

Babel

Version 3.47.2085
2020/07/31

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	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	11
1.10	Shorthands	12
1.11	Package options	16
1.12	The base option	18
1.13	ini files	18
1.14	Selecting fonts	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	32
1.18	Dates	34
1.19	Accessing language info	34
1.20	Hyphenation and line breaking	35
1.21	Selection based on BCP 47 tags	38
1.22	Selecting scripts	39
1.23	Selecting directions	40
1.24	Language attributes	44
1.25	Hooks	44
1.26	Languages supported by babel with ldf files	45
1.27	Unicode character properties in luatex	46
1.28	Tweaking some features	47
1.29	Tips, workarounds, known issues and notes	47
1.30	Current and future work	48
1.31	Tentative and experimental code	49
2	Loading languages with language.dat	49
2.1	Format	49
3	The interface between the core of babel and the language definition files	50
3.1	Guidelines for contributed languages	51
3.2	Basic macros	52
3.3	Skeleton	53
3.4	Support for active characters	54
3.5	Support for saving macro definitions	55
3.6	Support for extending macros	55
3.7	Macros common to a number of languages	55
3.8	Encoding-dependent strings	56
4	Changes	59
4.1	Changes in babel version 3.9	59
II	Source code	60

5	Identification and loading of required files	60
6	locale directory	60
7	Tools	61
7.1	Multiple languages	65
7.2	The Package File (\LaTeX , babel.sty)	66
7.3	base	67
7.4	Conditional loading of shorthands	69
7.5	Cross referencing macros	71
7.6	Marks	74
7.7	Preventing clashes with other packages	75
7.7.1	ifthen	75
7.7.2	varioref	75
7.7.3	hhline	76
7.7.4	hyperref	76
7.7.5	fancyhdr	76
7.8	Encoding and fonts	77
7.9	Basic bidi support	79
7.10	Local Language Configuration	84
8	The kernel of Babel (babel.def, common)	87
8.1	Tools	87
9	Multiple languages	88
9.1	Selecting the language	91
9.2	Errors	99
9.3	Hooks	102
9.4	Setting up language files	104
9.5	Shorthands	106
9.6	Language attributes	115
9.7	Support for saving macro definitions	117
9.8	Short tags	118
9.9	Hyphens	119
9.10	Multiencoding strings	120
9.11	Macros common to a number of languages	126
9.12	Making glyphs available	126
9.12.1	Quotation marks	127
9.12.2	Letters	128
9.12.3	Shorthands for quotation marks	129
9.12.4	Umlauts and tremas	130
9.13	Layout	131
9.14	Load engine specific macros	132
9.15	Creating and modifying languages	132
10	Adjusting the Babel bahavior	151
11	Loading hyphenation patterns	153
12	Font handling with fontspec	158

13	Hooks for XeTeX and LuaTeX	162
13.1	XeTeX	162
13.2	Layout	164
13.3	LuaTeX	165
13.4	Southeast Asian scripts	171
13.5	CJK line breaking	175
13.6	Automatic fonts and ids switching	175
13.7	Layout	186
13.8	Auto bidi with basic and basic-r	188
14	Data for CJK	199
15	The ‘nil’ language	199
16	Support for Plain T_EX (plain.def)	200
16.1	Not renaming hyphen.tex	200
16.2	Emulating some L ^A T _E X features	201
16.3	General tools	201
16.4	Encoding related macros	205
17	Acknowledgements	208

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	8
Unknown language ‘LANG’	9
Argument of \language@active@arg” has an extra }	13
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	28
Package babel Info: The following fonts are not babel standard families	28

Part I

User guide

- This user guide focuses on internationalization and localization with \LaTeX . There are also some notes on its use with Plain \TeX .
- Changes and new features with relation to version 3.8 are highlighted with **New X.XX**, and there are some notes for the latest versions in the babel wiki. The most recent features could be still unstable. Please, report any issues you find in GitHub, which is better than just complaining on an e-mail list or a web forum.
- If you are interested in the \TeX multilingual support, please join the kadingira mail list. You can follow the development of babel in GitHub (which provides many sample files, too). If you are the author of a package, feel free to send to me a few test files which I'll add to mine, so that possible issues could be caught in the development phase.
- See section 3.1 for contributing a language.
- The first sections describe the traditional way of loading a language (with `ldf` files). The alternative way based on `ini` files, which complements the previous one (it does *not* replace it), is described below.

