {}
2414       Babel.chr_to_loc = Babel.chr_to_loc or {}
2415     end}%
2416 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2417 \ifin@
2418   \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2419     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2420   \fi
2421   \bbl@exp{\bbl@add\bbl@starthyphens
2422     {\bbl@patterns@lua{language}}}%
2423   % TODO - error/warning if no script
2424   \directlua{
2425     if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2426       Babel.loc_to_scr[\the\localeid] =
2427         Babel.script_blocks['\bbl@cl{sbcpr}']
2428       Babel.locale_props[\the\localeid].lc = \the\localeid\space
2429       Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@language}\space
2430     end
2431   }%
2432 \fi
2433 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2434 \ifin@
2435   \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys@language}}}%

```

```

2436 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
2437 \directlua{
2438   if Babel.script_blocks['\bbl@cl{sbc}'] then
2439     Babel.loc_to_scr[\the\localeid] =
2440       Babel.script_blocks['\bbl@cl{sbc}']
2441   end}%
2442 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2443 \AtBeginDocument{%
2444   \bbl@patchfont{\bbl@mapselect}%
2445   {\selectfont}%
2446 \def\bbl@mapselect{%
2447   \let\bbl@mapselect\relax
2448   \edef\bbl@prefontid{\fontid\font}%
2449 \def\bbl@mapdir##1{%
2450   {\def\language{##1}%
2451   \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2452   \bbl@switchfont
2453   \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2454     \directlua{
2455       Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2456       [\bbl@prefontid] = \fontid\font\space}%
2457     \fi}}%
2458   \fi
2459   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2460 \fi
2461 % TODO - catch non-valid values
2462 \fi
2463 % == mapfont ==
2464 % For bidi texts, to switch the font based on direction
2465 \ifx\bbl@KVP@mapfont\@nnil\else
2466   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2467   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
2468     mapfont. Use 'direction'.%
2469     {See the manual for details.}}}%
2470 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
2471 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
2472 \ifx\bbl@mapselect\undefined % TODO. See onchar.
2473 \AtBeginDocument{%
2474   \bbl@patchfont{\bbl@mapselect}%
2475   {\selectfont}%
2476 \def\bbl@mapselect{%
2477   \let\bbl@mapselect\relax
2478   \edef\bbl@prefontid{\fontid\font}%
2479 \def\bbl@mapdir##1{%
2480   {\def\language{##1}%
2481   \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2482   \bbl@switchfont
2483   \directlua{Babel.fontmap
2484     [\the\csname bbl@wdir@##1\endcsname]%
2485     [\bbl@prefontid]=\fontid\font}}}%
2486   \fi
2487   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2488 \fi
2489 % == Line breaking: intraspace, intrapenalty ==
2490 % For CJK, East Asian, Southeast Asian, if interspace in ini
2491 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2492   \bbl@csarg\edef{intsp#2}{\bbl@KVP@intraspace}%
2493 \fi
2494 \bbl@provide@intraspace
2495 % == Line breaking: CJK quotes ==
2496 \ifcase\bbl@engine\or
2497   \bbl@xin@{/c}{\bbl@cl{lnbrk}}%
2498 \ifin@

```

```

2499 \bbl@ifunset{bbl@quote@\languagename}{}%
2500 {\directlua{
2501   Babel.locale_props[\the\localeid].CJK_quotes = {}
2502   local cs = 'op'
2503   for c in string.utfvalues(
2504     [[\csname bbl@quote@\languagename\endcsname]]) do
2505     if Babel.CJK_characters[c].c == 'qu' then
2506       Babel.locale_props[\the\localeid].CJK_quotes[c] = cs
2507     end
2508     cs = (cs == 'op') and 'cl' or 'op'
2509   end
2510 }}%
2511 \fi
2512 \fi
2513 % == Line breaking: justification ==
2514 \ifx\bbl@KVP@justification\@nnil\else
2515   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2516 \fi
2517 \ifx\bbl@KVP@linebreaking\@nnil\else
2518   \bbl@xin@{\bbl@KVP@linebreaking,}{,elongated,kashida,CJK,unhyphenated,}%
2519 \fin@
2520   \bbl@csarg\xdef
2521     {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2522 \fi
2523 \fi
2524 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
2525 \fin@\else\bbl@xin@{/k}{/\bbl@cl{\lnbrk}}\fi
2526 \fin@\bbl@arabicjust\fi
2527 % == Line breaking: hyphenate.other.(locale|script) ==
2528 \ifx\bbl@lbkflag\@empty
2529   \bbl@ifunset{bbl@hyotl@\languagename}{}%
2530   {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2531     \bbl@startcommands*\languagename}{}%
2532     \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2533       \ifcase\bbl@engine
2534         \ifnum##1<257
2535           \SetHyphenMap{\BabelLower{##1}{##1}}%
2536         \fi
2537       \else
2538         \SetHyphenMap{\BabelLower{##1}{##1}}%
2539       \fi}%
2540   \bbl@endcommands}%
2541 \bbl@ifunset{bbl@hyots@\languagename}{}%
2542 {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2543   \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2544     \ifcase\bbl@engine
2545       \ifnum##1<257
2546         \global\lccode##1=##1\relax
2547       \fi
2548     \else
2549       \global\lccode##1=##1\relax
2550     \fi}}%
2551 \fi
2552 % == Counters: maparabic ==
2553 % Native digits, if provided in ini (TeX level, xe and lua)
2554 \ifcase\bbl@engine\else
2555   \bbl@ifunset{bbl@dgnat@\languagename}{}%
2556   {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2557     \expandafter\expandafter\expandafter
2558     \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2559     \ifx\bbl@KVP@maparabic\@nnil\else
2560       \ifx\bbl@latinarabic\undefined
2561         \expandafter\let\expandafter\@arabic

```

```

2562         \csname bbl@counter@\language\endcsname
2563     \else % ie, if layout=counters, which redefines \@arabic
2564         \expandafter\let\expandafter\bbl@latin@arabic
2565         \csname bbl@counter@\language\endcsname
2566     \fi
2567 \fi
2568 \fi}%
2569 \fi
2570 % == Counters: mapdigits ==
2571 % Native digits (lua level).
2572 \ifodd\bbl@engine
2573     \ifx\bbl@KVP@mapdigits\@nnil\else
2574         \bbl@ifunset{\bbl@dgnat@\language}{}%
2575         {\RequirePackage{luatexbase}%
2576         \bbl@activate@preotf
2577         \directlua{
2578             Babel = Babel or {} %%% -> presets in luababel
2579             Babel.digits_mapped = true
2580             Babel.digits = Babel.digits or {}
2581             Babel.digits[\the\localeid] =
2582                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2583             if not Babel.numbers then
2584                 function Babel.numbers(head)
2585                     local LOCALE = Babel.attr_locale
2586                     local GLYPH = node.id'glyph'
2587                     local inmath = false
2588                     for item in node.traverse(head) do
2589                         if not inmath and item.id == GLYPH then
2590                             local temp = node.get_attribute(item, LOCALE)
2591                             if Babel.digits[temp] then
2592                                 local chr = item.char
2593                                 if chr > 47 and chr < 58 then
2594                                     item.char = Babel.digits[temp][chr-47]
2595                                 end
2596                             end
2597                         elseif item.id == node.id'math' then
2598                             inmath = (item.subtype == 0)
2599                         end
2600                     end
2601                     return head
2602                 end
2603             end
2604         } }%
2605     \fi
2606 \fi
2607 % == Counters: alph, Alph ==
2608 % What if extras<lang> contains a \babel@save\@alph? It won't be
2609 % restored correctly when exiting the language, so we ignore
2610 % this change with the \bbl@alph@saved trick.
2611 \ifx\bbl@KVP@alph\@nnil\else
2612     \bbl@extras@wrap{\bbl@alph@saved}%
2613     {\let\bbl@alph@saved\@alph}%
2614     {\let\@alph\bbl@alph@saved
2615     \babel@save\@alph}%
2616 \bbl@exp{%
2617     \bbl@add\<extras\language>{%
2618     \let\@alph\bbl@cntr\bbl@KVP@alph @\language}}}%
2619 \fi
2620 \ifx\bbl@KVP@Alph\@nnil\else
2621     \bbl@extras@wrap{\bbl@Alph@saved}%
2622     {\let\bbl@Alph@saved\@Alph}%
2623     {\let\@Alph\bbl@Alph@saved
2624     \babel@save\@Alph}%

```

```

2625 \bbl@exp{%
2626   \\\bbl@add\<extras\language>{%
2627     \let\\@Alph\<bbl@cntr@bbl@KVP@Alph @\language>}}%
2628 \fi
2629 % == Calendars ==
2630 \ifx\bbl@KVP@calendar\@nnil
2631   \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2632 \fi
2633 \def\bbl@tempe##1 ##2\@{% Get first calendar
2634   \def\bbl@tempa{##1}}%
2635   \bbl@exp{\\ \bbl@tempe\bbl@KVP@calendar\space\\ \@}%
2636 \def\bbl@tempe##1.##2.##3\@{%
2637   \def\bbl@tempc{##1}%
2638   \def\bbl@tempb{##2}}%
2639 \expandafter\bbl@tempe\bbl@tempa..\@
2640 \bbl@csarg\edef{calpr@\language}{%
2641   \ifx\bbl@tempc\@empty\else
2642     calendar=\bbl@tempc
2643   \fi
2644   \ifx\bbl@tempb\@empty\else
2645     ,variant=\bbl@tempb
2646   \fi}%
2647 % == require.babel in ini ==
2648 % To load or reload the babel-*.tex, if require.babel in ini
2649 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2650   \bbl@ifunset{bbl@rqtex@\language}{}%
2651   {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
2652     \let\BabelBeforeIni\@gobbles
2653     \chardef\atcatcode=\catcode`@
2654     \catcode`\@=11\relax
2655     \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2656     \catcode`\@=\atcatcode
2657     \let\atcatcode\relax
2658     \global\bbl@csarg\let{rqtex@\language}\relax
2659   \fi}%
2660 \bbl@foreach\bbl@calendars{%
2661   \bbl@ifunset{bbl@ca##1}{%
2662     \chardef\atcatcode=\catcode`@
2663     \catcode`\@=11\relax
2664     \InputIfFileExists{babel-ca-##1.tex}{}}%
2665     \catcode`\@=\atcatcode
2666     \let\atcatcode\relax}%
2667   {}}%
2668 \fi
2669 % == frenchspacing ==
2670 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2671 \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2672 \ifin@
2673   \bbl@extras@wrap{\\ \bbl@pre@fs}%
2674   {\bbl@pre@fs}%
2675   {\bbl@post@fs}%
2676 \fi
2677 % == Release saved transforms ==
2678 \bbl@release@transforms\relax % \relax closes the last item.
2679 % == main ==
2680 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2681   \let\language\bbl@savelangname
2682   \chardef\localeid\bbl@savelocaleid\relax
2683 \fi}

```

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

```

2684 \def\bbl@provide@new#1{%

```

```

2685 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2686 \@namedef{extras#1}{}%
2687 \@namedef{noextras#1}{}%
2688 \bbl@startcommands*{#1}{captions}%
2689 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2690 \def\bbl@tempb##1{% elt for \bbl@captionslist
2691 \ifx##1\@empty\else
2692 \bbl@exp{%
2693 \\\SetString\\##1{%
2694 \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2695 \expandafter\bbl@tempb
2696 \fi}%
2697 \expandafter\bbl@tempb\bbl@captionslist\@empty
2698 \else
2699 \ifx\bbl@initoload\relax
2700 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2701 \else
2702 \bbl@read@ini{\bbl@initoload}2% % Same
2703 \fi
2704 \fi
2705 \StartBabelCommands*{#1}{date}%
2706 \ifx\bbl@KVP@import\@nnil
2707 \bbl@exp{%
2708 \\\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2709 \else
2710 \bbl@savetoday
2711 \bbl@savestate
2712 \fi
2713 \bbl@endcommands
2714 \bbl@load@basic{#1}%
2715 % == hyphenmins == (only if new)
2716 \bbl@exp{%
2717 \gdef<#1hyphenmins>{%
2718 {\bbl@ifunset{\bbl@lfthm#1}{2}{\bbl@cs{lfthm#1}}}%
2719 {\bbl@ifunset{\bbl@rgthm#1}{3}{\bbl@cs{rgthm#1}}}}}%
2720 % == hyphenrules (also in renew) ==
2721 \bbl@provide@hyphens{#1}%
2722 \ifx\bbl@KVP@main\@nnil\else
2723 \expandafter\main@language\expandafter{#1}%
2724 \fi}
2725 %
2726 \def\bbl@provide@renew#1{%
2727 \ifx\bbl@KVP@captions\@nnil\else
2728 \StartBabelCommands*{#1}{captions}%
2729 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2730 \EndBabelCommands
2731 \fi
2732 \ifx\bbl@KVP@import\@nnil\else
2733 \StartBabelCommands*{#1}{date}%
2734 \bbl@savetoday
2735 \bbl@savestate
2736 \EndBabelCommands
2737 \fi
2738 % == hyphenrules (also in new) ==
2739 \ifx\bbl@lbkflag\@empty
2740 \bbl@provide@hyphens{#1}%
2741 \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.)

```

2742 \def\bbl@load@basic#1{%
2743 \ifcase\bbl@howloaded\or\or

```



```

2744 \ifcase\csname bbl@llevel@\language\endcsname
2745 \bbl@csarg\let\lname@\language\relax
2746 \fi
2747 \fi
2748 \bbl@ifunset{bbl@lname@#1}%
2749 {\def\BabelBeforeIni##1##2{%
2750 \begingroup
2751 \let\bbl@ini@captions@aux@gobbletwo
2752 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2753 \bbl@read@ini{##1}1%
2754 \ifx\bbl@initoload\relax\endinput\fi
2755 \endgroup}%
2756 \begingroup % boxed, to avoid extra spaces:
2757 \ifx\bbl@initoload\relax
2758 \bbl@input@texini{#1}%
2759 \else
2760 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2761 \fi
2762 \endgroup}%
2763 {}

```

The hyphenrules option is handled with an auxiliary macro.

```

2764 \def\bbl@provide@hyphens#1{%
2765 \let\bbl@tempa\relax
2766 \ifx\bbl@KVP@hyphenrules\@nnil\else
2767 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2768 \bbl@foreach\bbl@KVP@hyphenrules{%
2769 \ifx\bbl@tempa\relax % if not yet found
2770 \bbl@ifsamestring{##1}{+}%
2771 {\bbl@exp{\addlanguage\<l@##1>}}}%
2772 }%
2773 \bbl@ifunset{l@##1}%
2774 {}%
2775 {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
2776 \fi}%
2777 \fi
2778 \ifx\bbl@tempa\relax % if no opt or no language in opt found
2779 \ifx\bbl@KVP@import\@nnil
2780 \ifx\bbl@initoload\relax\else
2781 \bbl@exp{% and hyphenrules is not empty
2782 \bbl@ifblank{\bbl@cs{hyphr@#1}}%
2783 }%
2784 {\let\bbl@tempa\<l@bbl@cl{hyphr}>}}%
2785 \fi
2786 \else % if importing
2787 \bbl@exp{% and hyphenrules is not empty
2788 \bbl@ifblank{\bbl@cs{hyphr@#1}}%
2789 }%
2790 {\let\bbl@tempa\<l@bbl@cl{hyphr}>}}%
2791 \fi
2792 \fi
2793 \bbl@ifunset{bbl@tempa}% ie, relax or undefined
2794 {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
2795 {\bbl@exp{\adddialect\<l@#1>\language}}%
2796 }% so, l@<lang> is ok - nothing to do
2797 {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}% found in opt list or ini

```

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

```

2798 \def\bbl@input@texini#1{%
2799 \bbl@bsphack
2800 \bbl@exp{%
2801 \catcode\%%=14 \catcode\==0
2802 \catcode\{=1 \catcode\}=2
2803 \lowercase{\InputIfFileExists{babel-#1.tex}{}}%

```

```

2804 \catcode`\%%=\the\catcode`\% \relax
2805 \catcode`\%%=\the\catcode`\% \relax
2806 \catcode`\%=\the\catcode`\% \relax
2807 \catcode`\%=\the\catcode`\% \relax}%
2808 \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.

```

2809 \def\bbl@inline#1\bbl@inline{%
2810 \ifnextchar[\bbl@inisect{\ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2811 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2812 \def\bbl@iniskip#1\@@{% if starts with ;
2813 \def\bbl@inistore#1=#2\@@{% full (default)
2814 \bbl@trim@def\bbl@tempa{#1}%
2815 \bbl@trim\toks@{#2}%
2816 \bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2817 \ifin@ \else
2818 \bbl@exp{%
2819 \\\g@addto@macro\\bbl@inidata{%
2820 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2821 \fi}
2822 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2823 \bbl@trim@def\bbl@tempa{#1}%
2824 \bbl@trim\toks@{#2}%
2825 \bbl@xin@{.identification.}{.\bbl@section.}%
2826 \ifin@
2827 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2828 \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2829 \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.

```

2830 \ifx\bbl@readstream\undefined
2831 \csname newread\endcsname\bbl@readstream
2832 \fi
2833 \def\bbl@read@ini#1#2{%
2834 \global\let\bbl@extend@ini@gobble
2835 \openin\bbl@readstream=babel-#1.ini
2836 \ifeof\bbl@readstream
2837 \bbl@error
2838 {There is no ini file for the requested language\\%
2839 (#1: \language). Perhaps you misspelled it or your\\%
2840 installation is not complete.}%
2841 {Fix the name or reinstall babel.}%
2842 \else
2843 % == Store ini data in \bbl@inidata ==
2844 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2845 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2846 \bbl@info{Importing
2847 \ifcase#2font and identification \or basic \fi
2848 data for \language\\%
2849 from babel-#1.ini. Reported}%
2850 \ifnum#2=\z@
2851 \global\let\bbl@inidata@empty
2852 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2853 \fi
2854 \def\bbl@section{identification}%
2855 \bbl@exp{\\bbl@inistore tag.ini=#1\\}%
2856 \bbl@inistore load.level=#2\@@

```

```

2857 \loop
2858 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2859 \endlinechar\m@ne
2860 \read\bbl@readstream to \bbl@line
2861 \endlinechar`\^^M
2862 \ifx\bbl@line\@empty\else
2863 \expandafter\bbl@inline\bbl@line\bbl@inline
2864 \fi
2865 \repeat
2866 % == Process stored data ==
2867 \bbl@csarg\xdef{lini@\language}\bbl@ini%
2868 \bbl@read@ini@aux
2869 % == 'Export' data ==
2870 \bbl@ini@exports{#2}%
2871 \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2872 \global\let\bbl@inidata\@empty
2873 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
2874 \bbl@toglobal\bbl@ini@loaded
2875 \fi}
2876 \def\bbl@read@ini@aux{%
2877 \let\bbl@savestrings\@empty
2878 \let\bbl@savetoday\@empty
2879 \let\bbl@savestate\@empty
2880 \def\bbl@elt##1##2##3{%
2881 \def\bbl@section{##1}%
2882 \in@{=date.}{=##1}% Find a better place
2883 \ifin@
2884 \bbl@ifunset{bbl@inikv@##1}%
2885 {\bbl@ini@calendar{##1}}%
2886 {}%
2887 \fi
2888 \in@{=identification/extension.}{=##1/##2}%
2889 \ifin@
2890 \bbl@ini@extension{##2}%
2891 \fi
2892 \bbl@ifunset{bbl@inikv@##1}{}%
2893 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2894 \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.

```

2895 \def\bbl@extend@ini@aux#1{%
2896 \bbl@startcommands*{#1}{captions}%
2897 % Activate captions/... and modify exports
2898 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2899 \setlocalecaption{#1}{##1}{##2}}%
2900 \def\bbl@inikv@captions##1##2{%
2901 \bbl@ini@captions@aux{##1}{##2}}%
2902 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2903 \def\bbl@exportkey##1##2##3{%
2904 \bbl@ifunset{bbl@kv@##2}{}%
2905 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2906 \bbl@exp{\global\let\<bbl@##1@\language>\<bbl@kv@##2>}%
2907 \fi}}%
2908 % As with \bbl@read@ini, but with some changes
2909 \bbl@read@ini@aux
2910 \bbl@ini@exports\tw@
2911 % Update inidata@lang by pretending the ini is read.
2912 \def\bbl@elt##1##2##3{%
2913 \def\bbl@section{##1}%
2914 \bbl@inline##2=##3\bbl@inline}%
2915 \csname bbl@inidata@#1\endcsname
2916 \global\bbl@csarg\let{inidata@#1}\bbl@inidata

```

```

2917 \StartBabelCommands*{#1}{date}% And from the import stuff
2918 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2919 \bbl@savetoday
2920 \bbl@savedate
2921 \bbl@endcommands}

```

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

```

2922 \def\bbl@ini@calendar#1{%
2923 \lowercase{\def\bbl@tempa{=#1=}}%
2924 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2925 \bbl@replace\bbl@tempa{=date.}{}%
2926 \in@{.licr=}{#1=}%
2927 \ifin@
2928 \ifcase\bbl@engine
2929 \bbl@replace\bbl@tempa{.licr=}{}%
2930 \else
2931 \let\bbl@tempa\relax
2932 \fi
2933 \fi
2934 \ifx\bbl@tempa\relax\else
2935 \bbl@replace\bbl@tempa{=}{}%
2936 \ifx\bbl@tempa\@empty\else
2937 \xdef\bbl@calendars{\bbl@tempa}%
2938 \fi
2939 \bbl@exp{%
2940 \def<\bbl@inikv@#1>####1####2{%
2941 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2942 \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).

```

2943 \def\bbl@renewinikey#1/#2\@#3{%
2944 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2945 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2946 \bbl@trim\toks@{#3}% value
2947 \bbl@exp{%
2948 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2949 \\\g@addto@macro\\bbl@inidata{%
2950 \\\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.

```

2951 \def\bbl@exportkey#1#2#3{%
2952 \bbl@ifunset{\bbl@kv@#2}%
2953 {\bbl@csarg\gdef{#1@\language}\bbl@tempa{#3}}%
2954 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
2955 \bbl@csarg\gdef{#1@\language}\bbl@tempa{#3}}%
2956 \else
2957 \bbl@exp{\global\let<\bbl@#1@\language>\<\bbl@kv@#2>}%
2958 \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.

```

2959 \def\bbl@iniwarning#1{%
2960 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2961 {\bbl@warning{%
2962 From babel-\bbl@cs{lini@\language}.ini:\%
2963 \bbl@cs{@kv@identification.warning#1}\%
2964 Reported }}}
2965 %
2966 \let\bbl@release@transforms\@empty

```

BCP 47 extensions are separated by a single letter (eg, latin-x-medieval. The following macro handles this special case to create correctly the correspondig info.

```

2967 \def\bbl@ini@extension#1{%
2968   \def\bbl@tempa{#1}%
2969   \bbl@replace\bbl@tempa{extension.}{}%
2970   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2971   \bbl@ifunset\bbl@info@#1{%
2972     {\bbl@csarg\xdef\info@#1}{ext/\bbl@tempa}%
2973     \bbl@exp{%
2974       \\g@addto@macro\\bbl@moreinfo{%
2975         \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
2976     {}%
2977 \let\bbl@moreinfo\@empty
2978 %
2979 \def\bbl@ini@exports#1{%
2980   % Identification always exported
2981   \bbl@iniwarning{}%
2982   \ifcase\bbl@engine
2983     \bbl@iniwarning{.pdflatex}%
2984   \or
2985     \bbl@iniwarning{.lualatex}%
2986   \or
2987     \bbl@iniwarning{.xelatex}%
2988   \fi%
2989   \bbl@exportkey{llevel}{identification.load.level}{}%
2990   \bbl@exportkey{elname}{identification.name.english}{}%
2991   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2992     {\csname bbl@elname@\language\endcsname}}%
2993   \bbl@exportkey{tbcpr}{identification.tag.bcp47}{}%
2994   \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
2995   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2996   \bbl@exportkey{esname}{identification.script.name}{}%
2997   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2998     {\csname bbl@esname@\language\endcsname}}%
2999   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3000   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3001   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3002   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
3003   \bbl@moreinfo
3004   % Also maps bcp47 -> language
3005   \ifbbl@bcptoname
3006     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcpr}}{\language}%
3007   \fi
3008   % Conditional
3009   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
3010     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3011     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3012     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3013     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3014     \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3015     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3016     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3017     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3018     \bbl@exportkey{intsp}{typography.intraspace}{}%
3019     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3020     \bbl@exportkey{chrng}{characters.ranges}{}%
3021     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3022     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3023   \ifnum#1=\tw@         % only (re)new
3024     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3025     \bbl@toglobal\bbl@savetoday
3026     \bbl@toglobal\bbl@savestate
3027     \bbl@savestrings

```

```

3028 \fi
3029 \fi}

```

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

```

3030 \def\bbl@inikv#1#2{%      key=value
3031 \toks@{#2}%              This hides #'s from ini values
3032 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3033 \let\bbl@inikv@identification\bbl@inikv
3034 \let\bbl@inikv@date\bbl@inikv
3035 \let\bbl@inikv@typography\bbl@inikv
3036 \let\bbl@inikv@characters\bbl@inikv
3037 \let\bbl@inikv@numbers\bbl@inikv

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

3038 \def\bbl@inikv@counters#1#2{%
3039 \bbl@ifsamestring{#1}{digits}%
3040 {\bbl@error{The counter name 'digits' is reserved for mapping\%
3041 decimal digits}%
3042 {Use another name.}}%
3043 }%
3044 \def\bbl@tempc{#1}%
3045 \bbl@trim@def{\bbl@tempb*}{#2}%
3046 \in@{.1$}{#1$}%
3047 \ifin@
3048 \bbl@replace\bbl@tempc{.1}{}%
3049 \bbl@csarg\protected@xdef{cnt@#1@\language@}{%
3050 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3051 \fi
3052 \in@{.F.}{#1}%
3053 \ifin@\else\in@{.S.}{#1}\fi
3054 \ifin@
3055 \bbl@csarg\protected@xdef{cnt@#1@\language@}{\bbl@tempb*}%
3056 \else
3057 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3058 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3059 \bbl@csarg{\global\expandafter\let}{cnt@#1@\language@}\bbl@tempa
3060 \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.