1 The user interface

1.1 Monolingual documents

In most cases, a single language is required, and then all you need in \LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current \LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

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

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[russian]{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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.

1.2 Multilingual documents

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

EXAMPLE In \LaTeX , the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

PDFTEX

```
\documentclass{article}

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

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

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

1.3 Mostly monolingual documents

New 3.39 Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need

is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of `\babelfont`, if used.)

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

EXAMPLE A trivial document is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

1.5 Troubleshooting

- Loading directly sty files in \LaTeX (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}.
```

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.

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

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

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

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

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

Spaces after the environment are ignored.

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

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

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a

line. However, by default it never complied with the documented behavior and it is just a version as environment of `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while `otherlanguage*` does not.

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

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

1.9 More on selection

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

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

NOTE Note the following:

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

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

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 $\{\langle shorthands-list \rangle\}$
\shorthandoff $\ast\{\langle shorthands-list \rangle\}$

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

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

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

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

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

\useshorthands $\ast\{\langle char \rangle\}$

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

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

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

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

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Languages with no shorthands Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh
Languages with only " as defined shorthand character Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

Basque " ' ~
Breton : ; ? !
Catalan " ' `
Czech " -
Esperanto ^
Estonian " ~
French (all varieties) : ; ? !
Galician " . ' ~ < >
Greek ~
Hungarian `
Kurmanji ^
Latin " ^ =
Slovak " ^ ' -
Spanish " . < > ' ~
Turkish : ! =

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

`\ifbabelshorthand` {<character>}{<true>}{<false>}

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

`\aliasshorthand` {<original>}{<alias>}

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

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

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

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

⁷Thanks to Enrico Gregorio

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

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

1.11 Package options

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

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

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

activegrave Same for ```.

shorthands= $\langle char \rangle \langle char \rangle \dots$ | off

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

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

If `'` is included, `activeacute` is set; if ``` is included, `activegrave` is set. Active characters (like `~`) should be preceded by `\string` (otherwise they will be expanded by \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= $\langle file \rangle$

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

main=	<code><language></code> 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=	<code><language></code> By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
noconfigs	Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected .cfg file. However, if the key config is set, this file is loaded.
showlanguages	Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
nocase	New 3.9l Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code>) are ignored. Use only if there are incompatibilities with other packages.
silent	New 3.9l No warnings and no <i>infos</i> are written to the log file. ⁹
strings=	<code>generic unicode encoded <label> </code> Selects the encoding of strings in languages supporting this feature. Predefined labels are <code>generic</code> (for traditional \TeX , LICR and ASCII strings), <code>unicode</code> (for engines like xetex and luatex) and <code>encoded</code> (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in <code>\MakeUppercase</code> and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
hyphenmap=	<code>off first select other other*</code> New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it. ¹⁰ It can take the following values: off deactivates this feature and no case mapping is applied; first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at <code>\begin{document}</code>), but also the first <code>\selectlanguage</code> in the preamble, and it's the default if a single language option has been stated; ¹¹ select sets it only at <code>\selectlanguage</code> ; other also sets it at <code>otherlanguage</code> ; other* also sets it at <code>otherlanguage*</code> as well as in heads and foots (if the option <code>headfoot</code> is used) and in auxiliary files (ie, at <code>\select@language</code>), and it's the default if several language options have been stated. The option <code>first</code> can be regarded as an optimized version of <code>other*</code> for monolingual documents. ¹²
bidi=	<code>default basic basic-r bidi-l bidi-r</code>

⁹You can use alternatively the package `silence`.

¹⁰Turned off in plain.

¹¹Duplicated options count as several ones.

¹²Providing `foreign` is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, `other` is provided even if I [JBL]

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

layout=

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

1.12 The base option

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

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

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

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

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

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

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

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

1.13 ini files

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

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

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

think it isn't really useful, but who knows.

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

LUATEX/XETEX

```
\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

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

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{lᦺ lᦴ lᦳ lᦵ lᦶ lᦷ} % 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{ltjbook}
\usepackage[japanese]{babel}
```

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

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

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

af	Afrikaans ^{ul}	bo	Tibetan ^u
agq	Aghem	brx	Bodo
ak	Akan	bs-Cyrl	Bosnian
am	Amharic ^{ul}	bs-Latn	Bosnian ^{ul}
ar	Arabic ^{ul}	bs	Bosnian ^{ul}
ar-DZ	Arabic ^{ul}	ca	Catalan ^{ul}
ar-MA	Arabic ^{ul}	ce	Chechen
ar-SY	Arabic ^{ul}	cgg	Chiga
as	Assamese	chr	Cherokee
asa	Asu	ckb	Central Kurdish
ast	Asturian ^{ul}	cop	Coptic
az-Cyrl	Azerbaijani	cs	Czech ^{ul}
az-Latn	Azerbaijani	cu	Church Slavic
az	Azerbaijani ^{ul}	cu-Cyrs	Church Slavic
bas	Basaa	cu-Glag	Church Slavic
be	Belarusian ^{ul}	cy	Welsh ^{ul}
bem	Bemba	da	Danish ^{ul}
bez	Bena	dav	Taita
bg	Bulgarian ^{ul}	de-AT	German ^{ul}
bm	Bambara	de-CH	German ^{ul}
bn	Bangla ^{ul}	de	German ^{ul}

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

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

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

aghem	cantonese
akan	catalan
albanian	centralatlastamazight
american	centralkurdish
amharic	chechen
ancientgreek	cherokee
arabic	chiga
arabic-algeria	chinese-hans-hk
arabic-DZ	chinese-hans-mo
arabic-morocco	chinese-hans-sg
arabic-MA	chinese-hans
arabic-syria	chinese-hant-hk
arabic-SY	chinese-hant-mo
armenian	chinese-hant
assamese	chinese-simplified-hongkongsarchina
asturian	chinese-simplified-macausarchina
asu	chinese-simplified-singapore
australian	chinese-simplified
austrian	chinese-traditional-hongkongsarchina
azerbaijani-cyrillic	chinese-traditional-macausarchina
azerbaijani-cyrl	chinese-traditional
azerbaijani-latin	chinese
azerbaijani-latn	churchslavic
azerbaijani	churchslavic-cyrs
bafia	churchslavic-oldcyrillic ¹³
bambara	churchsslavic-glag
basaa	churchsslavic-glagolitic
basque	cognian
belarusian	cornish
bemba	croatian
bena	czech
bengali	danish
bodo	duala
bosnian-cyrillic	dutch
bosnian-cyrl	dzongkha
bosnian-latin	embu
bosnian-latn	english-au
bosnian	english-australia
brazilian	english-ca
breton	english-canada
british	english-gb
bulgarian	english-newzealand
burmese	english-nz
canadian	english-unitedkingdom

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

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

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

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

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

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

Modifying and adding values to ini files

New 3.39 There is a way to modify the values of ini files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghijklj`. Keys may be added, too. Without `import` you may modify the identification keys.

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

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

Modifying the behavior of a language (say, the chapter “caption”), is sometimes necessary, but not always trivial.