```

3061 \ifcase\bbl@engine
3062 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3063 \bbl@ini@captions@aux{#1}{#2}}
3064 \else
3065 \def\bbl@inikv@captions#1#2{%
3066 \bbl@ini@captions@aux{#1}{#2}}
3067 \fi

```

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

```

3068 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3069 \bbl@replace\bbl@tempa{.template}{}%
3070 \def\bbl@toreplace{#1}{}%
3071 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3072 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3073 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3074 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3075 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3076 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3077 \ifin@

```

```

3078 \@nameuse{bbl@patch\bbl@tempa}%
3079 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3080 \fi
3081 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3082 \ifin@
3083 \toks@\expandafter{\bbl@toreplace}%
3084 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3085 \fi}
3086 \def\bbl@ini@captions@aux#1#2{%
3087 \bbl@trim@def\bbl@tempa{#1}%
3088 \bbl@xin@{.template}{\bbl@tempa}%
3089 \ifin@
3090 \bbl@ini@captions@template{#2}\language\language
3091 \else
3092 \bbl@ifblank{#2}%
3093 {\bbl@exp{%
3094 \toks@{\bbl@nocaption{\bbl@tempa}\language\bbl@tempa name}}}%
3095 {\bbl@trim\toks@{#2}}%
3096 \bbl@exp{%
3097 \bbl@add\bbl@savestrings{%
3098 \SetString\<\bbl@tempa name>{\the\toks@}}%
3099 \toks@\expandafter{\bbl@captionslist}%
3100 \bbl@exp{\in@{\<\bbl@tempa name>}\the\toks@}}%
3101 \ifin@else
3102 \bbl@exp{%
3103 \bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3104 \bbl@to\global\<\bbl@extracaps@\language>}%
3105 \fi
3106 \fi}

```

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

```

3107 \def\bbl@list@the{%
3108 part,chapter,section,subsection,subsubsection,paragraph,%
3109 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3110 table,page,footnote,mpfootnote,mpfn}
3111 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3112 \bbl@ifunset{bbl@map@#1@\language}%
3113 {\@nameuse{#1}}%
3114 {\@nameuse{bbl@map@#1@\language}}%
3115 \def\bbl@inikv@labels#1#2{%
3116 \in@{.map}{#1}%
3117 \ifin@
3118 \ifx\bbl@KVP@labels\@nnil\else
3119 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3120 \ifin@
3121 \def\bbl@tempc{#1}%
3122 \bbl@replace\bbl@tempc{.map}{}%
3123 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3124 \bbl@exp{%
3125 \gdef\<bbl@map@\bbl@tempc @\language>%
3126 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3127 \bbl@foreach\bbl@list@the{%
3128 \bbl@ifunset{the##1}{}%
3129 {\bbl@exp{\let\bbl@tempd\<the##1>}%
3130 \bbl@exp{%
3131 \bbl@sreplace\<the##1>%
3132 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3133 \bbl@sreplace\<the##1>%
3134 {\<\@empty @\bbl@tempc>\<c@##1>}\bbl@map@cnt{\bbl@tempc}{##1}}}%
3135 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3136 \toks@\expandafter\expandafter\expandafter{%
3137 \csname the##1\endcsname}%
3138 \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%

```

```

3139         \fi}}%
3140     \fi
3141     \fi
3142 %
3143 \else
3144 %
3145 % The following code is still under study. You can test it and make
3146 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3147 % language dependent.
3148 \in@{enumerate.}{#1}%
3149 \ifin@
3150     \def\bbl@tempa{#1}%
3151     \bbl@replace\bbl@tempa{enumerate.}{}%
3152     \def\bbl@toreplace{#2}%
3153     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3154     \bbl@replace\bbl@toreplace{[]}{\csname the}%
3155     \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3156     \toks@{\expandafter{\bbl@toreplace}%
3157 % TODO. Execute only once:
3158     \bbl@exp{%
3159         \\bbl@add\<extras\language>{%
3160             \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3161             \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3162             \\bbl@tglobal\<extras\language>%
3163     \fi
3164     \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.

```

3165 \def\bbl@chapttype{chapter}
3166 \ifx\@makechapterhead\@undefined
3167     \let\bbl@patchchapter\relax
3168 \else\ifx\thechapter\@undefined
3169     \let\bbl@patchchapter\relax
3170 \else\ifx\ps@headings\@undefined
3171     \let\bbl@patchchapter\relax
3172 \else
3173     \def\bbl@patchchapter{%
3174         \global\let\bbl@patchchapter\relax
3175         \gdef\bbl@chfmt{%
3176             \bbl@ifunset{\bbl@\bbl@chapttype fmt@\language}%
3177             {\@chapapp\space\thechapter}
3178             {\@nameuse{\bbl@\bbl@chapttype fmt@\language}}}
3179         \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3180         \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3181         \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3182         \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3183         \bbl@tglobal\appendix
3184         \bbl@tglobal\ps@headings
3185         \bbl@tglobal\chaptermark
3186         \bbl@tglobal\@makechapterhead}
3187     \let\bbl@patchappendix\bbl@patchchapter
3188 \fi\fi\fi
3189 \ifx\@part\@undefined
3190     \let\bbl@patchpart\relax
3191 \else
3192     \def\bbl@patchpart{%
3193         \global\let\bbl@patchpart\relax
3194         \gdef\bbl@partformat{%
3195             \bbl@ifunset{\bbl@partfmt@\language}%
3196             {\partname\nobreakspace\thepart}

```



```

3197         {\@nameuse{bbl@partfmt@\language name}}
3198     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3199     \bbl@tglobal\@part}
3200 \fi

Date. Arguments (year, month, day) are not protected, on purpose. In \today, arguments are always
gregorian, and therefore always converted with other calendars. TODO. Document

3201 % Arguments are _not_ protected.
3202 \let\bbl@calendar\@empty
3203 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3204 \def\bbl@localedate#1#2#3#4{%
3205     \begin{group}
3206         \edef\bbl@they{#2}%
3207         \edef\bbl@them{#3}%
3208         \edef\bbl@thed{#4}%
3209         \edef\bbl@tempe{%
3210             \bbl@ifunset{bbl@calpr@\language name}{\bbl@cl{calpr}},%
3211             #1}%
3212         \bbl@replace\bbl@tempe{ }{}%
3213         \bbl@replace\bbl@tempe{convert}{convert=}%
3214         \let\bbl@ld@calendar\@empty
3215         \let\bbl@ld@variant\@empty
3216         \let\bbl@ld@convert\relax
3217         \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld##1}{##2}}%
3218         \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3219         \bbl@replace\bbl@ld@calendar{gregorian}{}%
3220         \ifx\bbl@ld@calendar\@empty\else
3221             \ifx\bbl@ld@convert\relax\else
3222                 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3223                 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3224             \fi
3225         \fi
3226         \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3227         \edef\bbl@calendar{% Used in \month..., too
3228             \bbl@ld@calendar
3229             \ifx\bbl@ld@variant\@empty\else
3230                 .\bbl@ld@variant
3231             \fi}%
3232         \bbl@cased
3233         {\@nameuse{bbl@date@\language name @\bbl@calendar}%
3234          \bbl@they\bbl@them\bbl@thed}%
3235     \end{group}
3236 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3237 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3238     \bbl@trim@def\bbl@tempa{#1.#2}%
3239     \bbl@ifsamestring{\bbl@tempa}{months.wide}%           to savedate
3240     {\bbl@trim@def\bbl@tempa{#3}%
3241      \bbl@trim\toks@{#5}%
3242      \@temptokena\expandafter{\bbl@savestate}%
3243      \bbl@exp{% Reverse order - in ini last wins
3244          \def\\bbl@savestate{%
3245              \\SetString<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3246              \the\@temptokena}}%
3247      {\bbl@ifsamestring{\bbl@tempa}{date.long}%           defined now
3248       {\lowercase{\def\bbl@tempb{#6}}%
3249        \bbl@trim@def\bbl@toreplace{#5}%
3250        \bbl@TG@@date
3251        \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3252        \ifx\bbl@savetoday\@empty
3253            \bbl@exp{% TODO. Move to a better place.
3254                \\AfterBabelCommands{%
3255                    \def<\language name date>{\\protect<\language name date >}%
3256                    \\newcommand<\language name date >[4][\%

```

```

3257         \bbl@usedategroupttrue
3258         \<bbl@ensure@language>{%
3259             \\\localedate[####1]{####2}{####3}{####4}}}%
3260         \def\bbl@savetoday{%
3261             \\\SetString\\today{%
3262                 \<language date>[convert]%
3263                 {\the\year}{\the\month}{\the\day}}}%
3264         \fi}%
3265     }}}}

```

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).

```

3266 \let\bbl@calendar\@empty
3267 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3268     \@nameuse{bbl@ca#2}#1\@}
3269 \newcommand\babelDateSpace{\nobreakspace}
3270 \newcommand\babelDateDot{. \@ % TODO. \let instead of repeating
3271 \newcommand\babelDated[1]{\number#1}
3272 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3273 \newcommand\babelDateM[1]{\number#1}
3274 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3275 \newcommand\babelDateMMMM[1]{%
3276     \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3277 \newcommand\babelDatey[1]{\number#1}%
3278 \newcommand\babelDateyy[1]{%
3279     \ifnum#1<10 0\number#1 %
3280     \else\ifnum#1<100 \number#1 %
3281     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3282     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3283     \else
3284         \bbl@error
3285         {Currently two-digit years are restricted to the\
3286         range 0-9999.}%
3287         {There is little you can do. Sorry.}%
3288     \fi\fi\fi\fi}
3289 \newcommand\babelDateyyyy[1]{\number#1} % TODO - add leading 0
3290 \def\bbl@replace@finish@iii#1{%
3291     \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3292 \def\bbl@TG@date{%
3293     \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3294     \bbl@replace\bbl@toreplace{[. ]}{\babelDateDot}}%
3295     \bbl@replace\bbl@toreplace{[d]}{\babelDated{####3}}%
3296     \bbl@replace\bbl@toreplace{[dd]}{\babelDatedd{####3}}%
3297     \bbl@replace\bbl@toreplace{[M]}{\babelDateM{####2}}%
3298     \bbl@replace\bbl@toreplace{[MM]}{\babelDateMM{####2}}%
3299     \bbl@replace\bbl@toreplace{[MMMM]}{\babelDateMMMM{####2}}%
3300     \bbl@replace\bbl@toreplace{[y]}{\babelDatey{####1}}%
3301     \bbl@replace\bbl@toreplace{[yy]}{\babelDateyy{####1}}%
3302     \bbl@replace\bbl@toreplace{[yyyy]}{\babelDateyyyy{####1}}%
3303     \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3304     \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3305     \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3306     \bbl@replace@finish@iii\bbl@toreplace}
3307 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3308 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3309 \let\bbl@release@transforms\@empty
3310 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3311     \bbl@transforms\babelprehyphenation}
3312 \@namedef{bbl@inikv@transforms.posthyphenation}{%

```

```

3313 \bbl@transforms\babelposthyphenation}
3314 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3315   #1[#2]{#3}{#4}{#5}}
3316 \begingroup % A hack. TODO. Don't require an specific order
3317 \catcode\%=12
3318 \catcode\&=14
3319 \gdef\bbl@transforms#1#2#3{%&
3320   \ifx\bbl@KVP@transforms\@nnil\else
3321     \directlua{
3322       local str = [=[#2]=]
3323       str = str:gsub('%.%d+%.%d+$', '')
3324       tex.print([[def\string\babeltempa{]} .. str .. [[]]])
3325     }&
3326     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3327     \ifin@
3328       \in@{.0$}{#2$}&
3329     \ifin@
3330       \directlua{
3331         local str = string.match([[bbl@KVP@transforms]],
3332           '%(([^%(-)%][^%)]-\babeltempa')
3333         if str == nil then
3334           tex.print([[def\string\babeltempb{]}])
3335         else
3336           tex.print([[def\string\babeltempb{,attribute=]} .. str .. [[]]])
3337         end
3338       }
3339       \toks@{#3}&
3340       \bbl@exp{%&
3341         \\g@addto@macro\\bbl@release@transforms{%&
3342           \relax & Closes previous \bbl@transforms@aux
3343           \\bbl@transforms@aux
3344           \\#1{label=\babeltempa\babeltempb}{\thelanguage}\the\toks@}}&
3345       \else
3346         \g@addto@macro\bbl@release@transforms{, {#3}}&
3347       \fi
3348     \fi
3349   \fi}
3350 \endgroup

```

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

```

3351 \def\bbl@provide@lsys#1{%
3352   \bbl@ifunset{bbl@lname@#1}%
3353     {\bbl@load@info{#1}}%
3354   {}%
3355   \bbl@csarg\let{lsys@#1}\@empty
3356   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3357   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3358   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3359   \bbl@ifunset{bbl@lname@#1}{}%
3360     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3361   \ifcase\bbl@engine\or\or
3362     \bbl@ifunset{bbl@prehc@#1}{}%
3363     {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3364     }%
3365     {\ifx\bbl@xenohyph\undefined
3366       \global\let\bbl@xenohyph\bbl@xenohyph@d
3367       \ifx\AtBeginDocument\@notprerr
3368         \expandafter\@secondoftwo % to execute right now
3369       \fi
3370       \AtBeginDocument{%
3371         \bbl@patchfont{\bbl@xenohyph}%
3372         \expandafter\selectlanguage\expandafter{\thelanguage}}%

```

```

3373     \fi}}%
3374 \fi
3375 \bbl@csarg\bbl@tglobal{lsys@#1}}
3376 \def\bbl@xenoHyph@d{%
3377   \bbl@ifset{bbl@prehc{\languageName}%
3378     {\ifnum\hyphenchar\font=\defaultHyphenchar
3379       \iffontchar\font\bbl@c1{prehc}\relax
3380       \hyphenchar\font\bbl@c1{prehc}\relax
3381     \else\iffontchar\font"200B
3382       \hyphenchar\font"200B
3383     \else
3384       \bbl@warning
3385       {Neither 0 nor ZERO WIDTH SPACE are available\\%
3386        in the current font, and therefore the hyphen\\%
3387        will be printed. Try changing the fontspec's\\%
3388        'HyphenChar' to another value, but be aware\\%
3389        this setting is not safe (see the manual)}%
3390       \hyphenchar\font\defaultHyphenchar
3391     \fi\fi
3392   \fi}%
3393   {\hyphenchar\font\defaultHyphenchar}}
3394 % \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).

```

3395 \def\bbl@load@info#1{%
3396   \def\BabelBeforeIni##1##2{%
3397     \begingroup
3398     \bbl@read@ini{##1}0%
3399     \endinput          % babel- .tex may contain onlypreamble's
3400     \endgroup}%       boxed, to avoid extra spaces:
3401     {\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.

```

3402 \def\bbl@setdigits#1#2#3#4#5{%
3403   \bbl@exp{%
3404     \def\<\languageName digits>####1{%      ie, \langdigits
3405       \<bbl@digits@\languageName>####1\\\@nil}%
3406       \let\<bbl@cntr@digits@\languageName>\<\languageName digits>%
3407       \def\<\languageName counter>####1{%    ie, \langcounter
3408         \\\expandafter\<bbl@counter@\languageName>%
3409         \\\csname c@####1\endcsname}%
3410       \def\<bbl@counter@\languageName>####1{% ie, \bbl@counter@lang
3411         \\\expandafter\<bbl@digits@\languageName>%
3412         \\\number####1\\\@nil}}}%
3413   \def\bbl@tempa##1##2##3##4##5{%
3414     \bbl@exp{%      Wow, quite a lot of hashes! :-(
3415       \def\<bbl@digits@\languageName>#####1{%
3416         \\\ifx#####1\\\@nil          % ie, \bbl@digits@lang
3417         \\\else
3418           \\\ifx0#####1#1%
3419           \\\else\\\ifx1#####1#2%
3420           \\\else\\\ifx2#####1#3%
3421           \\\else\\\ifx3#####1#4%
3422           \\\else\\\ifx4#####1#5%
3423           \\\else\\\ifx5#####1#1%
3424           \\\else\\\ifx6#####1#2%
3425           \\\else\\\ifx7#####1#3%
3426           \\\else\\\ifx8#####1#4%
3427           \\\else\\\ifx9#####1#5%

```

[illegible]

```

3433 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3434   \ifx\\#1%                % \\ before, in case #1 is multiletter
3435     \bbl@exp{%
3436       \def\\ \bbl@tempa####1{%
3437         \<ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3438   \else
3439     \toks@\expandafter{\the\toks@\or #1}%
3440     \expandafter\bbl@buildifcase
3441   \fi}

```

```

3442 \newcommand\localenatural[2]{\bbl@cs{cntr@#1\language}\#2}}
3443 \def\bbl@localecntr#1#2{\localenatural{#2}{#1}}
3444 \newcommand\localecounter[2]{%
3445   \expandafter\bbl@localecntr
3446   \expandafter{\number\csname c@#2\endcsname}{#1}}
3447 \def\bbl@alphnumeral#1#2{%
3448   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3449 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3450   \ifcase\car#8\nil\or    % Currently >10000, but prepared for bigger
3451     \bbl@alphnumeral@ii{#9}000000#1\or
3452     \bbl@alphnumeral@ii{#9}00000#1#2\or
3453     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3454     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3455     \bbl@alphnum@invalid{>9999}%
3456   \fi}
3457 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3458   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3459     {\bbl@cs{cntr@#1.4@\language}#5%
3460      \bbl@cs{cntr@#1.3@\language}#6%
3461      \bbl@cs{cntr@#1.2@\language}#7%
3462      \bbl@cs{cntr@#1.1@\language}#8%
3463      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3464        \bbl@ifunset{\bbl@cntr@#1.S.321@\language}{}%
3465        {\bbl@cs{cntr@#1.S.321@\language}}%
3466      \fi}%
3467   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}}
3468 \def\bbl@alphnum@invalid#1{%
3469   \bbl@error{Alphabetic numeral too large (#1)}%
3470   {Currently this is the limit.}}

```

```

3471 \def\bbl@localeinfo#1#2{%
3472   \bbl@ifunset\bbl@info@#2}{#1}%
3473   {\bbl@ifunset\bbl@\csname bbl@info@#2\endcsname @\language\name}{#1}%
3474   {\bbl@cs{\csname bbl@info@#2\endcsname @\language\name}}}%
3475 \newcommand\localeinfo[1]{%
3476   \ifx#1@empty    % TODO. A bit hackish to make it expandable.
3477     \bbl@afterelse\bbl@localeinfo}%
3478   \else
3479     \bbl@localeinfo
3480     {\bbl@error{I've found no info for the current locale.\\%

```

```

3481             The corresponding ini file has not been loaded\\%
3482             Perhaps it doesn't exist}%
3483             {See the manual for details.}}%
3484         {#1}%
3485     \fi}
3486 % \@namedef{bbl@info@name.locale}{lcname}
3487 \@namedef{bbl@info@tag.ini}{lini}
3488 \@namedef{bbl@info@name.english}{elname}
3489 \@namedef{bbl@info@name.opentype}{lname}
3490 \@namedef{bbl@info@tag.bcp47}{tbc}
3491 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3492 \@namedef{bbl@info@tag.opentype}{lotf}
3493 \@namedef{bbl@info@script.name}{esname}
3494 \@namedef{bbl@info@script.name.opentype}{sname}
3495 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3496 \@namedef{bbl@info@script.tag.opentype}{sotf}
3497 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3498 \@namedef{bbl@info@variant.tag.bcp47}{vbc}
3499 % Extensions are dealt with in a special way
3500 % Now, an internal \LaTeX{} macro:
3501 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}

```

With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it.

```

3502 <<(*More package options)>> ≡
3503 \DeclareOption{ensureinfo=off}{}
3504 <</More package options>>
3505 %
3506 \let\bbl@ensureinfo\@gobble
3507 \newcommand\BabelEnsureInfo{%
3508   \ifx\InputIfFileExists\undefined\else
3509     \def\bbl@ensureinfo##1{%
3510       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3511   \fi
3512   \bbl@foreach\bbl@loaded{%
3513     \def\languageinfo{##1}%
3514     \bbl@ensureinfo{##1}}}%
3515 \@ifpackagewith{babel}{ensureinfo=off}{}%
3516 {\AtEndOfPackage{% Test for plain.
3517   \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

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.

```

3518 \newcommand\getLocaleproperty{%
3519   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3520 \def\bbl@getproperty@s#1#2#3{%
3521   \let#1\relax
3522   \def\bbl@elt##1##2##3{%
3523     \bbl@ifsamestring{##1/##2}{##3}%
3524     {\providecommand#1{##3}%
3525     \def\bbl@elt####1####2####3{}}}%
3526   {}}%
3527   \bbl@cs{inidata@#2}}%
3528 \def\bbl@getproperty@x#1#2#3{%
3529   \bbl@getproperty@s{#1}{#2}{#3}%
3530   \ifx#1\relax
3531     \bbl@error
3532     {Unknown key for locale '#2':\\%
3533     #3\\%
3534     \string#1 will be set to \relax}%
3535     {Perhaps you misspelled it.}%
3536   \fi}
3537 \let\bbl@ini@loaded\@empty
3538 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

8 Adjusting the Babel bahavior

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

```
3539 \newcommand\babeladjust[1]{% TODO. Error handling.
3540   \bbl@forkv{#1}{%
3541     \bbl@ifunset{bbl@ADJ@##1@##2}%
3542     {\bbl@cs{ADJ@##1}{##2}}%
3543     {\bbl@cs{ADJ@##1@##2}}}
3544 %
3545 \def\bbl@adjust@lua#1#2{%
3546   \ifvmode
3547     \ifnum\currentgrouplevel=\z@
3548       \directlua{ Babel.#2 }%
3549       \expandafter\expandafter\expandafter\@gobble
3550     \fi
3551   \fi
3552   {\bbl@error % The error is gobbled if everything went ok.
3553     {Currently, #1 related features can be adjusted only\\%
3554       in the main vertical list.}%
3555     {Maybe things change in the future, but this is what it is.}}}
3556 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3557   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3558 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3559   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3560 \@namedef{bbl@ADJ@bidi.text@on}{%
3561   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3562 \@namedef{bbl@ADJ@bidi.text@off}{%
3563   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3564 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3565   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3566 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3567   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3568 %
3569 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3570   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3571 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3572   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3573 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3574   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3575 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3576   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3577 \@namedef{bbl@ADJ@justify.arabic@on}{%
3578   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3579 \@namedef{bbl@ADJ@justify.arabic@off}{%
3580   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3581 %
3582 \def\bbl@adjust@layout#1{%
3583   \ifvmode
3584     #1%
3585     \expandafter\@gobble
3586   \fi
3587   {\bbl@error % The error is gobbled if everything went ok.
3588     {Currently, layout related features can be adjusted only\\%
3589       in vertical mode.}%
3590     {Maybe things change in the future, but this is what it is.}}}
3591 \@namedef{bbl@ADJ@layout.tabular@on}{%
3592   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3593 \@namedef{bbl@ADJ@layout.tabular@off}{%
3594   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3595 \@namedef{bbl@ADJ@layout.lists@on}{%
3596   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3597 \@namedef{bbl@ADJ@layout.lists@off}{%
3598   \bbl@adjust@layout{\let\list\bbl@OL@list}}
```

```

3599 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3600   \bbl@activateposthyphen}
3601 %
3602 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3603   \bbl@bcpallowedtrue}
3604 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3605   \bbl@bcpallowedfalse}
3606 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3607   \def\bbl@bcp@prefix{#1}}
3608 \def\bbl@bcp@prefix{bcp47-}
3609 \@namedef{bbl@ADJ@autoload.options}#1{%
3610   \def\bbl@autoload@options{#1}}
3611 \let\bbl@autoload@bcptoptions\@empty
3612 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3613   \def\bbl@autoload@bcptoptions{#1}}
3614 \newif\ifbbl@bcptoname
3615 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3616   \bbl@bcptonametrue}
3617   \BabelEnsureInfo}
3618 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3619   \bbl@bcptonamefalse}
3620 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3621   \directlua{ Babel.ignore_pre_char = function(node)
3622     return (node.lang == \the\csname l@nohyphenation\endcsname)
3623   end }}
3624 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3625   \directlua{ Babel.ignore_pre_char = function(node)
3626     return false
3627   end }}
3628 \@namedef{bbl@ADJ@select.write@shift}{%
3629   \let\bbl@restorelastskip\relax
3630   \def\bbl@savelastskip{%
3631     \let\bbl@restorelastskip\relax
3632     \ifvmode
3633       \ifdim\lastskip=\z@
3634         \let\bbl@restorelastskip\nobreak
3635       \else
3636         \bbl@exp{%
3637           \def\\bbl@restorelastskip{%
3638             \skip@=\the\lastskip
3639             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3640       \fi
3641     \fi}}
3642 \@namedef{bbl@ADJ@select.write@keep}{%
3643   \let\bbl@restorelastskip\relax
3644   \let\bbl@savelastskip\relax}
3645 \@namedef{bbl@ADJ@select.write@omit}{%
3646   \let\bbl@restorelastskip\relax
3647   \def\bbl@savelastskip##1\bbl@restorelastskip{}}

```

As the final task, load the code for lua. TODO: use babel name, override

```

3648 \ifx\directlua\@undefined\else
3649   \ifx\bbl@luapatterns\@undefined
3650     \input luababel.def
3651   \fi
3652 \fi

```

Continue with \LaTeX .

```

3653 </package | core>
3654 <*package>

```

8.1 Cross referencing macros

The \LaTeX book states:

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

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

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

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

```

3655 <{*More package options}> ≡
3656 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3657 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3658 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3659 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3660 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3661 <{/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.

```

3662 \bbl@trace{Cross referencing macros}
3663 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3664   \def\@newl@bel#1#2#3{%
3665     {\@safe@activestrue
3666       \bbl@ifunset{#1@#2}%
3667       \relax
3668       {\gdef\@multiplelabels{%
3669         \@latex@warning@no@line{There were multiply-defined labels}}%
3670         \@latex@warning@no@line{Label `#2' multiply defined}}%
3671       \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.

```

3672 \CheckCommand*\@testdef[3]{%
3673   \def\reserved@a{#3}%
3674   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3675   \else
3676     \@tempswatrue
3677   \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.

```

3678 \def\@testdef#1#2#3{% TODO. With @samestring?
3679   \@safe@activestrue
3680   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3681   \def\bbl@tempb{#3}%
3682   \@safe@activesfalse
3683   \ifx\bbl@tempa\relax
3684   \else
3685     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3686   \fi
3687   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3688   \ifx\bbl@tempa\bbl@tempb
3689   \else
3690     \@tempswatrue
3691   \fi}
3692 \fi

```

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

```

3693 \bbl@xin@{R}\bbl@opt@safe
3694 \ifin@
3695 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3696 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3697 {\expandafter\strip@prefix\meaning\ref}%
3698 \ifin@
3699 \bbl@redefine\@kernel@ref#1{%
3700 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activfalse}
3701 \bbl@redefine\@kernel@pageref#1{%
3702 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3703 \bbl@redefine\@kernel@sref#1{%
3704 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3705 \bbl@redefine\@kernel@spageref#1{%
3706 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3707 \else
3708 \bbl@redefinero bust\ref#1{%
3709 \@safe@activetrue\org@@ref{#1}\@safe@activfalse}
3710 \bbl@redefinero bust\pageref#1{%
3711 \@safe@activetrue\org@@pageref{#1}\@safe@activfalse}
3712 \fi
3713 \else
3714 \let\org@ref\ref
3715 \let\org@pageref\pageref
3716 \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.

```

3717 \bbl@xin@{B}\bbl@opt@safe
3718 \ifin@
3719 \bbl@redefine\@citex[#1]#2{%
3720 \@safe@activetrue\edef\@tempa{#2}\@safe@activfalse
3721 \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.

```

3722 \AtBeginDocument{%
3723 \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.)

```

3724 \def\@citex[#1][#2]#3{%
3725 \@safe@activetrue\edef\@tempa{#3}\@safe@activfalse
3726 \org@@citex[#1][#2]{\@tempa}}%
3727 }{}

```

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

```

3728 \AtBeginDocument{%
3729 \@ifpackageloaded{cite}{%
3730 \def\@citex[#1]#2{%
3731 \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3732 }{}

```

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

```

3733 \bbl@redefine\nocite#1{%
3734 \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

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

```
3735 \bbl@redefine\bibcite{%
3736 \bbl@cite@choice
3737 \bibcite}
```

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

```
3738 \def\bbl@bibcite#1#2{%
3739 \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3740 \def\bbl@cite@choice{%
3741 \global\let\bibcite\bbl@bibcite
3742 \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3743 \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3744 \global\let\bbl@cite@choice\relax}
```

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

```
3745 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3746 \bbl@redefine\@bibitem#1{%
3747 \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3748 \else
3749 \let\org@nocite\nocite
3750 \let\org@@citex\@citex
3751 \let\org@bibcite\bibcite
3752 \let\org@@bibitem\@bibitem
3753 \fi
```

8.2 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.

```
3754 \bbl@trace{Marks}
3755 \IfBabelLayout{sectioning}
3756 {\ifx\bbl@opt@headfoot\@nnil
3757 \g@addto@macro\@resetactivechars{%
3758 \set@typeset@protect
3759 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3760 \let\protect\noexpand
3761 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3762 \edef\thepage{%
3763 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3764 \fi}%
3765 \fi}
3766 {\ifbbl@single\else
3767 \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3768 \markright#1{%
3769 \bbl@ifblank{#1}%
```

```

3770      {\org@markright{}}}%
3771      {\toks@{#1}}%
3772      \bbl@exp{%
3773          \\org@markright{\\protect\\foreignlanguage{\language}%
3774              {\\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, \TeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3775      \ifx\@mkboth\markboth
3776      \def\bbl@tempc{\let\@mkboth\markboth}
3777      \else
3778      \def\bbl@tempc{}
3779      \fi
3780      \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3781      \markboth#1#2{%
3782          \protected@edef\bbl@tempb##1{%
3783              \protect\foreignlanguage
3784                  {\language}{\protect\bbl@restore@actives##1}}%
3785          \bbl@ifblank{#1}%
3786              {\toks@{}}%
3787              {\toks@\expandafter{\bbl@tempb{#1}}}%
3788          \bbl@ifblank{#2}%
3789              {\@temptokena{}}%
3790              {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3791          \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
3792          \bbl@tempc
3793      \fi} % end ifbbl@single, end \IfBabelLayout

```

8.3 Preventing clashes with other packages

8.3.1 `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.

```

3794 \bbl@trace{Preventing clashes with other packages}
3795 \ifx\org@ref\undefined\else
3796   \bbl@xin@{R}\bbl@opt@safe
3797   \ifin@
3798     \AtBeginDocument{%
3799       \@ifpackageloaded{ifthen}{%
3800         \bbl@redefine@long\ifthenelse#1#2#3{%
3801           \let\bbl@temp@pref\pageref
3802           \let\pageref\org@pageref
3803           \let\bbl@temp@ref\ref
3804           \let\ref\org@ref

```

```

3805         \@safe@activetrue
3806         \org@ifthenelse{#1}%
3807         {\let\pageref\bbl@temp@pref
3808          \let\ref\bbl@temp@ref
3809          \@safe@activetrue
3810          #2}%
3811         {\let\pageref\bbl@temp@pref
3812          \let\ref\bbl@temp@ref
3813          \@safe@activetrue
3814          #3}%
3815     }%
3816 }{}%
3817 }
3818 \fi

```

8.3.2 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`.

```

3819 \AtBeginDocument{%
3820   \@ifpackageloaded{varioref}{%
3821     \bbl@redefine\@@vpageref#1[#2]#3{%
3822       \@safe@activetrue
3823       \org@@@vpageref{#1}[#2]{#3}%
3824       \@safe@activetrue}%
3825     \bbl@redefine\vrefpagemum#1#2{%
3826       \@safe@activetrue
3827       \org@vrefpagemum{#1}{#2}%
3828       \@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.

```

3829   \expandafter\def\csname Ref \endcsname#1{%
3830     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3831   }{}%
3832 }
3833 \fi

```

8.3.3 hpline

`\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.

```

3834 \AtEndOfPackage{%
3835   \AtBeginDocument{%
3836     \@ifpackageloaded{hhline}%
3837     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3838      \else
3839        \makeatletter
3840        \def\@currname{hhline}\input{hhline.sty}\makeatother
3841      \fi}%
3842     {}}

```

`\substitutefontfamily` Deprecated. Use the tools provided by \TeX . 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.

```

3843 \def\substitutefontfamily#1#2#3{%
3844   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3845   \immediate\write15{%
3846     \string\ProvidesFile{#1#2.fd}%
3847     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3848     \space generated font description file]^{}
3849     \string\DeclareFontFamily{#1}{#2}{}}^{}
3850     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^{}
3851     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^{}
3852     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^{}
3853     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^{}
3854     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^{}
3855     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^{}
3856     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^{}
3857     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^{}
3858   }%
3859   \closeout15
3860 }
3861 \@onlypreamble\substitutefontfamily

```

8.4 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`

```

3862 \bbl@trace{Encoding and fonts}
3863 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,LHE,LWN,LMA,LMC,LMS,LMU}
3864 \newcommand\BabelNonText{TS1,T3,TS3}
3865 \let\org@TeX\TeX
3866 \let\org@LaTeX\LaTeX
3867 \let\ensureascii@firstofone
3868 \AtBeginDocument{%
3869   \def\@elt#1{, #1,}%
3870   \edef\bbl@tempa{\expandafter\@gobbletwo\fontenc@load@list}%
3871   \let\@elt\relax
3872   \let\bbl@tempb\@empty
3873   \def\bbl@tempc{OT1}%
3874   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3875     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3876   \bbl@foreach\bbl@tempa{%
3877     \bbl@xin@{#1}{\BabelNonASCII}%
3878     \ifin@
3879       \def\bbl@tempb{#1}% Store last non-ascii
3880     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3881       \ifin@\else
3882         \def\bbl@tempc{#1}% Store last ascii
3883       \fi
3884     \fi}%
3885   \ifx\bbl@tempb\@empty\else
3886     \bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
3887     \ifin@\else
3888       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3889     \fi
3890   \edef\ensureascii#1{%
3891     {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3892   \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3893   \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3894   \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.

```
3895 \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.

```
3896 \AtBeginDocument{%
3897   \ifpackageloaded{fontspec}%
3898     {\xdef\latinencoding{%
3899       \ifx\UTFencname\@undefined
3900         EU\ifcase\bbl@engine\or2\or1\fi
3901       \else
3902         \UTFencname
3903       \fi}}%
3904   {\gdef\latinencoding{OT1}}%
3905   \ifx\cf@encoding\bbl@t@one
3906     \xdef\latinencoding{\bbl@t@one}%
3907   \else
3908     \def\@elt#1{,#1,%
3909     \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3910     \let\@elt\relax
3911     \bbl@xin@{,T1,}\bbl@tempa
3912     \ifin@
3913       \xdef\latinencoding{\bbl@t@one}%
3914     \fi
3915   \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.

```
3916 \DeclareRobustCommand{\latintext}{%
3917   \fontencoding{\latinencoding}\selectfont
3918   \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.

```
3919 \ifx\@undefined\DeclareTextFontCommand
3920   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3921 \else
3922   \DeclareTextFontCommand{\textlatin}{\latintext}
3923 \fi
```

For several functions, we need to execute some code with `\selectfont`. With \TeX 2021-06-01, there is a hook for this purpose, but in older versions the \TeX command is patched (the latter solution will be eventually removed).

```
3924 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}
```

8.5 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).

- pdf_{tex} provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- x_{etex} 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 T_EX grouping.
- lua_{tex} 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 LuaT_EX-ja shows, vertical typesetting is possible, too.

```

3925 \bbl@trace{Loading basic (internal) bidi support}
3926 \ifodd\bbl@engine
3927 \else % TODO. Move to txtbabel
3928   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3929     \bbl@error
3930     {The bidi method 'basic' is available only in\\%
3931       luatex. I'll continue with 'bidi=default', so\\%
3932       expect wrong results}%
3933     {See the manual for further details.}%
3934   \let\bbl@beforeforeign\leavevmode
3935   \AtEndOfPackage{%
3936     \EnableBabelHook{babel-bidi}%
3937     \bbl@xebidipar}
3938 \fi\fi
3939 \def\bbl@loadxebidi#1{%
3940   \ifx\RTLfootnotetext\@undefined
3941     \AtEndOfPackage{%
3942       \EnableBabelHook{babel-bidi}%
3943       \ifx\fontspec\@undefined
3944         \bbl@loadfontspec % bidi needs fontspec
3945       \fi
3946       \usepackage#1{bidi}}%
3947   \fi}
3948 \ifnum\bbl@bidimode>200
3949   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3950     \bbl@tentative{bidi=bidi}
3951     \bbl@loadxebidi{}
3952   \or
3953     \bbl@loadxebidi{[rldocument]}
3954   \or
3955     \bbl@loadxebidi{}
3956   \fi
3957 \fi
3958 \fi
3959 % TODO? Separate:
3960 \ifnum\bbl@bidimode=\@ne
3961   \let\bbl@beforeforeign\leavevmode
3962   \ifodd\bbl@engine
3963     \newattribute\bbl@attr@dir
3964     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3965     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3966   \fi
3967   \AtEndOfPackage{%
3968     \EnableBabelHook{babel-bidi}%
3969     \ifodd\bbl@engine\else
3970       \bbl@xebidipar
3971     \fi}
3972 \fi

```

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

```

3973 \bbl@trace{Macros to switch the text direction}
3974 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}

```



```

3975 \def\bbl@rscripts{% TODO. Base on codes ??
3976   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3977   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
3978   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
3979   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3980   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3981   Old South Arabian,}%
3982 \def\bbl@provide@dirs#1{%
3983   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3984   \ifin@
3985     \global\bbl@csarg\chardef{wdir@#1}\@ne
3986     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3987     \ifin@
3988       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3989       \fi
3990     \else
3991       \global\bbl@csarg\chardef{wdir@#1}\z@
3992       \fi
3993   \ifodd\bbl@engine
3994     \bbl@csarg\ifcase{wdir@#1}%
3995       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
3996     \or
3997       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
3998     \or
3999       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4000     \fi
4001   \fi}
4002 \def\bbl@switchdir{%
4003   \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4004   \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4005   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4006 \def\bbl@setdirs#1{% TODO - math
4007   \ifcase\bbl@select@type % TODO - strictly, not the right test
4008     \bbl@bodydir{#1}%
4009     \bbl@pardir{#1}%
4010   \fi
4011   \bbl@texdir{#1}}
4012 % TODO. Only if \bbl@bidimode > 0?:
4013 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4014 \DisableBabelHook{babel-bidi}

```

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

```

4015 \ifodd\bbl@engine % luatex=1
4016 \else % pdftex=0, xetex=2
4017   \newcount\bbl@dirlevel
4018   \chardef\bbl@thetextdir\z@
4019   \chardef\bbl@thepardir\z@
4020   \def\bbl@texdir#1{%
4021     \ifcase#1\relax
4022       \chardef\bbl@thetextdir\z@
4023       \bbl@texdir@i\beginL\endL
4024     \else
4025       \chardef\bbl@thetextdir\@ne
4026       \bbl@texdir@i\beginR\endR
4027     \fi}
4028   \def\bbl@texdir@i#1#2{%
4029     \ifhmode
4030       \ifnum\currentgrouplevel>\z@
4031         \ifnum\currentgrouplevel=\bbl@dirlevel
4032           \bbl@error{Multiple bidi settings inside a group}%
4033           {I'll insert a new group, but expect wrong results.}%
4034           \group\aftergroup#2\aftergroup\egroup
4035         \else

```

```

4036     \ifcase\currentgrouptype\or % 0 bottom
4037     \aftergroup#2% 1 simple {}
4038     \or
4039     \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4040     \or
4041     \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4042     \or\or\or % vbox vtop align
4043     \or
4044     \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4045     \or\or\or\or\or\or % output math disc insert vcent mathchoice
4046     \or
4047     \aftergroup#2% 14 \begingroup
4048     \else
4049     \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4050     \fi
4051     \fi
4052     \bbl@dirlevel\currentgrouplevel
4053     \fi
4054     #1%
4055     \fi}
4056 \def\bbl@paddir#1{\chardef\bbl@thepaddir#1\relax}
4057 \let\bbl@bodydir@gobble
4058 \let\bbl@pagedir@gobble
4059 \def\bbl@dirparastext{\chardef\bbl@thepaddir\bbl@thetextdir}

```

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

```

4060 \def\bbl@xebidipar{%
4061   \let\bbl@xebidipar\relax
4062   \TeXeTstate\@ne
4063   \def\bbl@xeeverypar{%
4064     \ifcase\bbl@thepaddir
4065     \ifcase\bbl@thetextdir\else\beginR\fi
4066     \else
4067     {\setbox\z@\lastbox\beginR\box\z@}%
4068     \fi}%
4069   \let\bbl@severypar\everypar
4070   \newtoks\everypar
4071   \everypar=\bbl@severypar
4072   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4073 \ifnum\bbl@bidimode>200
4074   \let\bbl@textdir@i@gobbletwo
4075   \let\bbl@xebidipar\@empty
4076   \AddBabelHook{bidi}{foreign}{%
4077     \def\bbl@tempa{\def\BabelText####1}%
4078     \ifcase\bbl@thetextdir
4079     \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4080     \else
4081     \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4082     \fi}
4083   \def\bbl@paddir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4084   \fi
4085 \fi

```

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

```

4086 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}}
4087 \AtBeginDocument{%
4088   \ifx\pdfstringdefDisableCommands\@undefined\else
4089     \ifx\pdfstringdefDisableCommands\relax\else
4090     \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4091     \fi
4092   \fi}

```

8.6 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`.

```
4093 \bbl@trace{Local Language Configuration}
4094 \ifx\loadlocalcfg\undefined
4095   \@ifpackagewith{babel}{noconfigs}%
4096   {\let\loadlocalcfg@gobble}%
4097   {\def\loadlocalcfg#1{%
4098     \InputIfFileExists{#1.cfg}%
4099     {\typeout{*****^J%
4100               * Local config file #1.cfg used^^J%
4101               *}}}%
4102   \@empty}}
4103 \fi
```

8.7 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).

```
4104 \bbl@trace{Language options}
4105 \let\bbl@afterlang\relax
4106 \let\BabelModifiers\relax
4107 \let\bbl@loaded\@empty
4108 \def\bbl@load@language#1{%
4109   \InputIfFileExists{#1.ldf}%
4110   {\edef\bbl@loaded{\CurrentOption
4111     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4112     \expandafter\let\expandafter\bbl@afterlang
4113     \csname\CurrentOption.ldf-h@@k\endcsname
4114     \expandafter\let\expandafter\BabelModifiers
4115     \csname\bbl@mod@\CurrentOption\endcsname}%
4116   {\bbl@error{%
4117     Unknown option '\CurrentOption'. Either you misspelled it\\%
4118     or the language definition file \CurrentOption.ldf was not found}%
4119     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4120     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4121     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.

```
4122 \def\bbl@try@load@lang#1#2#3{%
4123   \IfFileExists{\CurrentOption.ldf}%
4124   {\bbl@load@language{\CurrentOption}}%
4125   {#1\bbl@load@language{#2}#3}}
4126 %
4127 \DeclareOption{hebrew}{%
4128   \input{rlbabel.def}%
4129   \bbl@load@language{hebrew}}
4130 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4131 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4132 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4133 \DeclareOption{polutonikogreek}{%
4134   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4135 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4136 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4137 \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.

```

4138 \ifx\bbbl@opt@config\@nnil
4139   \@ifpackagewith{babel}{noconfigs}}}%
4140   {\InputIfFileExists{bblopts.cfg}%
4141     {\typeout{*****^J%
4142               * Local config file bblopts.cfg used^^J%
4143               *}}}%
4144     {}}}%
4145 \else
4146   \InputIfFileExists{\bbbl@opt@config.cfg}%
4147   {\typeout{*****^J%
4148             * Local config file \bbbl@opt@config.cfg used^^J%
4149             *}}}%
4150   {\bbbl@error{%
4151     Local config file '\bbbl@opt@config.cfg' not found}{%
4152     Perhaps you misspelled it.}}}%
4153 \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 `bbbl@language@opts` are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the main language, which is processed in the third ‘main’ pass, *except* if all files are `ldf` and there is no main key. In the latter case (`\bbbl@opt@main` is still `\@nnil`), the traditional way to set the main language is kept — the last loaded is the main language.

```

4154 \ifx\bbbl@opt@main\@nnil
4155   \ifnum\bbbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4156     \let\bbbl@tempb\@empty
4157     \edef\bbbl@tempa{\@classoptionslist,\bbbl@language@opts}%
4158     \bbbl@foreach\bbbl@tempa{\edef\bbbl@tempb{#1,\bbbl@tempb}}}%
4159     \bbbl@foreach\bbbl@tempb{% \bbbl@tempb is a reversed list
4160       \ifx\bbbl@opt@main\@nnil % ie, if not yet assigned
4161         \ifodd\bbbl@iniflag % = *=
4162           \IfFileExists{babel-#1.tex}{\def\bbbl@opt@main{#1}}}%
4163         \else % n +=
4164           \IfFileExists{#1.ldf}{\def\bbbl@opt@main{#1}}}%
4165       \fi
4166     \fi}%
4167 \fi
4168 \else
4169   \bbbl@info{Main language set with 'main='. Except if you have\\%
4170     problems, prefer the default mechanism for setting\\%
4171     the main language. Reported}%
4172 \fi

```

A few languages are still defined explicitly. They are stored in case they are needed in the ‘main’ pass (the value can be `\relax`).

```

4173 \ifx\bbbl@opt@main\@nnil\else
4174   \bbbl@csarg\let{loadmain\expandafter}\csname ds@\bbbl@opt@main\endcsname
4175   \expandafter\let\csname ds@\bbbl@opt@main\endcsname\relax
4176 \fi

```

Now define the corresponding loaders. With package options, assume the language exists. With class options, check if the option is a language by checking if the correspondin file exists.

```

4177 \bbbl@foreach\bbbl@language@opts{%
4178   \def\bbbl@tempa{#1}%
4179   \ifx\bbbl@tempa\bbbl@opt@main\else
4180     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4181       \bbbl@ifunset{ds@#1}%
4182       {\DeclareOption{#1}{\bbbl@load@language{#1}}}%

```

```

4183     {}%
4184 \else % + * (other = ini)
4185 \DeclareOption{#1}{%
4186 \bbl@ldfinit
4187 \babelprovide[import]{#1}%
4188 \bbl@afterldf{}}%
4189 \fi
4190 \fi}
4191 \bbl@foreach\@classoptionslist{%
4192 \def\bbl@tempa{#1}%
4193 \ifx\bbl@tempa\bbl@opt@main\else
4194 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4195 \bbl@ifunset{ds@#1}%
4196 {\IfFileExists{#1.ldf}%
4197 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4198 {}}%
4199 {}%
4200 \else % + * (other = ini)
4201 \IfFileExists{babel-#1.tex}%
4202 {\DeclareOption{#1}{%
4203 \bbl@ldfinit
4204 \babelprovide[import]{#1}%
4205 \bbl@afterldf{}}}%
4206 {}}%
4207 \fi
4208 \fi}

```

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

The options have to be processed in the order in which the user specified them (but remember class options are processed before):

```

4209 \def\AfterBabelLanguage#1{%
4210 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4211 \DeclareOption*{}
4212 \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. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \babelprovide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

```

4213 \bbl@trace{Option 'main'}
4214 \ifx\bbl@opt@main\@nnil
4215 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4216 \let\bbl@tempc\@empty
4217 \bbl@for\bbl@tempb\bbl@tempa{%
4218 \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4219 \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4220 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4221 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4222 \ifx\bbl@tempb\bbl@tempc\else
4223 \bbl@warning{%
4224 Last declared language option is '\bbl@tempc',\%
4225 but the last processed one was '\bbl@tempb'.\%
4226 The main language can't be set as both a global\%
4227 and a package option. Use 'main=\bbl@tempc' as\%
4228 option. Reported}
4229 \fi
4230 \else
4231 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4232 \bbl@ldfinit
4233 \let\CurrentOption\bbl@opt@main
4234 \bbl@exp{% \bbl@opt@provide = empty if *

```

```

4235     \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4236     \bbl@afterldf{}
4237     \DeclareOption{\bbl@opt@main}{}
4238 \else % case 0,2 (main is ldf)
4239     \ifx\bbl@loadmain\relax
4240         \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4241     \else
4242         \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4243     \fi
4244     \ExecuteOptions{\bbl@opt@main}
4245     \@namedef{ds@\bbl@opt@main}{}%
4246 \fi
4247 \DeclareOption*{}
4248 \ProcessOptions*
4249 \fi
4250 \def\AfterBabelLanguage{%
4251     \bbl@error
4252     {Too late for \string\AfterBabelLanguage}%
4253     {Languages have been loaded, so I can do nothing}}

In order to catch the case where the user didn't specify a language we check whether
\bbl@main@language, has become defined. If not, the nil language is loaded.

4254 \ifx\bbl@main@language\undefined
4255     \bbl@info{%
4256         You haven't specified a language. I'll use 'nil'\%
4257         as the main language. Reported}
4258     \bbl@load@language{nil}
4259 \fi
4260 \</package>

```

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

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

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

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

A proxy file for switch.def

```

4261 \<*kernel>
4262 \let\bbl@onlyswitch\@empty
4263 \input babel.def
4264 \let\bbl@onlyswitch\@undefined
4265 \</kernel>
4266 \<*patterns>

```

10 Loading hyphenation patterns

The following code is meant to be read by iniT_EX because it should instruct T_EX 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.

```

4267 \<<Make sure ProvidesFile is defined>>
4268 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4269 \xdef\bbl@format{\jobname}
4270 \def\bbl@version{\<<version>>}
4271 \def\bbl@date{\<<date>>}
4272 \ifx\AtBeginDocument\@undefined

```

```

4273 \def\@empty{}
4274 \fi
4275 <⟨Define core switching macros⟩>

```

`\process@line` Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with `=`. When the first token of a line is an `=`, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

```

4276 \def\process@line#1#2 #3 #4 {%
4277   \ifx=#1%
4278     \process@synonym{#2}%
4279   \else
4280     \process@language{#1#2}{#3}{#4}%
4281   \fi
4282   \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.

```

4283 \toks@{}
4284 \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 `hyphenmins` parameters for the synonym.

```

4285 \def\process@synonym#1{%
4286   \ifnum\last@language=m@ne
4287     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4288   \else
4289     \expandafter\chardef\csname l@#1\endcsname\last@language
4290     \wlog{\string\l@#1=\string\language\the\last@language}%
4291     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4292       \csname\language\hyphenmins\endcsname
4293     \let\bbl@elt\relax
4294     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
4295   \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` `\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.

```

4296 \def\process@language#1#2#3{%
4297   \expandafter\addlanguage\csname l@#1\endcsname
4298   \expandafter\language\csname l@#1\endcsname
4299   \edef\language{#1}%
4300   \bbl@hook@everylanguage{#1}%
4301   % > luatex
4302   \bbl@get@enc#1:.\@@@
4303   \begingroup
4304     \lefthyphenmin@m@ne
4305     \bbl@hook@loadpatterns{#2}%
4306     % > luatex
4307     \ifnum\lefthyphenmin=m@ne
4308     \else
4309       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4310         \the\lefthyphenmin\the\righthyphenmin}%
4311     \fi
4312   \endgroup
4313   \def\bbl@tempa{#3}%
4314   \ifx\bbl@tempa\@empty\else
4315     \bbl@hook@loadexceptions{#3}%
4316     % > luatex
4317   \fi
4318   \let\bbl@elt\relax
4319   \edef\bbl@languages{%
4320     \bbl@languages\bbl@elt{#1}\the\language}{#2}{\bbl@tempa}}%
4321   \ifnum\the\language=z@
4322     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4323       \set@hyphenmins\tw@\thr@@\relax
4324     \else
4325       \expandafter\expandafter\expandafter\set@hyphenmins
4326       \csname #1hyphenmins\endcsname
4327     \fi
4328     \the\toks@
4329     \toks@{}%
4330   \fi}

```

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

```

4331 \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.

```

4332 \def\bbl@hook@everylanguage#1{}
4333 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4334 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4335 \def\bbl@hook@loadkernel#1{%
4336   \def\addlanguage{\csname newlanguage\endcsname}%
4337   \def\adddialect##1##2{%
4338     \global\chardef##1##2\relax
4339     \wlog{\string##1 = a dialect from \string\language##2}}%
4340   \def\iflanguage##1{%
4341     \expandafter\ifx\csname l@##1\endcsname\relax
4342       \nolannerr{##1}%
4343     \else
4344       \ifnum\csname l@##1\endcsname=\language
4345         \expandafter\expandafter\expandafter\@firstoftwo
4346       \else
4347         \expandafter\expandafter\expandafter\@secondoftwo
4348       \fi
4349     \fi}%

```



```

4350 \def\providehyphenmins##1##2{%
4351 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4352 \namedef{##1hyphenmins}{##2}%
4353 \fi}%
4354 \def\set@hyphenmins##1##2{%
4355 \lefthyphenmin##1\relax
4356 \righthyphenmin##2\relax}%
4357 \def\selectlanguage{%
4358 \errhelp{Selecting a language requires a package supporting it}%
4359 \errmessage{Not loaded}}%
4360 \let\foreignlanguage\selectlanguage
4361 \let\otherlanguage\selectlanguage
4362 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4363 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4364 \def\setlocale{%
4365 \errhelp{Find an armchair, sit down and wait}%
4366 \errmessage{Not yet available}}%
4367 \let\uselocale\setlocale
4368 \let\locale\setlocale
4369 \let\selectlocale\setlocale
4370 \let\localename\setlocale
4371 \let\textlocale\setlocale
4372 \let\textlanguage\setlocale
4373 \let\languagetext\setlocale}
4374 \begingroup
4375 \def\AddBabelHook#1#2{%
4376 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4377 \def\next{\toks1}%
4378 \else
4379 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4380 \fi
4381 \next}
4382 \ifx\directlua\undefined
4383 \ifx\XeTeXinputencoding\undefined\else
4384 \input xebabel.def
4385 \fi
4386 \else
4387 \input luababel.def
4388 \fi
4389 \openin1 = babel-\bbl@format.cfg
4390 \ifeof1
4391 \else
4392 \input babel-\bbl@format.cfg\relax
4393 \fi
4394 \closein1
4395 \endgroup
4396 \bbl@hook@loadkernel{switch.def}

```

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

```

4397 \openin1 = language.dat

```

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

```

4398 \def\languagename{english}%
4399 \ifeof1
4400 \message{I couldn't find the file language.dat,\space
4401 I will try the file hyphen.tex}
4402 \input hyphen.tex\relax
4403 \chardef\l@english\z@
4404 \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`.