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

```
\addto\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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named arhinish:

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

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

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

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

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

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

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

import= *<language-tag>*

New 3.13 Imports data from an ini file, including captions, date, and hyphenmins. For example:

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

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

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

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

There are about 200 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages will show a warning about the current lack of suitability of the date format (french, breton, and occitan).

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

`\<language>date{\the\year}{\the\month}{\the\day}`. **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= \langle *language-list* \rangle

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

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

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

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

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

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

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

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

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

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

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

script= \langle *script-name* \rangle

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

language= \langle *language-name* \rangle

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

alph= \langle *counter-name* \rangle

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

Alph= $\langle counter-name \rangle$

Same for \Alph.

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

onchar= ids | fonts

New 3.38 This option is much like an ‘event’ called when a character belonging to the script of this locale is found. 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 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 could be enough.

mapfont= direction

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

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

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

intrapenalty= $\langle penalty \rangle$

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

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

1.17 Digits and counters

New 3.20 About thirty ini files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With the option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to

avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on `\arabic`.)

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

The styles are:

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

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

Arabic `abjad`, `maghrebi.abjad`

Belarusian, Bulgarian, Macedonian, Serbian `lower`, `upper`

Bengali `alphabetic`

Coptic `epact`, `lower.letters`

Hebrew `letters` (neither `geresh` nor `gershayim` yet)

Hindi alphabetic
Armenian lower.letter, upper.letter
Japanese hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Georgian letters
Greek lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
Khmer consonant
Korean consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Marathi alphabetic
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Syriac letters
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

\iflanguage {*<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_EXsense, as a set of hyphenation patterns, and *not* as its babel name. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is true or false respectively.

`\localeinfo` `{\field}`

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

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

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

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

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

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

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

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

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

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

`\getlocaleproperty` `*{\macro}{\locale}{\property}`

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

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

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

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

Babel remembers which ini files have been loaded. There is a loop named `\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that `\LocaleForEach{\message{ **#1** }}` just shows the loaded ini's.

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

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

New 3.9a It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in \TeX are entered as `-`, and (2) *optional* or *soft hyphens*, which are entered as `\-`. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in \TeX terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account). For example, you can use the `string` replacement to replace a character (or series of them) by another character (or series of

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

them). Thus, to enter *ž* as *zh* and *š* as *sh* in a newly created locale for transliterated Russian:

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

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

1.21 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

The behaviour is adjusted with `\babeladjust` with the following parameters:

`autoload.bcp47` with values on and off.

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

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

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

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

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

1.22 Selecting scripts

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

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

`\ensureascii` $\langle text \rangle$

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

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

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

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

¹⁷But still defined for backwards compatibility.

1.23 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}
```

```
\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

Most Arabic speakers consider the two varieties to be two registers
of one language, although the two registers can be referred to in
Arabic as \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 could depend on the counter format.

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

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

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

1.24 Language attributes

`\languageattribute`

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

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

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

1.25 Hooks

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

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

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{<name>}`, `\DisableBabelHook{<name>}`. Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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`.

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

1.26 Languages supported by babel with ldf files

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

Afrikaans afrikaans

Azerbaijani azerbaijani

Basque basque

Breton breton

Bulgarian bulgarian

Catalan catalan

Croatian croatian

Czech czech

Danish danish

Dutch dutch

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

Esperanto esperanto

Estonian estonian

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

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

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

```

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

```

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

1.27 Unicode character properties in luatex

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

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

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

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

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

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

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

1.28 Tweaking some features

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

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

1.29 Tips, workarounds, known issues and notes

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

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

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

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


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

(A recent version of inputenc is required.)

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

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

microtype Adjusts the typesetting according to some languages (kerning and spacing).
Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.30 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²¹ But that is the easy part, because they don't require modifying the L^AT_EX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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

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

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

1.31 Tentative and experimental code

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

\babelprehyphenation

New 3.44 Note it is tentative, but the current behavior for glyphs should be correct.

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

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

Performance is still somewhat poor.

2 Loading languages with language.dat

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

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

2.1 Format

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

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

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

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

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

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

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

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

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

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

A typical error when using `babel` is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the `babel` system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.

²⁵This 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⟨lang⟩`, `\date⟨lang⟩`, `\extras⟨lang⟩` and `\noextras⟨lang⟩` (the last two may be left empty); where `⟨lang⟩` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date⟨lang⟩` but not `\captions⟨lang⟩` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

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

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

3.1 Guidelines for contributed languages

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

²⁶But not removed, for backward compatibility.

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

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

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

\addlanguage The macro \addlanguage is a non-outer version of the macro \newlanguage, defined in plain.tex version 3.x. Here “language” is used in the 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.
<code>\noextras<lang></code>	Because we want to let the user switch between languages, but we do not know what state \TeX might be in after the execution of <code>\extras<lang></code> , a macro that brings \TeX into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras<lang></code> .
<code>\bbl@declare@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \LaTeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions<lang></code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct \LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

```

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

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

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

\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`
`\bbl@deactivate`

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

`\declare@shorthand`

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

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

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

3.5 Support for saving macro definitions

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

`\babel@save`

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

`\babel@savevariable`

A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the `⟨variable⟩`.

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

3.6 Support for extending macros

`\addto`

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

3.7 Macros common to a number of languages

`\bbl@allowhyphens`

In several languages compound words are used. This means that when T_EX 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@sfc@q`

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

`\bbl@frenchspacing`
`\bbl@nonfrenchspacing`

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

²⁷This mechanism was introduced by Bernd Raichle.

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

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

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

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

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

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

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

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

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

²⁸In future releases further categories may be added.

```

\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}
\SetString\monthivname{April}
\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
\SetString\monthviiname{Juli}
\SetString\monthviiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

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

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

maintainers of the current languages to decide if using it is appropriate.²⁹

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands $\{\langle code \rangle\}$
The code is delayed and executed at the global scope just after \EndBabelCommands.

\SetString $\{\langle macro-name \rangle\}\{\langle string \rangle\}$
Adds $\langle macro-name \rangle$ to the current category, and defines globally $\langle lang-macro-name \rangle$ to $\langle code \rangle$ (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop $\{\langle macro-name \rangle\}\{\langle string-list \rangle\}$
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with abday):

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

#1 is replaced by the roman numeral.

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

```
\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]  
\SetCase  
  {\uccode"10=`I\relax}  
  {\lccode`I="10\relax}  
  
\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]  
\SetCase  
  {\uccode`i=`İ\relax  
   \uccode`ı=`I\relax}  
  {\lccode`İ=`i\relax  
   \lccode`I=`ı\relax}  
  
\StartBabelCommands{turkish}{}  
\SetCase  
  {\uccode`i="9D\relax  
   \uccode"19=`I\relax}  
  {\lccode"9D=`i\relax  
   \lccode`I="19\relax}
```

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

```
\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

`\SetHyphenMap` `{\to-lower-macros}`

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

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

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

```
\SetHyphenMap{\BabelLowerMM{"100}{\lccode}{2}{\lccode}}
```

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

4 Changes

4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with `^` (if activated) and also if deactivated.

- Active chars were not reset at the end of language options, and that lead to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with babel were not recognized when called as global options.

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.
version of the ini file
level “version” of the ini specification . which keys are available (they may grow in a compatible way) and how they should be read.
encodings a descriptive list of font encodings.
[captions] section of captions in the file charset
[captions.licr] same, but in pure ASCII using the LICR
date.long fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMMM for the month name) and anything outside is text. In addition, [] is a non breakable space and [.] is an abbreviation dot.

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

7 Tools

```
1 <<version=3.47.2085>>
2 <<date=2020/07/31>>
```

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

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

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

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

```
21 \def\bbl@add@list#1#2{%
22   \edef#1{#1#2,}
```

```

23 \bbl@ifunset{\bbl@stripslash#1}%
24 {}%
25 {\ifx#1\@empty\else#1,\fi}%
26 #2}}

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

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

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

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

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

35 \def\bbl@tempa#1{%
36 \long\def\bbl@trim##1##2{%
37 \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38 \def\bbl@trim@c{%
39 \ifx\bbl@trim@a\@sptoken
40 \expandafter\bbl@trim@b
41 \else
42 \expandafter\bbl@trim@b\expandafter#1%
43 \fi}%
44 \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
45 \bbl@tempa{ }
46 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \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  $\epsilon$ -tex engine, it is based on \ifcsname, which is more
efficient, and do not waste memory.

48 \begingroup
49 \gdef\bbl@ifunset#1{%
50 \expandafter\ifx\csname#1\endcsname\relax
51 \expandafter\@firstoftwo
52 \else
53 \expandafter\@secondoftwo
54 \fi}
55 \bbl@ifunset{ifcsname}%
56 {}%