```
4405 \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.

```
4406 \loop
4407   \endlinechar\m@ne
4408   \read1 to \bbl@line
4409   \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.

```
4410   \if T\ifeof1F\fi T\relax
4411   \ifx\bbl@line\@empty\else
4412     \edef\bbl@line{\bbl@line\space\space\space}%
4413     \expandafter\process@line\bbl@line\relax
4414   \fi
4415 \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.

```
4416 \begingroup
4417   \def\bbl@elt#1#2#3#4{%
4418     \global\language=#2\relax
4419     \gdef\languagename{#1}%
4420     \def\bbl@elt##1##2###3###4{}}%
4421   \bbl@languages
4422 \endgroup
4423 \fi
4424 \closein1
```

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

```
4425 \if/\the\toks@\else
4426   \errhelp{language.dat loads no language, only synonyms}
4427   \errmessage{Orphan language synonym}
4428 \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.

```
4429 \let\bbl@line\@undefined
4430 \let\process@line\@undefined
4431 \let\process@synonym\@undefined
4432 \let\process@language\@undefined
4433 \let\bbl@get@enc\@undefined
4434 \let\bbl@hyph@enc\@undefined
4435 \let\bbl@tempa\@undefined
4436 \let\bbl@hook@loadkernel\@undefined
4437 \let\bbl@hook@everylanguage\@undefined
4438 \let\bbl@hook@loadpatterns\@undefined
4439 \let\bbl@hook@loadexceptions\@undefined
4440 \</patterns>
```

Here the code for `iniTeX` ends.

11 Font handling with fontspec

Add the bidi handler just before `luaotfload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
4441 <{*More package options}> ≡
```

```

4442 \chardef\bbl@bidimode\z@
4443 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4444 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4445 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4446 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4447 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4448 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4449 <{/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.

```

4450 <{*Font selection}> ≡
4451 \bbl@trace{Font handling with fontspec}
4452 \ifx\ExplSyntaxOn\@undefined\else
4453   \ExplSyntaxOn
4454   \catcode\ =10
4455   \def\bbl@loadfontspec{%
4456     \usepackage{fontspec}% TODO. Apply patch always
4457     \expandafter
4458     \def\csname msg-text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4459       Font '\l_fontspec_fontname_tl' is using the\\%
4460       default features for language '##1'.\\%
4461       That's usually fine, because many languages\\%
4462       require no specific features, but if the output is\\%
4463       not as expected, consider selecting another font.}
4464     \expandafter
4465     \def\csname msg-text~>~fontspec/no-script\endcsname##1##2##3##4{%
4466       Font '\l_fontspec_fontname_tl' is using the\\%
4467       default features for script '##2'.\\%
4468       That's not always wrong, but if the output is\\%
4469       not as expected, consider selecting another font.}}
4470   \ExplSyntaxOff
4471 \fi
4472 \@onlypreamble\babelfont
4473 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4474   \bbl@foreach{#1}{%
4475     \expandafter\ifx\csname date##1\endcsname\relax
4476       \IfFileExists{babel-##1.tex}%
4477       {\babelprovide{##1}}%
4478       {}%
4479     \fi}%
4480   \edef\bbl@tempa{#1}%
4481   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4482   \ifx\fontspec\@undefined
4483     \bbl@loadfontspec
4484   \fi
4485   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4486   \bbl@bblfont}
4487 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4488   \bbl@ifunset{\bbl@tempb family}%
4489   {\bbl@providefam{\bbl@tempb}}%
4490   {}%
4491   % For the default font, just in case:
4492   \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
4493   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4494   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4495   \bbl@exp{%
4496     \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4497     \bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%

```

```

4498         \<\bbl@tempb default>\<\bbl@tempb family>}}}%
4499     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4500         \bbl@csarg\def{\bbl@tempb dflt##1}{<#1>{#2}}}}}%

```

If the family in the previous command does not exist, it must be defined. Here is how:

```

4501 \def\bbl@providfam#1{%
4502   \bbl@exp{%
4503     \\newcommand\<#1default>{}% Just define it
4504     \\bbl@add@list\\bbl@font@fams{#1}%
4505     \\DeclareRobustCommand\<#1family>{%
4506       \\not@math@alphabet\<#1family>\relax
4507       % \\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4508       \\fontfamily\<#1default>%
4509       \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4510       \\selectfont}%
4511     \\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.

```

4512 \def\bbl@nostdfont#1{%
4513   \bbl@ifunset{\bbl@WFF@f@family}%
4514   {\bbl@csarg\gdef{\bbl@WFF@f@family}}}% Flag, to avoid dupl warns
4515   \bbl@infowarn{The current font is not a babel standard family:\\%
4516     #1%
4517     \fontname\font\\%
4518     There is nothing intrinsically wrong with this warning, and\\%
4519     you can ignore it altogether if you do not need these\\%
4520     families. But if they are used in the document, you should be\\%
4521     aware 'babel' will not set Script and Language for them, so\\%
4522     you may consider defining a new family with \string\babelfont.\\%
4523     See the manual for further details about \string\babelfont.\\%
4524     Reported}}
4525   {}}}%
4526 \gdef\bbl@switchfont{%
4527   \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys{\language}}}%
4528   \bbl@exp{% eg Arabic -> arabic
4529   \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}}}%
4530   \bbl@foreach\bbl@font@fams{%
4531     \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4532     {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4533       {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4534         {}}% 123=F - nothing!
4535         {\bbl@exp{% 3=T - from generic
4536           \global\let\<bbl@##1dflt@\language>%
4537           \<bbl@##1dflt@>}}}%
4538         {\bbl@exp{% 2=T - from script
4539           \global\let\<bbl@##1dflt@\language>%
4540           \<bbl@##1dflt@*\bbl@tempa>}}}%
4541         {}}% 1=T - language, already defined
4542   \def\bbl@tempa{\bbl@nostdfont}}}%
4543   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4544     \bbl@ifunset{\bbl@##1dflt@\language}%
4545     {\bbl@cs{famrst##1}%
4546       \global\bbl@csarg\let{famrst##1}\relax}%
4547     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4548       \\bbl@add\\originalTeX%
4549       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4550       \<##1default>\<##1family>{##1}}}%
4551     \\bbl@font@set\<bbl@##1dflt@\language>% the main part!
4552     \<##1default>\<##1family>}}}%
4553   \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.

```

4554 \ifx\fbfamily\undefined\else % if latex
4555 \ifcase\bbl@engine % if pdftex
4556 \let\bbl@cckstdfonts\relax
4557 \else
4558 \def\bbl@cckstdfonts{%
4559 \begingroup
4560 \global\let\bbl@cckstdfonts\relax
4561 \let\bbl@tempa\@empty
4562 \bbl@foreach\bbl@font@fams{%
4563 \bbl@ifunset\bbl@##1dflt@{%
4564 {\@nameuse{##1family}}%
4565 \bbl@csarg\gdef{WFF@\fbfamily}}}% Flag
4566 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \fbfamily\\}%
4567 \space\space\fontname\font\\}%
4568 \bbl@csarg\xdef{##1dflt@}{\fbfamily}%
4569 \expandafter\xdef\csname ##1default\endcsname{\fbfamily}}%
4570 {}}%
4571 \ifx\bbl@tempa\@empty\else
4572 \bbl@infowarn{The following font families will use the default\\%
4573 settings for all or some languages:\\%
4574 \bbl@tempa
4575 There is nothing intrinsically wrong with it, but\\%
4576 'babel' will no set Script and Language, which could\\%
4577 be relevant in some languages. If your document uses\\%
4578 these families, consider redefining them with \string\babelfont.\\%
4579 Reported}%
4580 \fi
4581 \endgroup}
4582 \fi
4583 \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.

```

4584 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4585 \bbl@xin@{<>}{#1}%
4586 \ifin@
4587 \bbl@exp{\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4588 \fi
4589 \bbl@exp{%
4590 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4591 \\bbl@ifsamestring{#2}{\fbfamily}%
4592 {\\#3%
4593 \\bbl@ifsamestring{\fbseries}{\bfdefault}{\\bfseries}}}%
4594 \let\\bbl@tempa\relax}%
4595 {}}}
4596 % TODO - next should be global?, but even local does its job. I'm
4597 % still not sure -- must investigate:
4598 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4599 \let\bbl@tempa\bbl@mapselect
4600 \let\bbl@mapselect\relax
4601 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4602 \let#4\@empty % Make sure \renewfontfamily is valid
4603 \bbl@exp{%
4604 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4605 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4606 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4607 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4608 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4609 \renewfontfamily\\#4%
4610 [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{.}{#3}
4611 \begingroup

```

```

4612      #4%
4613      \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4614 \endgroup
4615 \let#4\bbl@temp@fam
4616 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4617 \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.

```

4618 \def\bbl@font@rst#1#2#3#4{%
4619 \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.

```

4620 \def\bbl@font@fams{rm,sf,tt}
4621 <</Font selection>>

```

12 Hooks for XeTeX and LuaTeX

12.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

```

4622 <<{*Footnote changes}>> ≡
4623 \bbl@trace{Bidi footnotes}
4624 \ifnum\bbl@bidimode>\z@
4625 \def\bbl@footnote#1#2#3{%
4626 \@ifnextchar[%
4627 {\bbl@footnote@o{#1}{#2}{#3}}%
4628 {\bbl@footnote@x{#1}{#2}{#3}}}
4629 \long\def\bbl@footnote@x#1#2#3#4{%
4630 \bgroup
4631 \select@language@x{\bbl@main@language}%
4632 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4633 \egroup}
4634 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4635 \bgroup
4636 \select@language@x{\bbl@main@language}%
4637 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4638 \egroup}
4639 \def\bbl@footnotetext#1#2#3{%
4640 \@ifnextchar[%
4641 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4642 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4643 \long\def\bbl@footnotetext@x#1#2#3#4{%
4644 \bgroup
4645 \select@language@x{\bbl@main@language}%
4646 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4647 \egroup}
4648 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4649 \bgroup
4650 \select@language@x{\bbl@main@language}%
4651 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4652 \egroup}
4653 \def\BabelFootnote#1#2#3#4{%
4654 \ifx\bbl@fn@footnote\@undefined
4655 \let\bbl@fn@footnote\footnote
4656 \fi
4657 \ifx\bbl@fn@footnotetext\@undefined
4658 \let\bbl@fn@footnotetext\footnotetext
4659 \fi
4660 \bbl@ifblank{#2}%
4661 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}

```

```

4662 \namedef{\bbl@stripslash#1text}%
4663 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4664 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}}%
4665 \namedef{\bbl@stripslash#1text}%
4666 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4667 \fi
4668 <\/Footnote changes>

```

Now, the code.

```

4669 <*xetex>
4670 \def\BabelStringsDefault{unicode}
4671 \let\xebbl@stop\relax
4672 \AddBabelHook{xetex}{encodedcommands}{%
4673 \def\bbl@tempa{#1}%
4674 \ifx\bbl@tempa@empty
4675 \XeTeXinputencoding"bytes"%
4676 \else
4677 \XeTeXinputencoding"#1"%
4678 \fi
4679 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4680 \AddBabelHook{xetex}{stopcommands}{%
4681 \xebbl@stop
4682 \let\xebbl@stop\relax}
4683 \def\bbl@intraspace#1 #2 #3\@{%
4684 \bbl@csarg\gdef{xensp@language}%
4685 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4686 \def\bbl@intrapenalty#1\@{%
4687 \bbl@csarg\gdef{xepn@language}%
4688 {\XeTeXlinebreakpenalty #1\relax}}
4689 \def\bbl@provide@intraspace{%
4690 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4691 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4692 \ifin@
4693 \bbl@ifunset{bbl@intsp@language}{%
4694 {\expandafter\ifx\csname bbl@intsp@language\endcsname@empty\else
4695 \ifx\bbl@KVP@intraspace\@nnil
4696 \bbl@exp{%
4697 \bbl@intraspace\bbl@cl{intsp}\@}%
4698 \fi
4699 \ifx\bbl@KVP@intrapenalty\@nnil
4700 \bbl@intrapenalty0\@
4701 \fi
4702 \fi
4703 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4704 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
4705 \fi
4706 \ifx\bbl@KVP@intrapenalty\@nnil\else
4707 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
4708 \fi
4709 \bbl@exp{%
4710 % TODO. Execute only once (but redundant):
4711 \bbl@add\<extras\language>{%
4712 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4713 \<bbl@xensp@language>%
4714 \<bbl@xeipn@language>}%
4715 \bbl@tglobal\<extras\language>%
4716 \bbl@add\<noextras\language>{%
4717 \XeTeXlinebreaklocale "en"%
4718 \bbl@tglobal\<noextras\language>}%
4719 \ifx\bbl@ispace\@undefined
4720 \gdef\bbl@ispace{\bbl@cl{xensp}}}%
4721 \ifx\AtBeginDocument\@notprerr
4722 \expandafter\@secondoftwo % to execute right now

```

```

4723      \fi
4724      \AtBeginDocument{\bbl@patchfont{\bbl@ispacefont}}%
4725      \fi}%
4726  \fi}
4727  \ifx\DisableBabelHook\undefined\endinput\fi
4728  \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4729  \AddBabelHook{babel-fontspec}{beforestart}{\bbl@checkstdfonts}
4730  \DisableBabelHook{babel-fontspec}
4731  <<Font selection>>
4732  \input txtbabel.def
4733  </xetex>

```

12.2 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 T_EX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdftex and xetex.

```

4734 <*texet>
4735 \providecommand\bbl@provide@intraspace{}
4736 \bbl@trace{Redefinitions for bidi layout}
4737 \def\bbl@sspre@caption{%
4738   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}%
4739   \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4740   \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4741   \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4742   \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4743     \def\hangfrom#1{%
4744       \setbox\@tempboxa\hbox{#1}%
4745       \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4746       \noindent\box\@tempboxa}
4747   \def\raggedright{%
4748     \let\@centercr
4749     \bbl@startskip\z@skip
4750     \@rightskip\@flushglue
4751     \bbl@endskip\@rightskip
4752     \parindent\z@
4753     \parfillskip\bbl@startskip}
4754   \def\raggedleft{%
4755     \let\@centercr
4756     \bbl@startskip\@flushglue
4757     \bbl@endskip\z@skip
4758     \parindent\z@
4759     \parfillskip\bbl@endskip}
4760 \fi
4761 \IfBabelLayout{lists}
4762   {\bbl@sreplace\list
4763     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4764     \def\bbl@listleftmargin{%
4765       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4766     \ifcase\bbl@engine
4767       \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
4768       \def\p@enumiii{\p@enumii}\theenumii{}%
4769     \fi
4770     \bbl@sreplace\@verbatim
4771       {\leftskip\@totalleftmargin}%
4772       {\bbl@startskip\textwidth
4773         \advance\bbl@startskip-\linewidth}%
4774     \bbl@sreplace\@verbatim
4775       {\rightskip\z@skip}%

```



```

4776     {\bbl@endskip\z@skip}}%
4777 {}
4778 \IfBabelLayout{contents}%
4779 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4780  \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4781 {}
4782 \IfBabelLayout{columns}%
4783 {\bbl@sreplace\@outputdblcol{\hb@xt\textwidth}{\bbl@outputbox}%
4784  \def\bbl@outputbox#1{%
4785    \hb@xt\textwidth{%
4786      \hskip\columnwidth
4787      \hfil
4788      {\normalcolor\vrule \@width\columnseprule}%
4789      \hfil
4790      \hb@xt\columnwidth{\box\@leftcolumn \hss}%
4791      \hskip-\textwidth
4792      \hb@xt\columnwidth{\box\@outputbox \hss}%
4793      \hskip\columnsep
4794      \hskip\columnwidth}}}%
4795 {}
4796 <<Footnote changes>>
4797 \IfBabelLayout{footnotes}%
4798 {\BabelFootnote\footnote\language{}{}}%
4799  \BabelFootnote\localfootnote\language{}{}}%
4800  \BabelFootnote\mainfootnote{}{}}{}
4801 {}

```

Implicitly reverses sectioning labels in `bidibasic`, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

4802 \IfBabelLayout{counters}%
4803 {\let\bbl@latinarabic=\@arabic
4804  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4805  \let\bbl@asciroman=\@roman
4806  \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4807  \let\bbl@asciiRoman=\@Roman
4808  \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4809 </texxet>

```

12.3 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).

```

4810 <*luatex>
4811 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4812 \bbl@trace{Read language.dat}
4813 \ifx\bbl@readstream\@undefined
4814   \csname newread\endcsname\bbl@readstream
4815 \fi
4816 \begingroup
4817   \toks@{}
4818   \count\z@ % 0=start, 1=0th, 2=normal
4819   \def\bbl@process@line#1#2 #3 #4 {%
4820     \ifx=#1%
4821       \bbl@process@synonym{#2}%
4822     \else
4823       \bbl@process@language{#1#2}{#3}{#4}%
4824     \fi
4825     \ignorespaces}
4826   \def\bbl@manylang{%
4827     \ifnum\bbl@last>\@ne
4828       \bbl@info{Non-standard hyphenation setup}%
4829     \fi
4830     \let\bbl@manylang\relax}
4831   \def\bbl@process@language#1#2#3{%
4832     \ifcase\count@
4833       \@ifundefined{zth@#1}{\count@tw@}{\count@ne}%
4834     \or
4835       \count@tw@
4836     \fi
4837     \ifnum\count@=tw@
4838       \expandafter\addlanguage\csname l@#1\endcsname
4839       \language\allocationnumber
4840       \chardef\bbl@last\allocationnumber
4841       \bbl@manylang
4842       \let\bbl@elt\relax
4843       \xdef\bbl@languages{%
4844         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4845     \fi
4846     \the\toks@
4847     \toks@{}}
4848   \def\bbl@process@synonym@aux#1#2{%
4849     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4850     \let\bbl@elt\relax
4851     \xdef\bbl@languages{%
4852       \bbl@languages\bbl@elt{#1}{#2}{}}}%
4853   \def\bbl@process@synonym#1{%
4854     \ifcase\count@
4855       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4856     \or
4857       \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4858     \else
4859       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4860     \fi}
4861   \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4862     \chardef\l@english\z@

```

```

4863 \chardef\l@USenglish\z@
4864 \chardef\bbl@last\z@
4865 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex{}}
4866 \gdef\bbl@languages{%
4867   \bbl@elt{english}{0}{\hyphen.tex{}}%
4868   \bbl@elt{USenglish}{0}{}}
4869 \else
4870   \global\let\bbl@languages@format\bbl@languages
4871   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4872     \ifnum#2>\z@\else
4873       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4874     \fi}%
4875   \xdef\bbl@languages{\bbl@languages}%
4876 \fi
4877 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4878 \bbl@languages
4879 \openin\bbl@readstream=language.dat
4880 \ifeof\bbl@readstream
4881   \bbl@warning{I couldn't find language.dat. No additional\\%
4882     patterns loaded. Reported}%
4883 \else
4884   \loop
4885     \endlinechar\m@ne
4886     \read\bbl@readstream to \bbl@line
4887     \endlinechar``^^M
4888     \if T\ifeof\bbl@readstream F\fi T\relax
4889     \ifx\bbl@line\@empty\else
4890       \edef\bbl@line{\bbl@line\space\space\space}%
4891       \expandafter\bbl@process@line\bbl@line\relax
4892     \fi
4893   \repeat
4894 \fi
4895 \endgroup
4896 \bbl@trace{Macros for reading patterns files}
4897 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4898 \ifx\babelcatcodetablenum\@undefined
4899   \ifx\newcatcodetable\@undefined
4900     \def\babelcatcodetablenum{5211}
4901     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4902   \else
4903     \newcatcodetable\babelcatcodetablenum
4904     \newcatcodetable\bbl@pattcodes
4905   \fi
4906 \else
4907   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4908 \fi
4909 \def\bbl@luapatterns#1#2{%
4910   \bbl@get@enc#1::\@@@
4911   \setbox\z@\hbox\bgroup
4912   \begingroup
4913     \savecatcodetable\babelcatcodetablenum\relax
4914     \initcatcodetable\bbl@pattcodes\relax
4915     \catcodetable\bbl@pattcodes\relax
4916     \catcode`\#6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4917     \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4918     \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4919     \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4920     \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4921     \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4922     \input #1\relax
4923     \catcodetable\babelcatcodetablenum\relax
4924   \endgroup
4925   \def\bbl@tempa{#2}%

```

```

4926 \ifx\bbl@tempa\@empty\else
4927 \input #2\relax
4928 \fi
4929 \egroup}%
4930 \def\bbl@patterns@lua#1{%
4931 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4932 \csname l@#1\endcsname
4933 \edef\bbl@tempa{#1}%
4934 \else
4935 \csname l@#1:\f@encoding\endcsname
4936 \edef\bbl@tempa{#1:\f@encoding}%
4937 \fi\relax
4938 \@namedef{luatexhyphen@loaded@the\language}{}% Temp
4939 \@ifundefined{bbl@hyphendata@the\language}%
4940 {\def\bbl@elt##1##2##3##4{%
4941 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4942 \def\bbl@tempb{##3}%
4943 \ifx\bbl@tempb\@empty\else % if not a synonymous
4944 \def\bbl@tempc{{##3}{##4}}%
4945 \fi
4946 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4947 \fi}%
4948 \bbl@languages
4949 \@ifundefined{bbl@hyphendata@the\language}%
4950 {\bbl@info{No hyphenation patterns were set for\%
4951 language '\bbl@tempa'. Reported}}%
4952 {\expandafter\expandafter\expandafter\bbl@luapatterns
4953 \csname bbl@hyphendata@the\language\endcsname}}}%
4954 \endinput\fi
4955 % Here ends \ifx\AddBabelHook\@undefined
4956 % A few lines are only read by hyphen.cfg
4957 \ifx\DisableBabelHook\@undefined
4958 \AddBabelHook{luatex}{everylanguage}{%
4959 \def\process@language##1##2##3{%
4960 \def\process@line####1####2 ####3 ####4 {}}}
4961 \AddBabelHook{luatex}{loadpatterns}{%
4962 \input #1\relax
4963 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4964 {{#1}}}%
4965 \AddBabelHook{luatex}{loadexceptions}{%
4966 \input #1\relax
4967 \def\bbl@tempb##1##2{{##1}{#1}}%
4968 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4969 {\expandafter\expandafter\expandafter\bbl@tempb
4970 \csname bbl@hyphendata@the\language\endcsname}}
4971 \endinput\fi
4972 % Here stops reading code for hyphen.cfg
4973 % The following is read the 2nd time it's loaded
4974 \begingroup % TODO - to a lua file
4975 \catcode\%=12
4976 \catcode\`=12
4977 \catcode\`=12
4978 \catcode\:=12
4979 \directlua{
4980 Babel = Babel or {}
4981 function Babel.bytes(line)
4982 return line:gsub("(.)",
4983 function (chr) return unicode.utf8.char(string.byte(chr)) end)
4984 end
4985 function Babel.begin_process_input()
4986 if luatexbase and luatexbase.add_to_callback then
4987 luatexbase.add_to_callback('process_input_buffer',
4988 Babel.bytes,'Babel.bytes')

```

```

4989     else
4990         Babel.callback = callback.find('process_input_buffer')
4991         callback.register('process_input_buffer',Babel.bytes)
4992     end
4993 end
4994 function Babel.end_process_input ()
4995     if luatexbase and luatexbase.remove_from_callback then
4996         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4997     else
4998         callback.register('process_input_buffer',Babel.callback)
4999     end
5000 end
5001 function Babel.addpatterns(pp, lg)
5002     local lg = lang.new(lg)
5003     local pats = lang.patterns(lg) or ''
5004     lang.clear_patterns(lg)
5005     for p in pp:gmatch('[^%s]+') do
5006         ss = ''
5007         for i in string.utfcharacters(p:gsub('%d', '')) do
5008             ss = ss .. '%d?' .. i
5009         end
5010         ss = ss:gsub('^%%d%?%.','%.') .. '%d?'
5011         ss = ss:gsub('%.%%d%?$', '%%.')
5012         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5013         if n == 0 then
5014             tex.sprint(
5015                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5016                 .. p .. [[]])
5017             pats = pats .. ' ' .. p
5018         else
5019             tex.sprint(
5020                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5021                 .. p .. [[]])
5022         end
5023     end
5024     lang.patterns(lg, pats)
5025 end
5026 Babel.characters = Babel.characters or {}
5027 Babel.ranges = Babel.ranges or {}
5028 function Babel.hlist_has_bidi(head)
5029     local has_bidi = false
5030     local ranges = Babel.ranges
5031     for item in node.traverse(head) do
5032         if item.id == node.id'glyph' then
5033             local itemchar = item.char
5034             local chardata = Babel.characters[itemchar]
5035             local dir = chardata and chardata.d or nil
5036             if not dir then
5037                 for nn, et in ipairs(ranges) do
5038                     if itemchar < et[1] then
5039                         break
5040                     elseif itemchar <= et[2] then
5041                         dir = et[3]
5042                         break
5043                     end
5044                 end
5045             end
5046             if dir and (dir == 'al' or dir == 'r') then
5047                 has_bidi = true
5048             end
5049         end
5050     end
5051     return has_bidi

```

```

5052 end
5053 function Babel.set_chrnges_b (script, chrng)
5054   if chrng == '' then return end
5055   texio.write('Replacing ' .. script .. ' script ranges')
5056   Babel.script_blocks[script] = {}
5057   for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5058     table.insert(
5059       Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5060   end
5061 end
5062 }
5063 \endgroup
5064 \ifx\newattribute\@undefined\else
5065   \newattribute\bbl@attr@locale
5066   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5067   \AddBabelHook{luatex}{beforeextras}{%
5068     \setattribute\bbl@attr@locale\localeid}
5069 \fi
5070 \def\BabelStringsDefault{unicode}
5071 \let\luabbl@stop\relax
5072 \AddBabelHook{luatex}{encodedcommands}{%
5073   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5074   \ifx\bbl@tempa\bbl@tempb\else
5075     \directlua{Babel.begin_process_input()}%
5076     \def\luabbl@stop{%
5077       \directlua{Babel.end_process_input()}}%
5078   \fi}%
5079 \AddBabelHook{luatex}{stopcommands}{%
5080   \luabbl@stop
5081   \let\luabbl@stop\relax}
5082 \AddBabelHook{luatex}{patterns}{%
5083   \@ifundefined{bbl@hyphendata@the\language}%
5084   {\def\bbl@elt##1###3###4{%
5085     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5086     \def\bbl@tempb{##3}%
5087     \ifx\bbl@tempb\@empty\else % if not a synonymous
5088       \def\bbl@tempc{##3}{##4}}%
5089     \fi
5090     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5091     \fi}%
5092   \bbl@languages
5093   \@ifundefined{bbl@hyphendata@the\language}%
5094   {\bbl@info{No hyphenation patterns were set for\%
5095     language '#2'. Reported}}%
5096   {\expandafter\expandafter\expandafter\bbl@luapatterns
5097     \csname bbl@hyphendata@the\language\endcsname}}}%
5098   \@ifundefined{bbl@patterns@}{}%
5099   \begingroup
5100     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5101     \ifin@ \else
5102       \ifx\bbl@patterns@\@empty\else
5103         \directlua{ Babel.addpatterns(
5104           [[\bbl@patterns@]], \number\language) }%
5105       \fi
5106       \@ifundefined{bbl@patterns@#1}%
5107       \@empty
5108       {\directlua{ Babel.addpatterns(
5109         [[\space\csname bbl@patterns@#1\endcsname]],
5110         \number\language) }}%
5111       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5112     \fi
5113   \endgroup}%
5114   \bbl@exp{%

```

```

5115 \bbl@ifunset{\bbl@prehc@language}\relax}%
5116 {\bbl@ifblank{\bbl@cs{prehc@language}}}%
5117 {\prehyphenchar=\bbl@cl{prehc}\relax}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

5118 \@onlypreamble\babelpatterns
5119 \AtEndOfPackage{%
5120 \newcommand\babelpatterns[2][\@empty]{%
5121 \ifx\bbl@patterns@relax
5122 \let\bbl@patterns@empty
5123 \fi
5124 \ifx\bbl@pttnlist@empty\else
5125 \bbl@warning{%
5126 You must not intermingle \string\selectlanguage\space and\%
5127 \string\babelpatterns\space or some patterns will not\%
5128 be taken into account. Reported}%
5129 \fi
5130 \ifx\@empty#1%
5131 \protected@edef\bbl@patterns@{\bbl@patterns@space#2}%
5132 \else
5133 \edef\bbl@tempb{\zap@space#1 \@empty}%
5134 \bbl@for\bbl@tempa\bbl@tempb{%
5135 \bbl@fixname\bbl@tempa
5136 \bbl@iflanguage\bbl@tempa{%
5137 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5138 \@ifundefined{\bbl@patterns@\bbl@tempa}%
5139 \empty
5140 {\csname \bbl@patterns@\bbl@tempa\endcsname\space}%
5141 #2}}}%
5142 \fi}}

```

12.4 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.

```

5143 % TODO - to a lua file
5144 \directlua{
5145 Babel = Babel or {}
5146 Babel.linebreaking = Babel.linebreaking or {}
5147 Babel.linebreaking.before = {}
5148 Babel.linebreaking.after = {}
5149 Babel.locale = {} % Free to use, indexed by \localeid
5150 function Babel.linebreaking.add_before(func)
5151 tex.print([[noexpand\csname \bbl@luahyphenate\endcsname]])
5152 table.insert(Babel.linebreaking.before, func)
5153 end
5154 function Babel.linebreaking.add_after(func)
5155 tex.print([[noexpand\csname \bbl@luahyphenate\endcsname]])
5156 table.insert(Babel.linebreaking.after, func)
5157 end
5158 }
5159 \def\bbl@intraspace#1 #2 #3\@{
5160 \directlua{
5161 Babel = Babel or {}
5162 Babel.intraspaces = Babel.intraspaces or {}
5163 Babel.intraspaces['\csname \bbl@sbcp@language\endcsname'] = %
5164 {b = #1, p = #2, m = #3}
5165 Babel.locale_props[\the\localeid].intraspace = %

```

```

5166      {b = #1, p = #2, m = #3}
5167   }}
5168 \def\bbl@intrapenalty#1\@{
5169   \directlua{
5170     Babel = Babel or {}
5171     Babel.intrapenalties = Babel.intrapenalties or {}
5172     Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5173     Babel.locale_props[\the\localeid].intrapenalty = #1
5174   }}
5175 \begingroup
5176 \catcode`\%=12
5177 \catcode`\^=14
5178 \catcode`\'=12
5179 \catcode`\~=12
5180 \gdef\bbl@seaintraspace{
5181   \let\bbl@seaintraspace\relax
5182   \directlua{
5183     Babel = Babel or {}
5184     Babel.sea_enabled = true
5185     Babel.sea_ranges = Babel.sea_ranges or {}
5186     function Babel.set_chranges (script, chrng)
5187       local c = 0
5188       for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5189         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5190         c = c + 1
5191       end
5192     end
5193     function Babel.sea_disc_to_space (head)
5194       local sea_ranges = Babel.sea_ranges
5195       local last_char = nil
5196       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5197       for item in node.traverse(head) do
5198         local i = item.id
5199         if i == node.id'glyph' then
5200           last_char = item
5201         elseif i == 7 and item.subtype == 3 and last_char
5202           and last_char.char > 0xC99 then
5203           quad = font.getfont(last_char.font).size
5204           for lg, rg in pairs(sea_ranges) do
5205             if last_char.char > rg[1] and last_char.char < rg[2] then
5206               lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5207               local intraspace = Babel.intraspaces[lg]
5208               local intrapenalty = Babel.intrapenalties[lg]
5209               local n
5210               if intrapenalty ~= 0 then
5211                 n = node.new(14, 0) ^% penalty
5212                 n.penalty = intrapenalty
5213                 node.insert_before(head, item, n)
5214               end
5215               n = node.new(12, 13) ^% (glue, spaceskip)
5216               node.setglue(n, intraspace.b * quad,
5217                 intraspace.p * quad,
5218                 intraspace.m * quad)
5219               node.insert_before(head, item, n)
5220               node.remove(head, item)
5221             end
5222           end
5223         end
5224       end
5225     end
5226   }^^
5227   \bbl@luahyphenate}

```


12.5 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.

```
5228 \catcode`\%=14
5229 \gdef\bbl@cjkintraspacespace{%
5230   \let\bbl@cjkintraspacespace\relax
5231   \directlua{
5232     Babel = Babel or {}
5233     require('babel-data-cjk.lua')
5234     Babel.cjk_enabled = true
5235     function Babel.cjk_linebreak(head)
5236       local GLYPH = node.id'glyph'
5237       local last_char = nil
5238       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5239       local last_class = nil
5240       local last_lang = nil
5241
5242       for item in node.traverse(head) do
5243         if item.id == GLYPH then
5244
5245           local lang = item.lang
5246
5247           local LOCALE = node.get_attribute(item,
5248             Babel.attr_locale)
5249           local props = Babel.locale_props[LOCALE]
5250
5251           local class = Babel.cjk_class[item.char].c
5252
5253           if props.cjk_quotes and props.cjk_quotes[item.char] then
5254             class = props.cjk_quotes[item.char]
5255           end
5256
5257           if class == 'cp' then class = 'cl' end % ]] as CL
5258           if class == 'id' then class = 'I' end
5259
5260           local br = 0
5261           if class and last_class and Babel.cjk_breaks[last_class][class] then
5262             br = Babel.cjk_breaks[last_class][class]
5263           end
5264
5265           if br == 1 and props.linebreak == 'c' and
5266             lang ~= \the\l@nohyphenation\space and
5267             last_lang ~= \the\l@nohyphenation then
5268             local intrapenalty = props.intrapenalty
5269             if intrapenalty ~= 0 then
5270               local n = node.new(14, 0)      % penalty
5271               n.penalty = intrapenalty
5272               node.insert_before(head, item, n)
5273             end
5274             local intraspacespace = props.intraspacespace
5275             local n = node.new(12, 13)      % (glue, spaceskip)
5276             node.setglue(n, intraspacespace.b * quad,
5277               intraspacespace.p * quad,
5278               intraspacespace.m * quad)
5279             node.insert_before(head, item, n)
5280           end
5281
5282           if font.getfont(item.font) then
```

```

5283         quad = font.getfont(item.font).size
5284     end
5285     last_class = class
5286     last_lang = lang
5287     else % if penalty, glue or anything else
5288         last_class = nil
5289     end
5290 end
5291 lang.hyphenate(head)
5292 end
5293 }%
5294 \bbl@luahyphenate}
5295 \gdef\bbl@luahyphenate{%
5296 \let\bbl@luahyphenate\relax
5297 \directlua{
5298     luatexbase.add_to_callback('hyphenate',
5299     function (head, tail)
5300         if Babel.linebreaking.before then
5301             for k, func in ipairs(Babel.linebreaking.before) do
5302                 func(head)
5303             end
5304         end
5305         if Babel.cjk_enabled then
5306             Babel.cjk_linebreak(head)
5307         end
5308         lang.hyphenate(head)
5309         if Babel.linebreaking.after then
5310             for k, func in ipairs(Babel.linebreaking.after) do
5311                 func(head)
5312             end
5313         end
5314         if Babel.sea_enabled then
5315             Babel.sea_disc_to_space(head)
5316         end
5317     end,
5318     'Babel.hyphenate')
5319 }
5320 }
5321 \endgroup
5322 \def\bbl@provide@intraspace{%
5323 \bbl@ifunset{bbl@intsp@{language}}{%
5324     {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5325         \bbl@xin@{/c}{\bbl@cl{lnbrk}}%
5326         \ifin@ % cjk
5327         \bbl@cjk@intraspace
5328         \directlua{
5329             Babel = Babel or {}
5330             Babel.locale_props = Babel.locale_props or {}
5331             Babel.locale_props[\the\localeid].linebreak = 'c'
5332         }%
5333         \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5334         \ifx\bbl@KVP@intrapenalty\@nnil
5335             \bbl@intrapenalty0\@@
5336         \fi
5337     \else % sea
5338         \bbl@sea@intraspace
5339         \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5340         \directlua{
5341             Babel = Babel or {}
5342             Babel.sea_ranges = Babel.sea_ranges or {}
5343             Babel.set_chranges('\bbl@cl{sbcpr}',
5344                             '\bbl@cl{chrng}')
5345         }%

```

```

5346      \ifx\bbl@KVP@intrapenalty\@nnil
5347      \bbl@intrapenalty0\@@
5348      \fi
5349      \fi
5350      \fi
5351      \ifx\bbl@KVP@intrapenalty\@nnil\else
5352      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5353      \fi}}

```

12.6 Arabic justification

```

5354 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5355 \def\bblar@chars{%
5356   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5357   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5358   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5359 \def\bblar@elongated{%
5360   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5361   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5362   0649,064A}
5363 \begingroup
5364   \catcode`\_ =11 \catcode`\:=11
5365   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5366 \endgroup
5367 \gdef\bbl@arabicjust{%
5368   \let\bbl@arabicjust\relax
5369   \newattribute\bblar@kashida
5370   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5371   \bblar@kashida=\z@
5372   \bbl@patchfont{{\bbl@parsejalt}}%
5373   \directlua{
5374     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5375     Babel.arabic.elong_map[\the\localeid] = {}
5376     luatexbase.add_to_callback('post_linebreak_filter',
5377       Babel.arabic.justify, 'Babel.arabic.justify')
5378     luatexbase.add_to_callback('hpack_filter',
5379       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5380   }}%
5381 % Save both node lists to make replacement. TODO. Save also widths to
5382 % make computations
5383 \def\bblar@fetchjalt#1#2#3#4{%
5384   \bbl@exp{\bbl@foreach{#1}}{%
5385     \bbl@ifunset{bblar@JE##1}%
5386     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5387     {\setbox\z@\hbox{^^^200d\char"\@nameuse{bblar@JE##1}#2}}%
5388   \directlua{%
5389     local last = nil
5390     for item in node.traverse(tex.box[0].head) do
5391       if item.id == node.id'glyph' and item.char > 0x600 and
5392       not (item.char == 0x200D) then
5393         last = item
5394       end
5395     end
5396     Babel.arabic.#3['##1#4'] = last.char
5397   }}
5398 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5399 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5400 % positioning?
5401 \gdef\bbl@parsejalt{%
5402   \ifx\addfontfeature\@undefined\else
5403     \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
5404     \ifin@
5405       \directlua{%

```

```

5406         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5407             Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5408             tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5409         end
5410     }%
5411     \fi
5412 \fi}
5413 \gdef\bbl@parsejalti{%
5414     \begingroup
5415     \let\bbl@parsejalt\relax      % To avoid infinite loop
5416     \edef\bbl@tempb{\fontid\font}%
5417     \bblar@nofswarn
5418     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5419     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5420     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5421     \addfontfeature{RawFeature+=jalt}%
5422     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5423     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5424     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5425     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5426     \directlua{%
5427         for k, v in pairs(Babel.arabic.from) do
5428             if Babel.arabic.dest[k] and
5429                 not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5430                 Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5431                     [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5432             end
5433         end
5434     }%
5435     \endgroup}
5436 %
5437 \begingroup
5438 \catcode`#=11
5439 \catcode`~=11
5440 \directlua{
5441
5442 Babel.arabic = Babel.arabic or {}
5443 Babel.arabic.from = {}
5444 Babel.arabic.dest = {}
5445 Babel.arabic.justify_factor = 0.95
5446 Babel.arabic.justify_enabled = true
5447
5448 function Babel.arabic.justify(head)
5449     if not Babel.arabic.justify_enabled then return head end
5450     for line in node.traverse_id(node.id'hlist', head) do
5451         Babel.arabic.justify_hlist(head, line)
5452     end
5453     return head
5454 end
5455
5456 function Babel.arabic.justify_hbox(head, gc, size, pack)
5457     local has_inf = false
5458     if Babel.arabic.justify_enabled and pack == 'exactly' then
5459         for n in node.traverse_id(12, head) do
5460             if n.stretch_order > 0 then has_inf = true end
5461         end
5462         if not has_inf then
5463             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5464         end
5465     end
5466     return head
5467 end
5468

```

```

5469 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5470     local d, new
5471     local k_list, k_item, pos_inline
5472     local width, width_new, full, k_curr, wt_pos, goal, shift
5473     local subst_done = false
5474     local elong_map = Babel.arabic.elong_map
5475     local last_line
5476     local GLYPH = node.id'glyph'
5477     local KASHIDA = Babel.attr_kashida
5478     local LOCALE = Babel.attr_locale
5479
5480     if line == nil then
5481         line = {}
5482         line.glue_sign = 1
5483         line.glue_order = 0
5484         line.head = head
5485         line.shift = 0
5486         line.width = size
5487     end
5488
5489     % Exclude last line. todo. But-- it discards one-word lines, too!
5490     % ? Look for glue = 12:15
5491     if (line.glue_sign == 1 and line.glue_order == 0) then
5492         elongs = {}      % Stores elongated candidates of each line
5493         k_list = {}      % And all letters with kashida
5494         pos_inline = 0   % Not yet used
5495
5496         for n in node.traverse_id(GLYPH, line.head) do
5497             pos_inline = pos_inline + 1 % To find where it is. Not used.
5498
5499             % Elongated glyphs
5500             if elong_map then
5501                 local locale = node.get_attribute(n, LOCALE)
5502                 if elong_map[locale] and elong_map[locale][n.font] and
5503                     elong_map[locale][n.font][n.char] then
5504                     table.insert(elongs, {node = n, locale = locale} )
5505                     node.set_attribute(n.prev, KASHIDA, 0)
5506                 end
5507             end
5508
5509             % Tatwil
5510             if Babel.kashida_wts then
5511                 local k_wt = node.get_attribute(n, KASHIDA)
5512                 if k_wt > 0 then % todo. parameter for multi inserts
5513                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5514                 end
5515             end
5516
5517         end % of node.traverse_id
5518
5519         if #elongs == 0 and #k_list == 0 then goto next_line end
5520         full = line.width
5521         shift = line.shift
5522         goal = full * Babel.arabic.justify_factor % A bit crude
5523         width = node.dimensions(line.head) % The 'natural' width
5524
5525         % == Elongated ==
5526         % Original idea taken from 'chickenize'
5527         while (#elongs > 0 and width < goal) do
5528             subst_done = true
5529             local x = #elongs
5530             local curr = elongs[x].node
5531             local oldchar = curr.char

```

```

5532     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5533     width = node.dimensions(line.head) % Check if the line is too wide
5534     % Substitute back if the line would be too wide and break:
5535     if width > goal then
5536         curr.char = oldchar
5537         break
5538     end
5539     % If continue, pop the just substituted node from the list:
5540     table.remove(elongs, x)
5541 end
5542
5543 % == Tatwil ==
5544 if #k_list == 0 then goto next_line end
5545
5546 width = node.dimensions(line.head) % The 'natural' width
5547 k_curr = #k_list
5548 wt_pos = 1
5549
5550 while width < goal do
5551     subst_done = true
5552     k_item = k_list[k_curr].node
5553     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5554         d = node.copy(k_item)
5555         d.char = 0x0640
5556         line.head, new = node.insert_after(line.head, k_item, d)
5557         width_new = node.dimensions(line.head)
5558         if width > goal or width == width_new then
5559             node.remove(line.head, new) % Better compute before
5560             break
5561         end
5562         width = width_new
5563     end
5564     if k_curr == 1 then
5565         k_curr = #k_list
5566         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5567     else
5568         k_curr = k_curr - 1
5569     end
5570 end
5571
5572 ::next_line::
5573
5574 % Must take into account marks and ins, see luatex manual.
5575 % Have to be executed only if there are changes. Investigate
5576 % what's going on exactly.
5577 if subst_done and not gc then
5578     d = node.hpack(line.head, full, 'exactly')
5579     d.shift = shift
5580     node.insert_before(head, line, d)
5581     node.remove(head, line)
5582 end
5583 end % if process line
5584 end
5585 }
5586 \endgroup
5587 \fi\fi % Arabic just block

```

12.7 Common stuff

```

5588 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5589 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5590 \DisableBabelHook{babel-fontspec}
5591 <<Font selection>>

```

12.8 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 form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5592 % TODO - to a lua file
5593 \directlua{
5594 Babel.script_blocks = {
5595   ['dflt'] = {},
5596   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5597               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5598   ['Armn'] = {{0x0530, 0x058F}},
5599   ['Beng'] = {{0x0980, 0x09FF}},
5600   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5601   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5602   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5603               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5604   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5605   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5606               {0xAB00, 0xAB2F}},
5607   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5608   % Don't follow strictly Unicode, which places some Coptic letters in
5609   % the 'Greek and Coptic' block
5610   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5611   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5612               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5613               {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5614               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5615               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5616               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5617   ['Hebr'] = {{0x0590, 0x05FF}},
5618   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5619               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5620   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5621   ['Knda'] = {{0x0C80, 0x0CFF}},
5622   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5623               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5624               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5625   ['Laoo'] = {{0x0E80, 0x0EFF}},
5626   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5627               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5628               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5629   ['Mahj'] = {{0x11150, 0x1117F}},
5630   ['Mlym'] = {{0x0D00, 0x0D7F}},
5631   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5632   ['Orya'] = {{0x0B00, 0x0B7F}},
5633   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5634   ['Syr1'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5635   ['Taml'] = {{0x0B80, 0x0BFF}},
5636   ['Telu'] = {{0x0C00, 0x0C7F}},
5637   ['Tfng'] = {{0x2D30, 0x2D7F}},
5638   ['Thai'] = {{0x0E00, 0x0E7F}},
5639   ['Tibt'] = {{0x0F00, 0x0FFF}},
5640   ['Vaii'] = {{0xA500, 0xA63F}},
5641   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5642 }
5643
5644 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyr1
5645 Babel.script_blocks.Hant = Babel.script_blocks.Hans
```

```

5646 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5647
5648 function Babel.locale_map(head)
5649   if not Babel.locale_mapped then return head end
5650
5651   local LOCALE = Babel.attr_locale
5652   local GLYPH = node.id('glyph')
5653   local inmath = false
5654   local toloc_save
5655   for item in node.traverse(head) do
5656     local toloc
5657     if not inmath and item.id == GLYPH then
5658       % Optimization: build a table with the chars found
5659       if Babel.chr_to_loc[item.char] then
5660         toloc = Babel.chr_to_loc[item.char]
5661       else
5662         for lc, maps in pairs(Babel.loc_to_scr) do
5663           for _, rg in pairs(maps) do
5664             if item.char >= rg[1] and item.char <= rg[2] then
5665               Babel.chr_to_loc[item.char] = lc
5666               toloc = lc
5667               break
5668             end
5669           end
5670         end
5671       end
5672       % Now, take action, but treat composite chars in a different
5673       % fashion, because they 'inherit' the previous locale. Not yet
5674       % optimized.
5675       if not toloc and
5676         (item.char >= 0x0300 and item.char <= 0x036F) or
5677         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5678         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5679         toloc = toloc_save
5680       end
5681       if toloc and toloc > -1 then
5682         if Babel.locale_props[toloc].lg then
5683           item.lang = Babel.locale_props[toloc].lg
5684           node.set_attribute(item, LOCALE, toloc)
5685         end
5686         if Babel.locale_props[toloc]['/'..item.font] then
5687           item.font = Babel.locale_props[toloc]['/'..item.font]
5688         end
5689         toloc_save = toloc
5690       end
5691     elseif not inmath and item.id == 7 then
5692       item.replace = item.replace and Babel.locale_map(item.replace)
5693       item.pre      = item.pre and Babel.locale_map(item.pre)
5694       item.post      = item.post and Babel.locale_map(item.post)
5695     elseif item.id == node.id'math' then
5696       inmath = (item.subtype == 0)
5697     end
5698   end
5699   return head
5700 end
5701 }

```

The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.

```

5702 \newcommand\babelcharproperty[1]{%
5703   \count@=#1\relax
5704   \ifvmode
5705     \expandafter\bbl@chprop

```



```

5706 \else
5707 \bbl@error{\string\babelcharproperty\space can be used only in\\%
5708             vertical mode (preamble or between paragraphs)}%
5709             {See the manual for futher info}%
5710 \fi}
5711 \newcommand\bbl@chprop[3][\the\count@]{%
5712 \tempcnta=#1\relax
5713 \bbl@ifunset{\bbl@chprop@#2}%
5714 {\bbl@error{No property named '#2'. Allowed values are\\%
5715             direction (bc), mirror (bmg), and linebreak (lb)}%
5716             {See the manual for futher info}}%
5717 {}%
5718 \loop
5719 \bbl@cs{chprop@#2}{#3}%
5720 \ifnum\count@<\tempcnta
5721 \advance\count@\@ne
5722 \repeat}
5723 \def\bbl@chprop@direction#1{%
5724 \directlua{
5725 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5726 Babel.characters[\the\count@]['d'] = '#1'
5727 }}
5728 \let\bbl@chprop@bc\bbl@chprop@direction
5729 \def\bbl@chprop@mirror#1{%
5730 \directlua{
5731 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5732 Babel.characters[\the\count@]['m'] = '\number#1'
5733 }}
5734 \let\bbl@chprop@bmg\bbl@chprop@mirror
5735 \def\bbl@chprop@linebreak#1{%
5736 \directlua{
5737 Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5738 Babel.cjk_characters[\the\count@]['c'] = '#1'
5739 }}
5740 \let\bbl@chprop@lb\bbl@chprop@linebreak
5741 \def\bbl@chprop@locale#1{%
5742 \directlua{
5743 Babel.chr_to_loc = Babel.chr_to_loc or {}
5744 Babel.chr_to_loc[\the\count@] =
5745 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5746 }}

```

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.

```

5747 \directlua{
5748 Babel.nohyphenation = \the\l@nohyphenation
5749 }

```

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 $\text{function}(m)$ $\text{return } m[1]..m[1]..'-'$ end, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m)$ $\text{return Babel.capt_map}(m[1],1)$ 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).

```

5750 \begingroup
5751 \catcode`\~ = 12
5752 \catcode`\% = 12
5753 \catcode`\& = 14
5754 \catcode`\| = 12
5755 \gdef\babelprehyphenation{%%
5756 \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{]]}

```

```

5757 \gdef\babelposthyphenation{&&
5758   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
5759 \gdef\bbl@settransform#1[#2]#3#4#5{&&
5760   \ifcase#1
5761     \bbl@activateprehyphen
5762   \else
5763     \bbl@activateposthyphen
5764   \fi
5765   \begingroup
5766     \def\babeltempa{\bbl@add@list\babeltempb}&&
5767     \let\babeltempb\@empty
5768     \def\bbl@tempa{#5}&&
5769     \bbl@replace\bbl@tempa{,}{ ,}&& TODO. Ugly trick to preserve {}
5770     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&&
5771       \bbl@ifsamestring{##1}{remove}&&
5772       {\bbl@add@list\babeltempb{nil}}&&
5773       {\directlua{
5774         local rep = [=[#1]=]
5775         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5776         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5777         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5778         if #1 == 0 then
5779           rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5780             'space = {' .. '%2, %3, %4' .. '}')
5781           rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5782             'spacefactor = {' .. '%2, %3, %4' .. '}')
5783           rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5784         else
5785           rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5786           rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5787           rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5788         end
5789         tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
5790       }}&&
5791   \let\bbl@kv@attribute\relax
5792   \let\bbl@kv@label\relax
5793   \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&&
5794   \ifx\bbl@kv@attribute\relax\else
5795     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&&
5796   \fi
5797   \directlua{
5798     local lbr = Babel.linebreaking.replacements[#1]
5799     local u = unicode.utf8
5800     local id, attr, label
5801     if #1 == 0 then
5802       id = \the\csname bbl@id@@#3\endcsname\space
5803     else
5804       id = \the\csname l@#3\endcsname\space
5805     end
5806     \ifx\bbl@kv@attribute\relax
5807       attr = -1
5808     \else
5809       attr = luatexbase.registernumber'\bbl@kv@attribute'
5810     \fi
5811     \ifx\bbl@kv@label\relax\else && Same refs:
5812       label = [= [\bbl@kv@label]=]
5813     \fi
5814     && Convert pattern:
5815     local patt = string.gsub([=[#4]=], '%s', '')
5816     if #1 == 0 then
5817       patt = string.gsub(patt, '|', ' ')
5818     end
5819     if not u.find(patt, '()', nil, true) then