```

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

```

57   {\gdef\bb@ifunset#1{%
58     \ifcsname#1\endcsname
59     \expandafter\ifx\csname#1\endcsname\relax
60     \bb@afterelse\expandafter\@firstoftwo
61     \else
62     \bb@afterfi\expandafter\@secondoftwo
63     \fi
64     \else
65     \expandafter\@firstoftwo
66     \fi}}
67 \endgroup

```

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

```

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

```

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).

```

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

```

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

```

`\bb@replace`

```

91 \def\bb@replace#1#2#3{% in #1 -> repl #2 by #3
92   \toks@{ }%
93   \def\bb@replace@aux##1#2##2#2{%
94     \ifx\bb@nil##2%
95       \toks@\expandafter{\the\toks@##1}%
96     \else
97       \toks@\expandafter{\the\toks@##1#3}%
98       \bb@afterfi
99       \bb@replace@aux##2#2%
100     \fi}%
101   \expandafter\bb@replace@aux#1#2\bb@nil#2%
102   \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 checking the replacement is really necessary or just paranoia).

```

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

```

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

```

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

```

```

150 \@ne
151 \fi

```

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

```

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

```

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

```

160 <<*Make sure ProvidesFile is defined>> ≡
161 \ifx\ProvidesFile\@undefined
162 \def\ProvidesFile#1[#2 #3 #4]{%
163 \wlog{File: #1 #4 #3 <#2>}%
164 \let\ProvidesFile\@undefined}
165 \fi
166 <</Make sure ProvidesFile is defined>>

```

7.1 Multiple languages

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

```

167 <<*Define core switching macros>> ≡
168 \ifx\language\@undefined
169 \csname newcount\endcsname\language
170 \fi
171 <</Define core switching macros>>

```

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

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

```

172 <<*Define core switching macros>> ≡
173 <<*Define core switching macros>> ≡
174 \countdef\last@language=19 % TODO. why? remove?
175 \def\addlanguage{\csname newlanguage\endcsname}
176 <</Define core switching macros>>

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or \LaTeX 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

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

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

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

The first two options are for debugging.

```
177 (*package)
178 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
179 \ProvidesPackage{babel}[\langle\date\rangle\ \langle\version\rangle\ The Babel package]
180 \ifpackagewith{babel}{debug}
181   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
182   \let\bbl@debug\@firstofone}
183 {\providecommand\bbl@trace[1]{}%
184  \let\bbl@debug\gobble}
185 \langle\Basic macros\rangle
186 % Temporarily repeat here the code for errors
187 \def\bbl@error#1#2{%
188   \begingroup
189     \def\{\MessageBreak}%
190     \PackageError{babel}{#1}{#2}%
191   \endgroup}
192 \def\bbl@warning#1{%
193   \begingroup
194     \def\{\MessageBreak}%
195     \PackageWarning{babel}{#1}%
196   \endgroup}
197 \def\bbl@infowarn#1{%
198   \begingroup
199     \def\{\MessageBreak}%
200     \GenericWarning
201       {(babel) \@spaces\@spaces\@spaces}%
202       {Package babel Info: #1}%
203   \endgroup}
204 \def\bbl@info#1{%
205   \begingroup
206     \def\{\MessageBreak}%
207     \PackageInfo{babel}{#1}%
208   \endgroup}
209 \def\bbl@nocaption{\protect\bbl@nocaption@i}
210 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
211   \global\@namedef{#2}{\textbf{?#1?}}%
212   \@nameuse{#2}%
213   \bbl@warning{%
214     \@backslashchar#2 not set. Please, define it\\%
215     after the language has been loaded (typically\\%
216     in the preamble) with something like:\\%
217     \string\renewcommand\@backslashchar#2{..}\\%
218     Reported}}
219 \def\bbl@tentative{\protect\bbl@tentative@i}
220 \def\bbl@tentative@i#1{%
221   \bbl@warning{%
222     Some functions for '#1' are tentative.\\%
223     They might not work as expected and their behavior\\%
224     could change in the future.\\%
225     Reported}}
```