```

```

5820     patt = '()' .. patt .. '()'
5821 end
5822 if #1 == 1 then
5823     patt = string.gsub(patt, '%(%)^', '^()')
5824     patt = string.gsub(patt, '%$$(%)', '()$')
5825 end
5826 patt = u.gsub(patt, '{{(.)}}',
5827     function (n)
5828         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5829     end)
5830 patt = u.gsub(patt, '{{(%x%x%x%x+)%}}',
5831     function (n)
5832         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5833     end)
5834 lbkr[id] = lbkr[id] or {}
5835 table.insert(lbkr[id],
5836     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5837 }&%
5838 \endgroup}
5839 \endgroup
5840 \def\bbl@activateposthyphen{%
5841 \let\bbl@activateposthyphen\relax
5842 \directlua{
5843     require('babel-transforms.lua')
5844     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5845 }}
5846 \def\bbl@activateprehyphen{%
5847 \let\bbl@activateprehyphen\relax
5848 \directlua{
5849     require('babel-transforms.lua')
5850     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5851 }}

```

12.9 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.

```

5852 \def\bbl@activate@preotf{%
5853 \let\bbl@activate@preotf\relax % only once
5854 \directlua{
5855     Babel = Babel or {}
5856     %
5857     function Babel.pre_otfload_v(head)
5858         if Babel.numbers and Babel.digits_mapped then
5859             head = Babel.numbers(head)
5860         end
5861         if Babel.bidi_enabled then
5862             head = Babel.bidi(head, false, dir)
5863         end
5864         return head
5865     end
5866     %
5867     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5868         if Babel.numbers and Babel.digits_mapped then
5869             head = Babel.numbers(head)
5870         end
5871         if Babel.bidi_enabled then
5872             head = Babel.bidi(head, false, dir)
5873         end
5874         return head
5875     end
5876     %

```

```

5877   luatexbase.add_to_callback('pre_linebreak_filter',
5878       Babel.pre_otfload_v,
5879       'Babel.pre_otfload_v',
5880       luatexbase.priority_in_callback('pre_linebreak_filter',
5881           'luaotfload.node_processor') or nil)
5882   %
5883   luatexbase.add_to_callback('hpack_filter',
5884       Babel.pre_otfload_h,
5885       'Babel.pre_otfload_h',
5886       luatexbase.priority_in_callback('hpack_filter',
5887           'luaotfload.node_processor') or nil)
5888   }}

```

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`.

```

5889 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5890   \let\bbl@beforeforeign\leavevmode
5891   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5892   \RequirePackage{luatexbase}
5893   \bbl@activate@preotf
5894   \directlua{
5895       require('babel-data-bidi.lua')
5896       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5897         require('babel-bidi-basic.lua')
5898       \or
5899         require('babel-bidi-basic-r.lua')
5900       \fi}
5901   % TODO - to locale_props, not as separate attribute
5902   \newattribute\bbl@attr@dir
5903   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5904   % TODO. I don't like it, hackish:
5905   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5906   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5907 \fi\fi
5908 \chardef\bbl@thetextdir\z@
5909 \chardef\bbl@thepardir\z@
5910 \def\bbl@getluadir#1{%
5911   \directlua{
5912     if tex.#1dir == 'TLT' then
5913       tex.sprint('0')
5914     elseif tex.#1dir == 'TRT' then
5915       tex.sprint('1')
5916     end}}
5917 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5918   \ifcase#3\relax
5919     \ifcase\bbl@getluadir{#1}\relax\else
5920       #2 TLT\relax
5921     \fi
5922   \else
5923     \ifcase\bbl@getluadir{#1}\relax
5924       #2 TRT\relax
5925     \fi
5926   \fi}
5927 \def\bbl@thedir{0}
5928 \def\bbl@textdir#1{%
5929   \bbl@setluadir{text}\textdir{#1}%
5930   \chardef\bbl@thetextdir#1\relax
5931   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5932   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5933 \def\bbl@pardir#1{%
5934   \bbl@setluadir{par}\pardir{#1}%
5935   \chardef\bbl@thepardir#1\relax}

```

```

5936 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5937 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5938 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
5939 %
5940 \ifnum\bbl@bidimode>\z@
5941   \def\bbl@insidemath{0}%
5942   \def\bbl@everymath{\def\bbl@insidemath{1}}
5943   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5944   \frozen@everymath\expandafter{%
5945     \expandafter\bbl@everymath\the\frozen@everymath}
5946   \frozen@everydisplay\expandafter{%
5947     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
5948   \AtBeginDocument{
5949     \directlua{
5950       function Babel.math_box_dir(head)
5951         if not (token.get_macro('bbl@insidemath') == '0') then
5952           if Babel.hlist_has_bidi(head) then
5953             local d = node.new(node.id'dir')
5954             d.dir = '+TRT'
5955             node.insert_before(head, node.has_glyph(head), d)
5956             for item in node.traverse(head) do
5957               node.set_attribute(item,
5958                 Babel.attr_dir, token.get_macro('bbl@thedir'))
5959             end
5960           end
5961         end
5962         return head
5963       end
5964       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
5965         "Babel.math_box_dir", 0)
5966     }%
5967 \fi

```

12.10 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`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5968 \bbl@trace{Redefinitions for bidi layout}
5969 %
5970 <<{*More package options}>> ≡
5971 \chardef\bbl@eqnpos\z@
5972 \DeclareOption{leqn}{\chardef\bbl@eqnpos\@ne}
5973 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
5974 <</More package options>>
5975 %
5976 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
5977 \ifnum\bbl@bidimode>\z@
5978   \ifx\matheqdirmode\@undefined\else
5979     \matheqdirmode\@ne
5980   \fi
5981   \let\bbl@eqnodir\relax
5982   \def\bbl@eqdel{()}
5983   \def\bbl@eqnum{%

```

```

5984 {\normalfont\normalcolor
5985 \expandafter\@firstoftwo\bbl@eqdel
5986 \theequation
5987 \expandafter\@secondoftwo\bbl@eqdel}}
5988 \def\bbl@puteqno#1{\eqno\hbox{#1}}
5989 \def\bbl@putleqno#1{\leqno\hbox{#1}}
5990 \def\bbl@eqno@flip#1{%
5991 \ifdim\predisplaysize=-\maxdimen
5992 \eqno
5993 \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
5994 \else
5995 \leqno\hbox{#1}%
5996 \fi}
5997 \def\bbl@leqno@flip#1{%
5998 \ifdim\predisplaysize=-\maxdimen
5999 \leqno
6000 \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}}\hss}}%
6001 \else
6002 \eqno\hbox{#1}%
6003 \fi}
6004 \AtBeginDocument{%
6005 \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6006 \AddToHook{env/equation/begin}{%
6007 \ifnum\bbl@thetextdir>\z@
6008 \let\@eqnnum\bbl@eqnum
6009 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6010 \chardef\bbl@thetextdir\z@
6011 \bbl@add\normalfont{\bbl@eqnodir}%
6012 \ifcase\bbl@eqnpos
6013 \let\bbl@puteqno\bbl@eqno@flip
6014 \or
6015 \let\bbl@puteqno\bbl@leqno@flip
6016 \fi
6017 \fi}%
6018 \ifnum\bbl@eqnpos=\tw\else
6019 \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6020 \fi
6021 \AddToHook{env/eqnarray/begin}{%
6022 \ifnum\bbl@thetextdir>\z@
6023 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6024 \chardef\bbl@thetextdir\z@
6025 \bbl@add\normalfont{\bbl@eqnodir}%
6026 \ifnum\bbl@eqnpos=\@ne
6027 \def\@eqnnum{%
6028 \setbox\z@\hbox{\bbl@eqnum}%
6029 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6030 \else
6031 \let\@eqnnum\bbl@eqnum
6032 \fi
6033 \fi}
6034 % Hack. YA luatex bug?:
6035 \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$}$}%
6036 \else % amstex
6037 \ifx\bbl@noamsmath\@undefined
6038 \ifnum\bbl@eqnpos=\@ne
6039 \let\bbl@ams@lap\hbox
6040 \else
6041 \let\bbl@ams@lap\llap
6042 \fi
6043 \ExplSyntaxOn
6044 \bbl@sreplace\intertext@{\normalbaselines}%
6045 {\normalbaselines
6046 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%

```

```

6047 \ExplSyntaxOff
6048 \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6049 \ifx\bb1@ams@lap\hbox % leqno
6050 \def\bb1@ams@flip#1{%
6051 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6052 \else % eqno
6053 \def\bb1@ams@flip#1{%
6054 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6055 \fi
6056 \def\bb1@ams@preset#1{%
6057 \ifnum\bb1@thetextdir>\z@
6058 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6059 \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6060 \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6061 \fi}%
6062 \ifnum\bb1@eqnpos=\tw@ \else
6063 \def\bb1@ams@equation{%
6064 \ifnum\bb1@thetextdir>\z@
6065 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6066 \chardef\bb1@thetextdir\z@
6067 \bb1@add\normalfont{\bb1@eqnodir}%
6068 \ifcase\bb1@eqnpos
6069 \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6070 \or
6071 \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6072 \fi
6073 \fi}%
6074 \AddToHook{env/equation/begin}{\bb1@ams@equation}%
6075 \AddToHook{env/equation*/begin}{\bb1@ams@equation}%
6076 \fi
6077 \AddToHook{env/cases/begin}{\bb1@ams@preset\bb1@ams@lap}%
6078 \AddToHook{env/multline/begin}{\bb1@ams@preset\hbox}%
6079 \AddToHook{env/gather/begin}{\bb1@ams@preset\bb1@ams@lap}%
6080 \AddToHook{env/gather*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6081 \AddToHook{env/align/begin}{\bb1@ams@preset\bb1@ams@lap}%
6082 \AddToHook{env/align*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6083 \AddToHook{env/eqnalign/begin}{\bb1@ams@preset\hbox}%
6084 % Hackish, for proper alignment. Don't ask me why it works!:
6085 \bb1@exp{% Avoid a 'visible' conditional
6086 \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6087 \AddToHook{env/flalign/begin}{\bb1@ams@preset\hbox}%
6088 \AddToHook{env/split/before}{%
6089 \ifnum\bb1@thetextdir>\z@
6090 \bb1@ifsamestring\@currentenv{equation}%
6091 {\ifx\bb1@ams@lap\hbox % leqno
6092 \def\bb1@ams@flip#1{%
6093 \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6094 \else
6095 \def\bb1@ams@flip#1{%
6096 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6097 \fi}%
6098 }%
6099 \fi}%
6100 \fi
6101 \fi}
6102 \fi
6103 \ifx\bb1@opt@layout\@nnil\endinput\fi % if no layout
6104 \ifnum\bb1@bidimode>\z@
6105 \def\bb1@nextfake#1{% non-local changes, use always inside a group!
6106 \bb1@exp{%
6107 \def\\bb1@insidemath{0}%
6108 \mathdir\the\bodydir
6109 #1% Once entered in math, set boxes to restore values

```

```

6110 \<ifmode>%
6111 \everyvbox{%
6112 \the\everyvbox
6113 \bodydir\the\bodydir
6114 \mathdir\the\mathdir
6115 \everyhbox{\the\everyhbox}%
6116 \everyvbox{\the\everyvbox}}%
6117 \everyhbox{%
6118 \the\everyhbox
6119 \bodydir\the\bodydir
6120 \mathdir\the\mathdir
6121 \everyhbox{\the\everyhbox}%
6122 \everyvbox{\the\everyvbox}}%
6123 \<fi>}}%
6124 \def\@hangfrom#1{%
6125 \setbox\@tempboxa\hbox{\{#1\}}%
6126 \hangindent\wd\@tempboxa
6127 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
6128 \shapemode\@ne
6129 \fi
6130 \noindent\box\@tempboxa}
6131 \fi
6132 \IfBabelLayout{tabular}
6133 {\let\bbbl@OL@tabular\@tabular
6134 \bbbl@replace\@tabular{\$}\bbbl@nextfake$}%
6135 \let\bbbl@NL@tabular\@tabular
6136 \AtBeginDocument{%
6137 \ifx\bbbl@NL@tabular\@tabular\else
6138 \bbbl@replace\@tabular{\$}\bbbl@nextfake$}%
6139 \let\bbbl@NL@tabular\@tabular
6140 \fi}}
6141 {}
6142 \IfBabelLayout{lists}
6143 {\let\bbbl@OL@list\list
6144 \bbbl@sreplace\list{\parshape}\bbbl@listparshape}%
6145 \let\bbbl@NL@list\list
6146 \def\bbbl@listparshape#1#2#3{%
6147 \parshape #1 #2 #3 %
6148 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
6149 \shapemode\tw@
6150 \fi}}
6151 {}
6152 \IfBabelLayout{graphics}
6153 {\let\bbbl@pictresetdir\relax
6154 \def\bbbl@pictsetdir#1{%
6155 \ifcase\bbbl@thetextdir
6156 \let\bbbl@pictresetdir\relax
6157 \else
6158 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6159 \or\textdir TLT
6160 \else\bodydir TLT \textdir TLT
6161 \fi
6162 % \(\text|par)dir required in pgf:
6163 \def\bbbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6164 \fi}%
6165 \AddToHook{env/picture/begin}{\bbbl@pictsetdir\tw@}%
6166 \directlua{
6167 Babel.get_picture_dir = true
6168 Babel.picture_has_bidi = 0
6169 %
6170 function Babel.picture_dir (head)
6171 if not Babel.get_picture_dir then return head end
6172 if Babel.hlist_has_bidi(head) then

```



```

6173     Babel.picture_has_bidi = 1
6174     end
6175     return head
6176   end
6177   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6178     "Babel.picture_dir")
6179 }%
6180 \AtBeginDocument{%
6181   \long\def\put(#1,#2)#3{%
6182     \@killglue
6183     % Try:
6184     \ifx\bbbl@pictresetdir\relax
6185       \def\bbbl@tempc{0}%
6186     \else
6187       \directlua{
6188         Babel.get_picture_dir = true
6189         Babel.picture_has_bidi = 0
6190       }%
6191       \setbox\z@\hb@xt@\z@{%
6192         \@defaultunitsset\@tempdimc{#1}\unitlength
6193         \kern\@tempdimc
6194         #3\hss}% TODO: #3 executed twice (below). That's bad.
6195       \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6196     \fi
6197     % Do:
6198     \@defaultunitsset\@tempdimc{#2}\unitlength
6199     \raise\@tempdimc\hb@xt@\z@{%
6200       \@defaultunitsset\@tempdimc{#1}\unitlength
6201       \kern\@tempdimc
6202       {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6203     \ignorespaces}%
6204   \MakeRobust\put}%
6205 \AtBeginDocument
6206 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
6207 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6208   \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6209   \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6210   \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6211 \fi
6212 \ifx\tikzpicture\@undefined\else
6213   \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\z@}%
6214   \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6215   \bbbl@sreplace\tikz{\begingroup}{\begingroup\bbbl@pictsetdir\tw@}%
6216 \fi
6217 \ifx\tcolorbox\@undefined\else
6218   \def\tcb@drawing@env@begin{%
6219     \csname tcb@before@tcb@split@state\endcsname
6220     \bbbl@pictsetdir\tw@
6221     \begin{\kvtcb@graphenv}%
6222     \tcb@bbdraw%
6223     \tcb@apply@graph@patches
6224   }%
6225   \def\tcb@drawing@env@end{%
6226     \end{\kvtcb@graphenv}%
6227     \bbbl@pictresetdir
6228     \csname tcb@after@tcb@split@state\endcsname
6229   }%
6230 \fi
6231 }}
6232 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some

additional readjustments for bidi=default.

```

6233 \IfBabelLayout{counters}%
6234   {\let\bbl@OL@@textsuperscript\@textsuperscript
6235     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6236     \let\bbl@latinarabic=\@arabic
6237     \let\bbl@OL@@arabic\@arabic
6238     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
6239     \ifpackagewith{babel}{bidi=default}%
6240     {\let\bbl@asciroman=\@roman
6241       \let\bbl@OL@@roman\@roman
6242       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6243       \let\bbl@asciiRoman=\@Roman
6244       \let\bbl@OL@@roman\@Roman
6245       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6246       \let\bbl@OL@labelenumii\labelenumii
6247       \def\labelenumii{}\theenumii}%
6248       \let\bbl@OL@p@enumiii\p@enumiii
6249       \def\p@enumiii{\p@enumii}\theenumii{}\{\}\{\}}
6250 \langle Footnote changes \rangle
6251 \IfBabelLayout{footnotes}%
6252   {\let\bbl@OL@footnote\footnote
6253     \BabelFootnote\footnote\language{}{}\}%
6254     \BabelFootnote\localfootnote\language{}{}\}%
6255     \BabelFootnote\mainfootnote{}\{\}\{\}}
6256   {}

```

Some \TeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

6257 \IfBabelLayout{extras}%
6258   {\let\bbl@OL@underline\underline
6259     \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake\$@@underline}%
6260     \let\bbl@OL@LaTeX2e\LaTeX2e
6261     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6262       \if b\expandafter\@car\fi@series\@nil\boldmath\fi
6263       \babelsublr{%
6264         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6265   {}
6266 \langle /luatex \rangle

```

12.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

6267 \langle *transforms \rangle
6268 Babel.linebreaking.replacements = {}
6269 Babel.linebreaking.replacements[0] = {} -- pre
6270 Babel.linebreaking.replacements[1] = {} -- post
6271
6272 -- Discretionaries contain strings as nodes
6273 function Babel.str_to_nodes(fn, matches, base)
6274   local n, head, last
6275   if fn == nil then return nil end
6276   for s in string.utfvalues(fn(matches)) do

```

```

6277     if base.id == 7 then
6278         base = base.replace
6279     end
6280     n = node.copy(base)
6281     n.char = s
6282     if not head then
6283         head = n
6284     else
6285         last.next = n
6286     end
6287     last = n
6288 end
6289 return head
6290 end
6291
6292 Babel.fetch_subtext = {}
6293
6294 Babel.ignore_pre_char = function(node)
6295     return (node.lang == Babel.nohyphenation)
6296 end
6297
6298 -- Merging both functions doesn't seem feasible, because there are too
6299 -- many differences.
6300 Babel.fetch_subtext[0] = function(head)
6301     local word_string = ''
6302     local word_nodes = {}
6303     local lang
6304     local item = head
6305     local inmath = false
6306
6307     while item do
6308
6309         if item.id == 11 then
6310             inmath = (item.subtype == 0)
6311         end
6312
6313         if inmath then
6314             -- pass
6315
6316         elseif item.id == 29 then
6317             local locale = node.get_attribute(item, Babel.attr_locale)
6318
6319             if lang == locale or lang == nil then
6320                 lang = lang or locale
6321                 if Babel.ignore_pre_char(item) then
6322                     word_string = word_string .. Babel.us_char
6323                 else
6324                     word_string = word_string .. unicode.utf8.char(item.char)
6325                 end
6326                 word_nodes[#word_nodes+1] = item
6327             else
6328                 break
6329             end
6330
6331         elseif item.id == 12 and item.subtype == 13 then
6332             word_string = word_string .. ' '
6333             word_nodes[#word_nodes+1] = item
6334
6335         -- Ignore leading unrecognized nodes, too.
6336         elseif word_string ~= '' then
6337             word_string = word_string .. Babel.us_char
6338             word_nodes[#word_nodes+1] = item -- Will be ignored
6339         end

```

```

6340
6341     item = item.next
6342 end
6343
6344 -- Here and above we remove some trailing chars but not the
6345 -- corresponding nodes. But they aren't accessed.
6346 if word_string:sub(-1) == ' ' then
6347     word_string = word_string:sub(1,-2)
6348 end
6349 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6350 return word_string, word_nodes, item, lang
6351 end
6352
6353 Babel.fetch_subtext[1] = function(head)
6354     local word_string = ''
6355     local word_nodes = {}
6356     local lang
6357     local item = head
6358     local inmath = false
6359
6360     while item do
6361
6362         if item.id == 11 then
6363             inmath = (item.subtype == 0)
6364         end
6365
6366         if inmath then
6367             -- pass
6368
6369         elseif item.id == 29 then
6370             if item.lang == lang or lang == nil then
6371                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6372                     lang = lang or item.lang
6373                     word_string = word_string .. unicode.utf8.char(item.char)
6374                     word_nodes[#word_nodes+1] = item
6375                 end
6376             else
6377                 break
6378             end
6379
6380         elseif item.id == 7 and item.subtype == 2 then
6381             word_string = word_string .. '='
6382             word_nodes[#word_nodes+1] = item
6383
6384         elseif item.id == 7 and item.subtype == 3 then
6385             word_string = word_string .. '|'
6386             word_nodes[#word_nodes+1] = item
6387
6388         -- (1) Go to next word if nothing was found, and (2) implicitly
6389         -- remove leading USs.
6390         elseif word_string == '' then
6391             -- pass
6392
6393         -- This is the responsible for splitting by words.
6394         elseif (item.id == 12 and item.subtype == 13) then
6395             break
6396
6397         else
6398             word_string = word_string .. Babel.us_char
6399             word_nodes[#word_nodes+1] = item -- Will be ignored
6400         end
6401
6402         item = item.next

```

```

6403 end
6404
6405 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6406 return word_string, word_nodes, item, lang
6407 end
6408
6409 function Babel.pre_hyphenate_replace(head)
6410   Babel.hyphenate_replace(head, 0)
6411 end
6412
6413 function Babel.post_hyphenate_replace(head)
6414   Babel.hyphenate_replace(head, 1)
6415 end
6416
6417 Babel.us_char = string.char(31)
6418
6419 function Babel.hyphenate_replace(head, mode)
6420   local u = unicode.utf8
6421   local lbkr = Babel.linebreaking.replacements[mode]
6422
6423   local word_head = head
6424
6425   while true do -- for each subtext block
6426
6427     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6428
6429     if Babel.debug then
6430       print()
6431       print((mode == 0) and '@@@<' or '@@@>', w)
6432     end
6433
6434     if nw == nil and w == '' then break end
6435
6436     if not lang then goto next end
6437     if not lbkr[lang] then goto next end
6438
6439     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6440     -- loops are nested.
6441     for k=1, #lbkr[lang] do
6442       local p = lbkr[lang][k].pattern
6443       local r = lbkr[lang][k].replace
6444       local attr = lbkr[lang][k].attr or -1
6445
6446       if Babel.debug then
6447         print('*****', p, mode)
6448       end
6449
6450       -- This variable is set in some cases below to the first *byte*
6451       -- after the match, either as found by u.match (faster) or the
6452       -- computed position based on sc if w has changed.
6453       local last_match = 0
6454       local step = 0
6455
6456       -- For every match.
6457       while true do
6458         if Babel.debug then
6459           print('====')
6460         end
6461         local new -- used when inserting and removing nodes
6462
6463         local matches = { u.match(w, p, last_match) }
6464
6465         if #matches < 2 then break end

```

```

6466
6467 -- Get and remove empty captures (with ()'s, which return a
6468 -- number with the position), and keep actual captures
6469 -- (from (...)), if any, in matches.
6470 local first = table.remove(matches, 1)
6471 local last = table.remove(matches, #matches)
6472 -- Non re-fetched substrings may contain \31, which separates
6473 -- subsubstrings.
6474 if string.find(w:sub(first, last-1), Babel.us_char) then break end
6475
6476 local save_last = last -- with A()BC()D, points to D
6477
6478 -- Fix offsets, from bytes to unicode. Explained above.
6479 first = u.len(w:sub(1, first-1)) + 1
6480 last = u.len(w:sub(1, last-1)) -- now last points to C
6481
6482 -- This loop stores in a small table the nodes
6483 -- corresponding to the pattern. Used by 'data' to provide a
6484 -- predictable behavior with 'insert' (w_nodes is modified on
6485 -- the fly), and also access to 'remove'd nodes.
6486 local sc = first-1 -- Used below, too
6487 local data_nodes = {}
6488
6489 local enabled = true
6490 for q = 1, last-first+1 do
6491     data_nodes[q] = w_nodes[sc+q]
6492     if enabled
6493         and attr > -1
6494         and not node.has_attribute(data_nodes[q], attr)
6495     then
6496         enabled = false
6497     end
6498 end
6499
6500 -- This loop traverses the matched substring and takes the
6501 -- corresponding action stored in the replacement list.
6502 -- sc = the position in substr nodes / string
6503 -- rc = the replacement table index
6504 local rc = 0
6505
6506 while rc < last-first+1 do -- for each replacement
6507     if Babel.debug then
6508         print('.....', rc + 1)
6509     end
6510     sc = sc + 1
6511     rc = rc + 1
6512
6513     if Babel.debug then
6514         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6515         local ss = ''
6516         for itt in node.traverse(head) do
6517             if itt.id == 29 then
6518                 ss = ss .. unicode.utf8.char(itt.char)
6519             else
6520                 ss = ss .. '{' .. itt.id .. '}'
6521             end
6522         end
6523         print('*****', ss)
6524     end
6525
6526     local crep = r[rc]
6527     local item = w_nodes[sc]

```

```

6529     local item_base = item
6530     local placeholder = Babel.us_char
6531     local d
6532
6533     if crep and crep.data then
6534         item_base = data_nodes[crep.data]
6535     end
6536
6537     if crep then
6538         step = crep.step or 0
6539     end
6540
6541     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6542         last_match = save_last    -- Optimization
6543         goto next
6544
6545     elseif crep == nil or crep.remove then
6546         node.remove(head, item)
6547         table.remove(w_nodes, sc)
6548         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6549         sc = sc - 1    -- Nothing has been inserted.
6550         last_match = utf8.offset(w, sc+1+step)
6551         goto next
6552
6553     elseif crep and crep.kashida then -- Experimental
6554         node.set_attribute(item,
6555             Babel.attr_kashida,
6556             crep.kashida)
6557         last_match = utf8.offset(w, sc+1+step)
6558         goto next
6559
6560     elseif crep and crep.string then
6561         local str = crep.string(matches)
6562         if str == '' then -- Gather with nil
6563             node.remove(head, item)
6564             table.remove(w_nodes, sc)
6565             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6566             sc = sc - 1    -- Nothing has been inserted.
6567         else
6568             local loop_first = true
6569             for s in string.utfvalues(str) do
6570                 d = node.copy(item_base)
6571                 d.char = s
6572                 if loop_first then
6573                     loop_first = false
6574                     head, new = node.insert_before(head, item, d)
6575                     if sc == 1 then
6576                         word_head = head
6577                     end
6578                     w_nodes[sc] = d
6579                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6580                 else
6581                     sc = sc + 1
6582                     head, new = node.insert_before(head, item, d)
6583                     table.insert(w_nodes, sc, new)
6584                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6585                 end
6586                 if Babel.debug then
6587                     print('.....', 'str')
6588                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6589                 end
6590             end -- for
6591             node.remove(head, item)

```

```

6592         end -- if ''
6593         last_match = utf8.offset(w, sc+1+step)
6594         goto next
6595
6596     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6597         d = node.new(7, 0) -- (disc, discretionary)
6598         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6599         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6600         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6601         d.attr = item_base.attr
6602         if crep.pre == nil then -- TeXbook p96
6603             d.penalty = crep.penalty or tex.hyphenpenalty
6604         else
6605             d.penalty = crep.penalty or tex.exhyphenpenalty
6606         end
6607         placeholder = '|'
6608         head, new = node.insert_before(head, item, d)
6609
6610     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6611         -- ERROR
6612
6613     elseif crep and crep.penalty then
6614         d = node.new(14, 0) -- (penalty, userpenalty)
6615         d.attr = item_base.attr
6616         d.penalty = crep.penalty
6617         head, new = node.insert_before(head, item, d)
6618
6619     elseif crep and crep.space then
6620         -- 655360 = 10 pt = 10 * 65536 sp
6621         d = node.new(12, 13) -- (glue, spaceskip)
6622         local quad = font.getfont(item_base.font).size or 655360
6623         node.setglue(d, crep.space[1] * quad,
6624                      crep.space[2] * quad,
6625                      crep.space[3] * quad)
6626         if mode == 0 then
6627             placeholder = ' '
6628         end
6629         head, new = node.insert_before(head, item, d)
6630
6631     elseif crep and crep.spacefactor then
6632         d = node.new(12, 13) -- (glue, spaceskip)
6633         local base_font = font.getfont(item_base.font)
6634         node.setglue(d,
6635                     crep.spacefactor[1] * base_font.parameters['space'],
6636                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
6637                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
6638         if mode == 0 then
6639             placeholder = ' '
6640         end
6641         head, new = node.insert_before(head, item, d)
6642
6643     elseif mode == 0 and crep and crep.space then
6644         -- ERROR
6645
6646     end -- ie replacement cases
6647
6648     -- Shared by disc, space and penalty.
6649     if sc == 1 then
6650         word_head = head
6651     end
6652     if crep.insert then
6653         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6654         table.insert(w_nodes, sc, new)

```



```

6655         last = last + 1
6656     else
6657         w_nodes[sc] = d
6658         node.remove(head, item)
6659         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6660     end
6661
6662     last_match = utf8.offset(w, sc+1+step)
6663
6664     ::next::
6665
6666     end -- for each replacement
6667
6668     if Babel.debug then
6669         print('.....', '/')
6670         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6671     end
6672
6673     end -- for match
6674
6675     end -- for patterns
6676
6677     ::next::
6678     word_head = nw
6679 end -- for substring
6680 return head
6681 end
6682
6683 -- This table stores capture maps, numbered consecutively
6684 Babel.capture_maps = {}
6685
6686 -- The following functions belong to the next macro
6687 function Babel.capture_func(key, cap)
6688     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6689     local cnt
6690     local u = unicode.utf8
6691     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
6692     if cnt == 0 then
6693         ret = u.gsub(ret, '{{(%x%x%x%x)}}',
6694             function (n)
6695                 return u.char(tonumber(n, 16))
6696             end)
6697     end
6698     ret = ret:gsub("%[%[%]]%.", '')
6699     ret = ret:gsub("%.%[%[%]]%", '')
6700     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6701 end
6702
6703 function Babel.capt_map(from, mapno)
6704     return Babel.capture_maps[mapno][from] or from
6705 end
6706
6707 -- Handle the {n|abc|ABC} syntax in captures
6708 function Babel.capture_func_map(capno, from, to)
6709     local u = unicode.utf8
6710     from = u.gsub(from, '{{(%x%x%x%x)}}',
6711         function (n)
6712             return u.char(tonumber(n, 16))
6713         end)
6714     to = u.gsub(to, '{{(%x%x%x%x)}}',
6715         function (n)
6716             return u.char(tonumber(n, 16))
6717         end)

```

```

6718 local froms = {}
6719 for s in string.utfcharacters(from) do
6720     table.insert(froms, s)
6721 end
6722 local cnt = 1
6723 table.insert(Babel.capture_maps, {})
6724 local mlen = table.getn(Babel.capture_maps)
6725 for s in string.utfcharacters(to) do
6726     Babel.capture_maps[mlen][froms[cnt]] = s
6727     cnt = cnt + 1
6728 end
6729 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6730         (mlen) .. ").." .. "["
6731 end
6732
6733 -- Create/Extend reversed sorted list of kashida weights:
6734 function Babel.capture_kashida(key, wt)
6735     wt = tonumber(wt)
6736     if Babel.kashida_wts then
6737         for p, q in ipairs(Babel.kashida_wts) do
6738             if wt == q then
6739                 break
6740             elseif wt > q then
6741                 table.insert(Babel.kashida_wts, p, wt)
6742                 break
6743             elseif table.getn(Babel.kashida_wts) == p then
6744                 table.insert(Babel.kashida_wts, wt)
6745             end
6746         end
6747     else
6748         Babel.kashida_wts = { wt }
6749     end
6750     return 'kashida = ' .. wt
6751 end
6752 </transforms>

```

12.12 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):

Arrrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

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.

```

6753 <*basic-r>
6754 Babel = Babel or {}
6755
6756 Babel.bidi_enabled = true
6757
6758 require('babel-data-bidi.lua')
6759
6760 local characters = Babel.characters
6761 local ranges = Babel.ranges
6762
6763 local DIR = node.id("dir")
6764
6765 local function dir_mark(head, from, to, outer)
6766   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6767   local d = node.new(DIR)
6768   d.dir = '+' .. dir
6769   node.insert_before(head, from, d)
6770   d = node.new(DIR)
6771   d.dir = '-' .. dir
6772   node.insert_after(head, to, d)
6773 end
6774
6775 function Babel.bidi(head, ispar)
6776   local first_n, last_n          -- first and last char with nums
6777   local last_es                  -- an auxiliary 'last' used with nums
6778   local first_d, last_d          -- first and last char in L/R block
6779   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):

```

6780   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6781   local strong_lr = (strong == 'l') and 'l' or 'r'
6782   local outer = strong
6783
6784   local new_dir = false
6785   local first_dir = false
6786   local inmath = false
6787
6788   local last_lr
6789
6790   local type_n = ''
6791
6792   for item in node.traverse(head) do
6793
6794     -- three cases: glyph, dir, otherwise
6795     if item.id == node.id'glyph'
6796       or (item.id == 7 and item.subtype == 2) then
6797
6798       local itemchar
6799       if item.id == 7 and item.subtype == 2 then
6800         itemchar = item.replace.char

```

```

6801     else
6802         itemchar = item.char
6803     end
6804     local chardata = characters[itemchar]
6805     dir = chardata and chardata.d or nil
6806     if not dir then
6807         for nn, et in ipairs(ranges) do
6808             if itemchar < et[1] then
6809                 break
6810             elseif itemchar <= et[2] then
6811                 dir = et[3]
6812                 break
6813             end
6814         end
6815     end
6816     dir = dir or 'l'
6817     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).

```

6818     if new_dir then
6819         attr_dir = 0
6820         for at in node.traverse(item.attr) do
6821             if at.number == Babel.attr_dir then
6822                 attr_dir = at.value % 3
6823             end
6824         end
6825         if attr_dir == 1 then
6826             strong = 'r'
6827         elseif attr_dir == 2 then
6828             strong = 'al'
6829         else
6830             strong = 'l'
6831         end
6832         strong_lr = (strong == 'l') and 'l' or 'r'
6833         outer = strong_lr
6834         new_dir = false
6835     end
6836
6837     if dir == 'nsm' then dir = strong end -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

6838     dir_real = dir -- We need dir_real to set strong below
6839     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:

```

6840     if strong == 'al' then
6841         if dir == 'en' then dir = 'an' end -- W2
6842         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6843         strong_lr = 'r' -- W3
6844     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

6845     elseif item.id == node.id'dir' and not inmath then
6846         new_dir = true
6847         dir = nil
6848     elseif item.id == node.id'math' then
6849         inmath = (item.subtype == 0)
6850     else
6851         dir = nil -- Not a char
6852     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>.

```

6853   if dir == 'en' or dir == 'an' or dir == 'et' then
6854       if dir ~= 'et' then
6855           type_n = dir
6856       end
6857       first_n = first_n or item
6858       last_n = last_es or item
6859       last_es = nil
6860   elseif dir == 'es' and last_n then -- W3+W6
6861       last_es = item
6862   elseif dir == 'cs' then             -- it's right - do nothing
6863   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6864       if strong_lr == 'r' and type_n ~= '' then
6865           dir_mark(head, first_n, last_n, 'r')
6866       elseif strong_lr == 'l' and first_d and type_n == 'an' then
6867           dir_mark(head, first_n, last_n, 'r')
6868           dir_mark(head, first_d, last_d, outer)
6869           first_d, last_d = nil, nil
6870       elseif strong_lr == 'l' and type_n ~= '' then
6871           last_d = last_n
6872       end
6873       type_n = ''
6874       first_n, last_n = nil, nil
6875   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:

```

6876   if dir == 'l' or dir == 'r' then
6877       if dir ~= outer then
6878           first_d = first_d or item
6879           last_d = item
6880       elseif first_d and dir ~= strong_lr then
6881           dir_mark(head, first_d, last_d, outer)
6882           first_d, last_d = nil, nil
6883       end
6884   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.

```

6885   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6886       item.char = characters[item.char] and
6887           characters[item.char].m or item.char
6888   elseif (dir or new_dir) and last_lr ~= item then
6889       local mir = outer .. strong_lr .. (dir or outer)
6890       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6891           for ch in node.traverse(node.next(last_lr)) do
6892               if ch == item then break end
6893               if ch.id == node.id'glyph' and characters[ch.char] then
6894                   ch.char = characters[ch.char].m or ch.char
6895               end
6896           end
6897       end
6898   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).

```

6899   if dir == 'l' or dir == 'r' then
6900       last_lr = item
6901       strong = dir_real          -- Don't search back - best save now
6902       strong_lr = (strong == 'l') and 'l' or 'r'
6903   elseif new_dir then
6904       last_lr = nil
6905   end
6906 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

6907   if last_lr and outer == 'r' then
6908       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6909           if characters[ch.char] then
6910               ch.char = characters[ch.char].m or ch.char
6911           end
6912       end
6913   end
6914   if first_n then
6915       dir_mark(head, first_n, last_n, outer)
6916   end
6917   if first_d then
6918       dir_mark(head, first_d, last_d, outer)
6919   end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

6920   return node.prev(head) or head
6921 end
6922 </basic-r>

```

And here the Lua code for bidi=basic:

```

6923 <*basic>
6924 Babel = Babel or {}
6925
6926 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6927
6928 Babel.fontmap = Babel.fontmap or {}
6929 Babel.fontmap[0] = {}          -- l
6930 Babel.fontmap[1] = {}          -- r
6931 Babel.fontmap[2] = {}          -- al/an
6932
6933 Babel.bidi_enabled = true
6934 Babel.mirroring_enabled = true
6935
6936 require('babel-data-bidi.lua')
6937
6938 local characters = Babel.characters
6939 local ranges = Babel.ranges
6940
6941 local DIR = node.id('dir')
6942 local GLYPH = node.id('glyph')
6943
6944 local function insert_implicit(head, state, outer)
6945     local new_state = state
6946     if state.sim and state.eim and state.sim ~= state.eim then
6947         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6948         local d = node.new(DIR)
6949         d.dir = '+' .. dir
6950         node.insert_before(head, state.sim, d)
6951         local d = node.new(DIR)
6952         d.dir = '-' .. dir

```

```

6953     node.insert_after(head, state.eim, d)
6954 end
6955 new_state.sim, new_state.eim = nil, nil
6956 return head, new_state
6957 end
6958
6959 local function insert_numeric(head, state)
6960     local new
6961     local new_state = state
6962     if state.san and state.ean and state.san ~= state.ean then
6963         local d = node.new(DIR)
6964         d.dir = '+TLT'
6965         _, new = node.insert_before(head, state.san, d)
6966         if state.san == state.sim then state.sim = new end
6967         local d = node.new(DIR)
6968         d.dir = '-TLT'
6969         _, new = node.insert_after(head, state.ean, d)
6970         if state.ean == state.eim then state.eim = new end
6971     end
6972     new_state.san, new_state.ean = nil, nil
6973     return head, new_state
6974 end
6975
6976 -- TODO - \hbox with an explicit dir can lead to wrong results
6977 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6978 -- was made to improve the situation, but the problem is the 3-dir
6979 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6980 -- well.
6981
6982 function Babel.bidi(head, ispar, hdir)
6983     local d -- d is used mainly for computations in a loop
6984     local prev_d = ''
6985     local new_d = false
6986
6987     local nodes = {}
6988     local outer_first = nil
6989     local inmath = false
6990
6991     local glue_d = nil
6992     local glue_i = nil
6993
6994     local has_en = false
6995     local first_et = nil
6996
6997     local ATDIR = Babel.attr_dir
6998
6999     local save_outer
7000     local temp = node.get_attribute(head, ATDIR)
7001     if temp then
7002         temp = temp % 3
7003         save_outer = (temp == 0 and 'l') or
7004                     (temp == 1 and 'r') or
7005                     (temp == 2 and 'al')
7006     elseif ispar then -- Or error? Shouldn't happen
7007         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7008     else -- Or error? Shouldn't happen
7009         save_outer = ('TRT' == hdir) and 'r' or 'l'
7010     end
7011     -- when the callback is called, we are just _after_ the box,
7012     -- and the textdir is that of the surrounding text
7013     -- if not ispar and hdir ~= tex.textdir then
7014     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7015     -- end

```

```

7016 local outer = save_outer
7017 local last = outer
7018 -- 'al' is only taken into account in the first, current loop
7019 if save_outer == 'al' then save_outer = 'r' end
7020
7021 local fontmap = Babel.fontmap
7022
7023 for item in node.traverse(head) do
7024
7025     -- In what follows, #node is the last (previous) node, because the
7026     -- current one is not added until we start processing the neutrals.
7027
7028     -- three cases: glyph, dir, otherwise
7029     if item.id == GLYPH
7030         or (item.id == 7 and item.subtype == 2) then
7031
7032         local d_font = nil
7033         local item_r
7034         if item.id == 7 and item.subtype == 2 then
7035             item_r = item.replace    -- automatic discs have just 1 glyph
7036         else
7037             item_r = item
7038         end
7039         local chardata = characters[item_r.char]
7040         d = chardata and chardata.d or nil
7041         if not d or d == 'nsm' then
7042             for nn, et in ipairs(ranges) do
7043                 if item_r.char < et[1] then
7044                     break
7045                 elseif item_r.char <= et[2] then
7046                     if not d then d = et[3]
7047                     elseif d == 'nsm' then d_font = et[3]
7048                     end
7049                     break
7050                 end
7051             end
7052         end
7053         d = d or 'l'
7054
7055         -- A short 'pause' in bidi for mapfont
7056         d_font = d_font or d
7057         d_font = (d_font == 'l' and 0) or
7058             (d_font == 'nsm' and 0) or
7059             (d_font == 'r' and 1) or
7060             (d_font == 'al' and 2) or
7061             (d_font == 'an' and 2) or nil
7062         if d_font and fontmap and fontmap[d_font][item_r.font] then
7063             item_r.font = fontmap[d_font][item_r.font]
7064         end
7065
7066         if new_d then
7067             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7068             if inmath then
7069                 attr_d = 0
7070             else
7071                 attr_d = node.get_attribute(item, ATDIR)
7072                 attr_d = attr_d % 3
7073             end
7074             if attr_d == 1 then
7075                 outer_first = 'r'
7076                 last = 'r'
7077             elseif attr_d == 2 then
7078                 outer_first = 'r'

```



```

7079         last = 'al'
7080     else
7081         outer_first = 'l'
7082         last = 'l'
7083     end
7084     outer = last
7085     has_en = false
7086     first_et = nil
7087     new_d = false
7088 end
7089
7090 if glue_d then
7091     if (d == 'l' and 'l' or 'r') ~= glue_d then
7092         table.insert(nodes, {glue_i, 'on', nil})
7093     end
7094     glue_d = nil
7095     glue_i = nil
7096 end
7097
7098 elseif item.id == DIR then
7099     d = nil
7100     if head ~= item then new_d = true end
7101
7102 elseif item.id == node.id'glue' and item.subtype == 13 then
7103     glue_d = d
7104     glue_i = item
7105     d = nil
7106
7107 elseif item.id == node.id'math' then
7108     inmath = (item.subtype == 0)
7109
7110 else
7111     d = nil
7112 end
7113
7114 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7115 if last == 'al' and d == 'en' then
7116     d = 'an'          -- W3
7117 elseif last == 'al' and (d == 'et' or d == 'es') then
7118     d = 'on'          -- W6
7119 end
7120
7121 -- EN + CS/ES + EN      -- W4
7122 if d == 'en' and #nodes >= 2 then
7123     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7124         and nodes[#nodes-1][2] == 'en' then
7125         nodes[#nodes][2] = 'en'
7126     end
7127 end
7128
7129 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7130 if d == 'an' and #nodes >= 2 then
7131     if (nodes[#nodes][2] == 'cs')
7132         and nodes[#nodes-1][2] == 'an' then
7133         nodes[#nodes][2] = 'an'
7134     end
7135 end
7136
7137 -- ET/EN                  -- W5 + W7->l / W6->on
7138 if d == 'et' then
7139     first_et = first_et or (#nodes + 1)
7140 elseif d == 'en' then
7141     has_en = true

```

```

7142     first_et = first_et or (#nodes + 1)
7143 elseif first_et then      -- d may be nil here !
7144     if has_en then
7145         if last == 'l' then
7146             temp = 'l'      -- W7
7147         else
7148             temp = 'en'     -- W5
7149         end
7150     else
7151         temp = 'on'        -- W6
7152     end
7153     for e = first_et, #nodes do
7154         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7155     end
7156     first_et = nil
7157     has_en = false
7158 end
7159
7160 -- Force mathdir in math if ON (currently works as expected only
7161 -- with 'l')
7162 if inmath and d == 'on' then
7163     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7164 end
7165
7166 if d then
7167     if d == 'al' then
7168         d = 'r'
7169         last = 'al'
7170     elseif d == 'l' or d == 'r' then
7171         last = d
7172     end
7173     prev_d = d
7174     table.insert(nodes, {item, d, outer_first})
7175 end
7176
7177 outer_first = nil
7178
7179 end
7180
7181 -- TODO -- repeated here in case EN/ET is the last node. Find a
7182 -- better way of doing things:
7183 if first_et then      -- dir may be nil here !
7184     if has_en then
7185         if last == 'l' then
7186             temp = 'l'      -- W7
7187         else
7188             temp = 'en'     -- W5
7189         end
7190     else
7191         temp = 'on'        -- W6
7192     end
7193     for e = first_et, #nodes do
7194         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7195     end
7196 end
7197
7198 -- dummy node, to close things
7199 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7200
7201 ----- NEUTRAL -----
7202
7203 outer = save_outer
7204 last = outer

```

```

7205
7206 local first_on = nil
7207
7208 for q = 1, #nodes do
7209     local item
7210
7211     local outer_first = nodes[q][3]
7212     outer = outer_first or outer
7213     last = outer_first or last
7214
7215     local d = nodes[q][2]
7216     if d == 'an' or d == 'en' then d = 'r' end
7217     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7218
7219     if d == 'on' then
7220         first_on = first_on or q
7221     elseif first_on then
7222         if last == d then
7223             temp = d
7224         else
7225             temp = outer
7226         end
7227         for r = first_on, q - 1 do
7228             nodes[r][2] = temp
7229             item = nodes[r][1] -- MIRRORING
7230             if Babel.mirroring_enabled and item.id == GLYPH
7231                 and temp == 'r' and characters[item.char] then
7232                 local font_mode = ''
7233                 if font.fonts[item.font].properties then
7234                     font_mode = font.fonts[item.font].properties.mode
7235                 end
7236                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7237                     item.char = characters[item.char].m or item.char
7238                 end
7239             end
7240         end
7241         first_on = nil
7242     end
7243
7244     if d == 'r' or d == 'l' then last = d end
7245 end
7246
7247 ----- IMPLICIT, REORDER -----
7248
7249 outer = save_outer
7250 last = outer
7251
7252 local state = {}
7253 state.has_r = false
7254
7255 for q = 1, #nodes do
7256
7257     local item = nodes[q][1]
7258
7259     outer = nodes[q][3] or outer
7260
7261     local d = nodes[q][2]
7262
7263     if d == 'nsm' then d = last end -- W1
7264     if d == 'en' then d = 'an' end
7265     local isdir = (d == 'r' or d == 'l')
7266
7267     if outer == 'l' and d == 'an' then

```

```

7268     state.san = state.san or item
7269     state.ean = item
7270 elseif state.san then
7271     head, state = insert_numeric(head, state)
7272 end
7273
7274 if outer == 'l' then
7275     if d == 'an' or d == 'r' then      -- im -> implicit
7276         if d == 'r' then state.has_r = true end
7277         state.sim = state.sim or item
7278         state.eim = item
7279     elseif d == 'l' and state.sim and state.has_r then
7280         head, state = insert_implicit(head, state, outer)
7281     elseif d == 'l' then
7282         state.sim, state.eim, state.has_r = nil, nil, false
7283     end
7284 else
7285     if d == 'an' or d == 'l' then
7286         if nodes[q][3] then -- nil except after an explicit dir
7287             state.sim = item -- so we move sim 'inside' the group
7288         else
7289             state.sim = state.sim or item
7290         end
7291         state.eim = item
7292     elseif d == 'r' and state.sim then
7293         head, state = insert_implicit(head, state, outer)
7294     elseif d == 'r' then
7295         state.sim, state.eim = nil, nil
7296     end
7297 end
7298
7299 if isdir then
7300     last = d          -- Don't search back - best save now
7301 elseif d == 'on' and state.san then
7302     state.san = state.san or item
7303     state.ean = item
7304 end
7305
7306 end
7307
7308 return node.prev(head) or head
7309 end
7310 </basic>

```

13 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.

14 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.

```
7311 < *nil>
7312 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7313 \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.

```
7314 \ifx\l@nil\undefined
7315 \newlanguage\l@nil
7316 \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
7317 \let\bbl@elt\relax
7318 \edef\bbl@languages{% Add it to the list of languages
7319 \bbl@languages\bbl@elt{nil}{\the\l@nil}\{}}
7320 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
7321 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil
7322 \let\captionnil\empty
7323 \let\datenil\empty
```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```
7324 \def\bbl@inidata@nil{%
7325 \bbl@elt{identification}{tag.ini}{und}%
7326 \bbl@elt{identification}{load.level}{0}%
7327 \bbl@elt{identification}{charset}{utf8}%
7328 \bbl@elt{identification}{version}{1.0}%
7329 \bbl@elt{identification}{date}{2022-05-16}%
7330 \bbl@elt{identification}{name.local}{nil}%
7331 \bbl@elt{identification}{name.english}{nil}%
7332 \bbl@elt{identification}{name.babel}{nil}%
7333 \bbl@elt{identification}{tag.bcp47}{und}%
7334 \bbl@elt{identification}{language.tag.bcp47}{und}%
7335 \bbl@elt{identification}{tag.opentype}{dflt}%
7336 \bbl@elt{identification}{script.name}{Latin}%
7337 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7338 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7339 \bbl@elt{identification}{level}{1}%
7340 \bbl@elt{identification}{encodings}{}%
7341 \bbl@elt{identification}{derivate}{no}}
7342 \@namedef{bbl@tbc@nil}{und}
7343 \@namedef{bbl@lbc@nil}{und}
7344 \@namedef{bbl@lotf@nil}{dflt}
7345 \@namedef{bbl@elname@nil}{nil}
7346 \@namedef{bbl@lname@nil}{nil}
7347 \@namedef{bbl@esname@nil}{Latin}
7348 \@namedef{bbl@sname@nil}{Latin}
7349 \@namedef{bbl@sbc@nil}{Latn}
7350 \@namedef{bbl@sotf@nil}{Latn}
```

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.

```
7351 \ldf@finish{nil}
7352 </nil>
```

15 Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the identification section with `require.calendars`.

Start with function to compute the Julian day. It's based on the little library `calendar.js`, by John Walker, in the public domain.

```
7353 <<Compute Julian day>> ≡
7354 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7355 \def\bbl@cs@gregleap#1{%
7356   (\bbl@fpmo{#1}{4} == 0) &&
7357   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7358 \def\bbl@cs@jd#1#2#3{% year, month, day
7359   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7360     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7361     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
7362     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 }
7363 <</Compute Julian day>>
```

15.1 Islamic

The code for the Civil calendar is based on it, too.

```
7364 <*ca-islamic>
7365 \ExplSyntaxOn
7366 <<Compute Julian day>>
7367 % == islamic (default)
7368 % Not yet implemented
7369 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar:

```
7370 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7371   ((#3 + ceil(29.5 * (#2 - 1)) +
7372     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7373     1948439.5) - 1) }
7374 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+2}}
7375 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7376 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7377 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7378 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7379 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7380   \edef\bbl@tempa{%
7381     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 }%
7382     \edef#5{%
7383       \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7384     \edef#6{\fp_eval:n{
7385       min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7386     \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1 }}}
```

The Umm al-Qura calendar, used mainly in Saudi Arabia, is based on moment-hijri, by Abdullah Alsigar (license MIT).

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ~1435/~1460 (Gregorian ~2014/~2038).

```
7387 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7388 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7389 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7390 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7391 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7392 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7393 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7394 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7395 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7396 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7397 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7398 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
```

```

7399 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7400 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7401 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7402 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7403 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7404 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7405 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7406 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7407 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7408 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7409 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7410 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7411 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7412 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7413 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7414 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7415 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7416 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7417 65401,65431,65460,65490,65520}
7418 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7419 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7420 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7421 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7422   \ifnum#2>2014 \ifnum#2<2038
7423     \bbl@afterfi\expandafter\@gobble
7424   \fi\fi
7425   {\bbl@error{Year~out-of-range}{The~allowed~range-is~2014-2038}}%
7426   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7427     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7428   \count@\@ne
7429   \bbl@foreach\bbl@cs@umalqura@data{%
7430     \advance\count@\@ne
7431     \ifnum##1>\bbl@tempd\else
7432       \edef\bbl@tempe{\the\count@}%
7433       \edef\bbl@tempb{##1}%
7434     \fi}%
7435   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
7436   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7437   \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7438   \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7439   \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7440 \ExplSyntaxOff
7441 \bbl@add\bbl@precalendar{%
7442   \bbl@replace\bbl@ld@calendar{-civil}}}%
7443   \bbl@replace\bbl@ld@calendar{-umalqura}}}%
7444   \bbl@replace\bbl@ld@calendar{+}}}%
7445   \bbl@replace\bbl@ld@calendar{-}}}%
7446 \</ca-islamic>

```

16 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty

```

7447 <*ca-hebrew>
7448 \newcount\bbl@cntcommon
7449 \def\bbl@remainder#1#2#3{%
7450   #3=#1\relax
7451   \divide #3 by #2\relax
7452   \multiply #3 by -#2\relax
7453   \advance #3 by #1\relax}%
7454 \newif\ifbbl@divisible

```

```

7455 \def\bbl@checkifdivisible#1#2{%
7456   {\countdef\tmp=0
7457     \bbl@remainder{#1}{#2}{\tmp}%
7458     \ifnum \tmp=0
7459       \global\bbl@divisibletrue
7460     \else
7461       \global\bbl@divisiblefalse
7462     \fi}}
7463 \newif\ifbbl@gregleap
7464 \def\bbl@ifgregleap#1{%
7465   \bbl@checkifdivisible{#1}{4}%
7466   \ifbbl@divisible
7467     \bbl@checkifdivisible{#1}{100}%
7468     \ifbbl@divisible
7469       \bbl@checkifdivisible{#1}{400}%
7470       \ifbbl@divisible
7471         \bbl@gregleaptrue
7472       \else
7473         \bbl@gregleapfalse
7474       \fi
7475     \else
7476       \bbl@gregleaptrue
7477     \fi
7478   \else
7479     \bbl@gregleapfalse
7480   \fi
7481   \ifbbl@gregleap}
7482 \def\bbl@gregdayspriormonths#1#2#3{%
7483   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7484     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7485   \bbl@ifgregleap{#2}%
7486   \ifnum #1 > 2
7487     \advance #3 by 1
7488   \fi
7489   \fi
7490   \global\bbl@cntcommon=#3}%
7491   #3=\bbl@cntcommon}
7492 \def\bbl@gregdaysprioryears#1#2{%
7493   {\countdef\tmpc=4
7494     \countdef\tmpb=2
7495     \tmpb=#1\relax
7496     \advance \tmpb by -1
7497     \tmpc=\tmpb
7498     \multiply \tmpc by 365
7499     #2=\tmpc
7500     \tmpc=\tmpb
7501     \divide \tmpc by 4
7502     \advance #2 by \tmpc
7503     \tmpc=\tmpb
7504     \divide \tmpc by 100
7505     \advance #2 by -\tmpc
7506     \tmpc=\tmpb
7507     \divide \tmpc by 400
7508     \advance #2 by \tmpc
7509     \global\bbl@cntcommon=#2\relax}%
7510   #2=\bbl@cntcommon}
7511 \def\bbl@absfromgreg#1#2#3#4{%
7512   {\countdef\tmpd=0
7513     #4=#1\relax
7514     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7515     \advance #4 by \tmpd
7516     \bbl@gregdaysprioryears{#3}{\tmpd}%
7517     \advance #4 by \tmpd

```



```

7518 \global\bbl@cntcommon=#4\relax}%
7519 #4=\bbl@cntcommon}
7520 \newif\ifbbl@hebrleap
7521 \def\bbl@checkleaphebrewyear#1{%
7522 {\countdef\tmpa=0
7523 \countdef\tmpb=1
7524 \tmpa=#1\relax
7525 \multiply \tmpa by 7
7526 \advance \tmpa by 1
7527 \bbl@remainder{\tmpa}{19}{\tmpb}%
7528 \ifnum \tmpb < 7
7529 \global\bbl@hebrleaptrue
7530 \else
7531 \global\bbl@hebrleapfalse
7532 \fi}}
7533 \def\bbl@hebreleapsedmonths#1#2{%
7534 {\countdef\tmpa=0
7535 \countdef\tmpb=1
7536 \countdef\tmpc=2
7537 \tmpa=#1\relax
7538 \advance \tmpa by -1
7539 #2=\tmpa
7540 \divide #2 by 19
7541 \multiply #2 by 235
7542 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7543 \tmpc=\tmpb
7544 \multiply \tmpb by 12
7545 \advance #2 by \tmpb
7546 \multiply \tmpc by 7
7547 \advance \tmpc by 1
7548 \divide \tmpc by 19
7549 \advance #2 by \tmpc
7550 \global\bbl@cntcommon=#2}%
7551 #2=\bbl@cntcommon}
7552 \def\bbl@hebreleapseddays#1#2{%
7553 {\countdef\tmpa=0
7554 \countdef\tmpb=1
7555 \countdef\tmpc=2
7556 \bbl@hebreleapsedmonths{#1}{#2}%
7557 \tmpa=#2\relax
7558 \multiply \tmpa by 13753
7559 \advance \tmpa by 5604
7560 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7561 \divide \tmpa by 25920
7562 \multiply #2 by 29
7563 \advance #2 by 1
7564 \advance #2 by \tmpa
7565 \bbl@remainder{#2}{7}{\tmpa}%
7566 \ifnum \tmpc < 19440
7567 \ifnum \tmpc < 9924
7568 \else
7569 \ifnum \tmpa=2
7570 \bbl@checkleaphebrewyear{#1}% of a common year
7571 \ifbbl@hebrleap
7572 \else
7573 \advance #2 by 1
7574 \fi
7575 \fi
7576 \fi
7577 \ifnum \tmpc < 16789
7578 \else
7579 \ifnum \tmpa=1
7580 \advance #1 by -1

```

```

7581             \bbl@checkleaphebyear{#1}% at the end of leap year
7582             \ifbbl@hebrleap
7583                 \advance #2 by 1
7584             \fi
7585         \fi
7586     \fi
7587 \else
7588     \advance #2 by 1
7589 \fi
7590 \bbl@remainder{#2}{7}{\tmpa}%
7591 \ifnum \tmpa=0
7592     \advance #2 by 1
7593 \else
7594     \ifnum \tmpa=3
7595         \advance #2 by 1
7596     \else
7597         \ifnum \tmpa=5
7598             \advance #2 by 1
7599         \fi
7600     \fi
7601 \fi
7602 \global\bbl@cntcommon=#2\relax}%
7603 #2=\bbl@cntcommon}
7604 \def\bbl@daysinhebyear#1#2{%
7605     {\countdef\tmpe=12
7606     \bbl@hebreleapseddays{#1}{\tmpe}%
7607     \advance #1 by 1
7608     \bbl@hebreleapseddays{#1}{#2}%
7609     \advance #2 by -\tmpe
7610     \global\bbl@cntcommon=#2}%
7611     #2=\bbl@cntcommon}
7612 \def\bbl@hebrdayspriormonths#1#2#3{%
7613     {\countdef\tmpf= 14
7614     #3=\ifcase #1\relax
7615         0 \or
7616         0 \or
7617         30 \or
7618         59 \or
7619         89 \or
7620         118 \or
7621         148 \or
7622         148 \or
7623         177 \or
7624         207 \or
7625         236 \or
7626         266 \or
7627         295 \or
7628         325 \or
7629         400
7630     \fi
7631     \bbl@checkleaphebyear{#2}%
7632     \ifbbl@hebrleap
7633         \ifnum #1 > 6
7634             \advance #3 by 30
7635         \fi
7636     \fi
7637     \bbl@daysinhebyear{#2}{\tmpf}%
7638     \ifnum #1 > 3
7639         \ifnum \tmpf=353
7640             \advance #3 by -1
7641         \fi
7642         \ifnum \tmpf=383
7643             \advance #3 by -1

```

```

7644     \fi
7645 \fi
7646 \ifnum #1 > 2
7647     \ifnum \tmpf=355
7648         \advance #3 by 1
7649     \fi
7650     \ifnum \tmpf=385
7651         \advance #3 by 1
7652     \fi
7653 \fi
7654 \global\bbl@cntcommon=#3\relax}%
7655 #3=\bbl@cntcommon}
7656 \def\bbl@absfromhebr#1#2#3#4{%
7657     {#4=#1\relax
7658     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7659     \advance #4 by #1\relax
7660     \bbl@hebreleapseddays{#3}{#1}%
7661     \advance #4 by #1\relax
7662     \advance #4 by -1373429
7663     \global\bbl@cntcommon=#4\relax}%
7664 #4=\bbl@cntcommon}
7665 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7666     {\countdef\tmpx= 17
7667     \countdef\tmpy= 18
7668     \countdef\tmpz= 19
7669     #6=#3\relax
7670     \global\advance #6 by 3761
7671     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7672     \tmpz=1 \tmpy=1
7673     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7674     \ifnum \tmpx > #4\relax
7675         \global\advance #6 by -1
7676         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7677     \fi
7678     \advance #4 by -\tmpx
7679     \advance #4 by 1
7680     #5=#4\relax
7681     \divide #5 by 30
7682     \loop
7683         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7684         \ifnum \tmpx < #4\relax
7685             \advance #5 by 1
7686             \tmpy=\tmpx
7687         \repeat
7688     \global\advance #5 by -1
7689     \global\advance #4 by -\tmpy}}
7690 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
7691 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7692 \def\bbl@ca@hebrew#1-#2-#3@@#4#5#6{%
7693     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7694     \bbl@hebrfromgreg
7695         {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7696         {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
7697     \edef#4{\the\bbl@hebyear}%
7698     \edef#5{\the\bbl@hebrmonth}%
7699     \edef#6{\the\bbl@hebrday}}
7700 \</ca-hebrew>

```

17 Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use

with LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```

7701 <*ca-persian>
7702 \ExplSyntaxOn
7703 <<Compute Julian day>>
7704 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7705 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7706 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
7707 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
7708 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7709 \bbl@afterfi\expandafter\@gobble
7710 \fi\fi
7711 {\bbl@error{Year-out-of-range}{The~allowed-range-is~2013-2050}}%
7712 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7713 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7714 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7715 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7716 \ifnum\bbl@tempc<\bbl@tempb
7717 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7718 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7719 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7720 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7721 \fi
7722 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7723 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7724 \edef#5{\fp_eval:n{% set Jalali month
7725 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7726 \edef#6{\fp_eval:n{% set Jalali day
7727 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30 + 6))}}}%
7728 \ExplSyntaxOff
7729 </ca-persian>

```

18 Coptic

Adapted from jquery.calendars.package-1.1.4, written by Keith Wood, 2010. Dual license: GPL and MIT.

```

7730 <*ca-coptic>
7731 \ExplSyntaxOn
7732 <<Compute Julian day>>
7733 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
7734 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7735 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
7736 \edef#4{\fp_eval:n{%
7737 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7738 \edef\bbl@tempc{\fp_eval:n{%
7739 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7740 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7741 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7742 \ExplSyntaxOff
7743 </ca-coptic>

```

19 Buddhist

That's very simple.

```

7744 <*ca-buddhist>
7745 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
7746 \edef#4{\number\numexpr#1+543\relax}%
7747 \edef#5{#2}%
7748 \edef#6{#3}%
7749 </ca-buddhist>

```

20 Support for Plain T_EX (plain.def)

20.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```
7750 <*bplain | blplain>
7751 \catcode\{=1 % left brace is begin-group character
7752 \catcode\}=2 % right brace is end-group character
7753 \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).

```
7754 \openin 0 hyphen.cfg
7755 \ifeof0
7756 \else
7757   \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.

```
7758   \def\input #1 {%
7759     \let\input\input
7760     \a hyphen.cfg
7761     \let\input\undefined
7762   }
7763 \fi
7764 </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`.

```
7765 <bplain>\a plain.tex
7766 <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.

```
7767 <bplain>\def\fmtname{babel-plain}
7768 <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.

20.2 Emulating some L^AT_EX features

The file `babel.def` expects some definitions made in the `LATEX 2ε` 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).

```
7769 <<*Emulate LaTeX>> ≡
7770 \def\@empty{}
```

```

7771 \def\loadlocalcfg#1{%
7772   \openin0#1.cfg
7773   \ifeof0
7774     \closein0
7775   \else
7776     \closein0
7777     {\immediate\write16{*****}%
7778      \immediate\write16{* Local config file #1.cfg used}%
7779      \immediate\write16{*}%
7780     }
7781   \input #1.cfg\relax
7782 \fi
7783 \@endofldf}

```

20.3 General tools

A number of \LaTeX macro's that are needed later on.

```

7784 \long\def\@firstofone#1{#1}
7785 \long\def\@firstoftwo#1#2{#1}
7786 \long\def\@secondoftwo#1#2{#2}
7787 \def\@nnil{\nil}
7788 \def\@gobbletwo#1#2{}
7789 \def\@ifstar#1{\ifnextchar *{\@firstoftwo{#1}}}
7790 \def\@star@or@long#1{%
7791   \@ifstar
7792   {\let\@ngrel@x\relax#1}%
7793   {\let\@ngrel@x\long#1}}
7794 \let\@ngrel@x\relax
7795 \def\@car#1#2\@nil{#1}
7796 \def\@cdr#1#2\@nil{#2}
7797 \let\@typeset@protect\relax
7798 \let\protected@edef\edef
7799 \long\def\@gobble#1{}
7800 \edef\@backslashchar{\expandafter\@gobble\string\}
7801 \def\@strip@prefix#1>{}
7802 \def\@addto@macro#1#2{%
7803   \toks@\expandafter{#1#2}%
7804   \xdef#1{\the\toks@}}
7805 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7806 \def\@nameuse#1{\csname #1\endcsname}
7807 \def\@ifundefined#1{%
7808   \expandafter\ifx\csname#1\endcsname\relax
7809     \expandafter\@firstoftwo
7810   \else
7811     \expandafter\@secondoftwo
7812   \fi}
7813 \def\@expandtwoargs#1#2#3{%
7814   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7815 \def\zap@space#1 #2{%
7816   #1%
7817   \ifx#2\@empty\else\expandafter\zap@space\fi
7818   #2}
7819 \let\bbl@trace\@gobble
7820 \def\bbl@error#1#2{%
7821   \begingroup
7822     \newlinechar=`^^J
7823     \def\{^^J(babel) }%
7824     \errhelp{#2}\errmessage{\{#1}%
7825   \endgroup}
7826 \def\bbl@warning#1{%
7827   \begingroup
7828     \newlinechar=`^^J
7829     \def\{^^J(babel) }%

```

```

7830 \message{\#1}%
7831 \endgroup}
7832 \let\bbl@infowarn\bbl@warning
7833 \def\bbl@info#1{%
7834 \begingroup
7835 \newlinechar=`^^J
7836 \def\{^^J}%
7837 \wlog{#1}%
7838 \endgroup}

```

$\LaTeX 2\epsilon$ has the command `\onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

7839 \ifx\@preamblecmds\undefined
7840 \def\@preamblecmds{}
7841 \fi
7842 \def\@onlypreamble#1{%
7843 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7844 \@preamblecmds\do#1}}
7845 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

7846 \def\begindocument{%
7847 \@begindocumenthook
7848 \global\let\@begindocumenthook\undefined
7849 \def\do##1{\global\let##1\undefined}%
7850 \@preamblecmds
7851 \global\let\do\noexpand}

7852 \ifx\@begindocumenthook\undefined
7853 \def\@begindocumenthook{}
7854 \fi
7855 \@onlypreamble\@begindocumenthook
7856 \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`.

```

7857 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7858 \@onlypreamble\AtEndOfPackage
7859 \def\@endofldf{}
7860 \@onlypreamble\@endofldf
7861 \let\bbl@afterlang\empty
7862 \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.

```

7863 \catcode`\&=\z@
7864 \ifx&\if@files\undefined
7865 \expandafter\let\csname if@files\expandafter\endcsname
7866 \csname iffalse\endcsname
7867 \fi
7868 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

7869 \def\newcommand{\@star@or@long\new@command}
7870 \def\new@command#1{%
7871 \@testopt{\@newcommand#1}0}
7872 \def\@newcommand#1[#2]{%
7873 \ifnextchar [{\@xargdef#1[#2]}%
7874 {\@argdef#1[#2]}}
7875 \long\def\@argdef#1[#2]#3{%
7876 \@yargdef#1\@ne{#2}{#3}}
7877 \long\def\@xargdef#1[#2][#3]#4{%
7878 \expandafter\def\expandafter#1\expandafter{%

```

```

7879 \expandafter\@protected@testopt\expandafter #1%
7880 \csname\string#1\expandafter\endcsname{#3}}%
7881 \expandafter\@yargdef \csname\string#1\endcsname
7882 \tw@{#2}{#4}}
7883 \long\def\@yargdef#1#2#3{%
7884 \@tempcnta#3\relax
7885 \advance \@tempcnta \@ne
7886 \let\@hash@\relax
7887 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7888 \@tempcntb #2%
7889 \@whilenum\@tempcntb <\@tempcnta
7890 \do{%
7891 \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
7892 \advance\@tempcntb \@ne}%
7893 \let\@hash@##%
7894 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7895 \def\providecommand{\@star@or@long\provide@command}
7896 \def\provide@command#1{%
7897 \begingroup
7898 \escapechar\m@ne\xdef\@gtempa{\string#1}}%
7899 \endgroup
7900 \expandafter\@ifundefined\@gtempa
7901 {\def\reserved@a{\new@command#1}}%
7902 {\let\reserved@a\relax
7903 \def\reserved@a{\new@command\reserved@a}}%
7904 \reserved@a}%
7905 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7906 \def\declare@robustcommand#1{%
7907 \edef\reserved@a{\string#1}%
7908 \def\reserved@b{#1}%
7909 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7910 \edef#1{%
7911 \ifx\reserved@a\reserved@b
7912 \noexpand\x@protect
7913 \noexpand#1%
7914 \fi
7915 \noexpand\protect
7916 \expandafter\noexpand\csname
7917 \expandafter\@gobble\string#1 \endcsname
7918 }%
7919 \expandafter\new@command\csname
7920 \expandafter\@gobble\string#1 \endcsname
7921 }
7922 \def\x@protect#1{%
7923 \ifx\protect\@typeset@protect\else
7924 \@x@protect#1%
7925 \fi
7926 }
7927 \catcode\&=\z@ % Trick to hide conditionals
7928 \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`.

```

7929 \def\bbl@tempa{\csname newif\endcsname&fin@}
7930 \catcode\&=4
7931 \ifx\in@\@undefined
7932 \def\in@#1#2{%
7933 \def\in@@##1##2##3\in@@{%
7934 \ifx\in@@##2\in@false\else\in@true\fi}%
7935 \in@@##1\in@\in@@}
7936 \else
7937 \let\bbl@tempa\@empty

```



```

7938 \fi
7939 \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).

```

7940 \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.

```

7941 \def@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\LaTeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

7942 \ifx\@tempcnta\@undefined
7943   \csname newcount\endcsname\@tempcnta\relax
7944 \fi
7945 \ifx\@tempcntb\@undefined
7946   \csname newcount\endcsname\@tempcntb\relax
7947 \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`).

```

7948 \ifx\bye\@undefined
7949   \advance\count10 by -2\relax
7950 \fi
7951 \ifx\@ifnextchar\@undefined
7952   \def\@ifnextchar#1#2#3{%
7953     \let\reserved@d=#1%
7954     \def\reserved@a{#2}\def\reserved@b{#3}%
7955     \futurelet\@let@token\@ifnch}
7956   \def\@ifnch{%
7957     \ifx\@let@token\@sptoken
7958       \let\reserved@c\@xifnch
7959     \else
7960       \ifx\@let@token\reserved@d
7961         \let\reserved@c\reserved@a
7962       \else
7963         \let\reserved@c\reserved@b
7964       \fi
7965     \fi
7966     \reserved@c}
7967   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
7968   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7969 \fi
7970 \def\@testopt#1#2{%
7971   \@ifnextchar[{\#1}{\#1[\#2]}}
7972 \def\@protected@testopt#1{%
7973   \ifx\protect\@typeset@protect
7974     \expandafter\@testopt
7975   \else
7976     \@x@protect#1%
7977   \fi}
7978 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7979   #2\relax}\fi}
7980 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7981   \else\expandafter\@gobble\fi{#1}}

```

20.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

7982 \def\DeclareTextCommand{%
7983   \@dec@text@cmd\providecommand
7984 }
7985 \def\ProvideTextCommand{%
7986   \@dec@text@cmd\providecommand
7987 }
7988 \def\DeclareTextSymbol#1#2#3{%
7989   \@dec@text@cmd\chardef#1{#2}#3\relax
7990 }
7991 \def\@dec@text@cmd#1#2#3{%
7992   \expandafter\def\expandafter#2%
7993     \expandafter{%
7994       \csname#3-cmd\expandafter\endcsname
7995       \expandafter#2%
7996       \csname#3\string#2\endcsname
7997     }%
7998 %   \let\@ifdefinable\@rc@ifdefinable
7999   \expandafter#1\csname#3\string#2\endcsname
8000 }
8001 \def\@current@cmd#1{%
8002   \ifx\protect\@typeset@protect\else
8003     \noexpand#1\expandafter\@gobble
8004   \fi
8005 }
8006 \def\@changed@cmd#1#2{%
8007   \ifx\protect\@typeset@protect
8008     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8009       \expandafter\ifx\csname ?\string#1\endcsname\relax
8010         \expandafter\def\csname ?\string#1\endcsname{%
8011           \@changed@x@err{#1}%
8012         }%
8013       \fi
8014       \global\expandafter\let
8015         \csname\cf@encoding\string#1\expandafter\endcsname
8016         \csname ?\string#1\endcsname
8017     \fi
8018     \csname\cf@encoding\string#1%
8019       \expandafter\endcsname
8020   \else
8021     \noexpand#1%
8022   \fi
8023 }
8024 \def\@changed@x@err#1{%
8025   \errhelp{Your command will be ignored, type <return> to proceed}%
8026   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8027 \def\DeclareTextCommandDefault#1{%
8028   \DeclareTextCommand#1?%
8029 }
8030 \def\ProvideTextCommandDefault#1{%
8031   \ProvideTextCommand#1?%
8032 }
8033 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8034 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8035 \def\DeclareTextAccent#1#2#3{%
8036   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8037 }
8038 \def\DeclareTextCompositeCommand#1#2#3#4{%
8039   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8040   \edef\reserved@b{\string##1}%
8041   \edef\reserved@c{%
8042     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8043   \ifx\reserved@b\reserved@c
8044     \expandafter\expandafter\expandafter\ifx

```

```

8045     \expandafter\@car\reserved@a\relax\relax\@nil
8046     \@text@composite
8047   \else
8048     \edef\reserved@b##1{%
8049       \def\expandafter\noexpand
8050         \csname#2\string#1\endcsname####1{%
8051         \noexpand\@text@composite
8052           \expandafter\noexpand\csname#2\string#1\endcsname
8053           ####1\noexpand\@empty\noexpand\@text@composite
8054           {##1}%
8055       }%
8056   }%
8057   \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8058   \fi
8059   \expandafter\def\csname\expandafter\string\csname
8060     #2\endcsname\string#1-\string#3\endcsname{#4}
8061   \else
8062     \errhelp{Your command will be ignored, type <return> to proceed}%
8063     \errmessage{\string\DeclareTextCompositeCommand\space used on
8064       inappropriate command \protect#1}
8065   \fi
8066 }
8067 \def\@text@composite#1#2#3\@text@composite{%
8068   \expandafter\@text@composite@x
8069     \csname\string#1-\string#2\endcsname
8070 }
8071 \def\@text@composite@x#1#2{%
8072   \ifx#1\relax
8073     #2%
8074   \else
8075     #1%
8076   \fi
8077 }
8078 %
8079 \def\@strip@args#1:#2-#3\@strip@args{#2}
8080 \def\DeclareTextComposite#1#2#3#4{%
8081   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8082   \bgroup
8083     \lccode`\@=#4%
8084     \lowercase{%
8085   \egroup
8086     \reserved@a @%
8087   }%
8088 }
8089 %
8090 \def\UseTextSymbol#1#2{#2}
8091 \def\UseTextAccent#1#2#3{}
8092 \def\@use@text@encoding#1{}
8093 \def\DeclareTextSymbolDefault#1#2{%
8094   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8095 }
8096 \def\DeclareTextAccentDefault#1#2{%
8097   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8098 }
8099 \def\cf@encoding{OT1}

```

Currently we only use the $\text{\LaTeX} 2_{\epsilon}$ method for accents for those that are known to be made active in *some* language definition file.

```

8100 \DeclareTextAccent{"}{OT1}{127}
8101 \DeclareTextAccent{'}{OT1}{19}
8102 \DeclareTextAccent{^}{OT1}{94}
8103 \DeclareTextAccent{\`}{OT1}{18}
8104 \DeclareTextAccent{\~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```
8105 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8106 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
8107 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
8108 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8109 \DeclareTextSymbol{\i}{OT1}{16}
8110 \DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX has, we just `\let` it to `\sevenrm`.

```
8111 \ifx\scriptsize\undefined
8112   \let\scriptsize\sevenrm
8113 \fi
```

And a few more “dummy” definitions.

```
8114 \def\language{english}%
8115 \let\bbl@opt@shorthands\@nnil
8116 \def\bbl@ifshorthand#1#2#3{#2}%
8117 \let\bbl@language@opts\@empty
8118 \ifx\babeloptionstrings\undefined
8119   \let\bbl@opt@strings\@nnil
8120 \else
8121   \let\bbl@opt@strings\babeloptionstrings
8122 \fi
8123 \def\BabelStringsDefault{generic}
8124 \def\bbl@tempa{normal}
8125 \ifx\babeloptionmath\bbl@tempa
8126   \def\bbl@mathnormal{\noexpand\textormath}
8127 \fi
8128 \def\AfterBabelLanguage#1#2{}
8129 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
8130 \let\bbl@afterlang\relax
8131 \def\bbl@opt@safe{BR}
8132 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
8133 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
8134 \expandafter\newif\csname ifbbl@single\endcsname
8135 \chardef\bbl@bidimode\z@
8136 <</Emulate LaTeX>>
```

A proxy file:

```
8137 <*\plain>
8138 \input babel.def
8139 </\plain>
```

21 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.

During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.

- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *L^AT_EX, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: T_EXhax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German T_EX*, TUGboat 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International L^AT_EX is ready to use*, TUGboat 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).