```

226 \def\@nolanerr#1{%
227   \bbl@error
228   {You haven't defined the language #1\space yet.\%
229     Perhaps you misspelled it or your installation\%
230     is not complete}%
231   {Your command will be ignored, type <return> to proceed}}
232 \def\@nopatterns#1{%
233   \bbl@warning
234   {No hyphenation patterns were preloaded for\%
235     the language `#1' into the format.\%
236     Please, configure your TeX system to add them and\%
237     rebuild the format. Now I will use the patterns\%
238     preloaded for \bbl@nulllanguage\space instead}}
239   % End of errors
240 \@ifpackagewith{babel}{silent}
241   {\let\bbl@info\@gobble
242    \let\bbl@infowarn\@gobble
243    \let\bbl@warning\@gobble}
244   {}
245 %
246 \def\AfterBabelLanguage#1{%
247   \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.

248 \ifx\bbl@languages\@undefined\else
249   \begingroup
250     \catcode`\^^I=12
251     \@ifpackagewith{babel}{showlanguages}{%
252       \begingroup
253         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
254         \wlog{<*languages>}%
255         \bbl@languages
256         \wlog{</languages>}%
257       \endgroup}{%
258     \endgroup
259     \def\bbl@elt#1#2#3#4{%
260       \ifnum#2=\z@
261         \gdef\bbl@nulllanguage{#1}%
262         \def\bbl@elt##1##2##3##4{}}%
263     \fi}%
264   \bbl@languages
265 \fi%

```

7.3 base

The first ‘real’ option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that \LaTeX 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.

```

266 \bbl@trace{Defining option 'base'}
267 \@ifpackagewith{babel}{base}{%
268   \let\bbl@onlyswitch\@empty
269   \let\bbl@provide@locale\relax
270   \input babel.def
271   \let\bbl@onlyswitch\@undefined

```

```

272 \ifx\directlua\@undefined
273 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
274 \else
275 \input luababel.def
276 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
277 \fi
278 \DeclareOption{base}{}%
279 \DeclareOption{showlanguages}{}%
280 \ProcessOptions
281 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
282 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
283 \global\let@ifl@ter@@\ifl@ter
284 \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\ifl@ter@@}%
285 \endinput}{}%
286 % \end{macrocode}
287 %
288 % \subsection{\texttt{key=value} options and other general option}
289 %
290 % The following macros extract language modifiers, and only real
291 % package options are kept in the option list. Modifiers are saved
292 % and assigned to |\BabelModifiers| at |\bbl@load@language|; when
293 % no modifiers have been given, the former is |\relax|. How
294 % modifiers are handled are left to language styles; they can use
295 % |\in@|, loop them with |\@for| or load |keyval|, for example.
296 %
297 % \begin{macrocode}
298 \bbl@trace{key=value and another general options}
299 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
300 \def\bbl@tempb#1.#2{%
301   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
302 \def\bbl@tempd#1.#2\@nnil{%
303   \ifx\@empty#2%
304     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
305   \else
306     \in@{=}{#1}\ifin@
307     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
308   \else
309     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
310   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
311   \fi
312 \fi}
313 \let\bbl@tempc\@empty
314 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
315 \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.

```

316 \DeclareOption{KeepShorthandsActive}{}
317 \DeclareOption{activeacute}{}
318 \DeclareOption{activegrave}{}
319 \DeclareOption{debug}{}
320 \DeclareOption{noconfigs}{}
321 \DeclareOption{showlanguages}{}
322 \DeclareOption{silent}{}
323 \DeclareOption{mono}{}
324 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
325 % Don't use. Experimental. TODO.

```

```

326 \newif\ifbbl@single
327 \DeclareOption{selectors=off}{\bbl@singletrue}
328 <<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.

```

329 \let\bbl@opt@shorthands@nil
330 \let\bbl@opt@config@nil
331 \let\bbl@opt@main@nil
332 \let\bbl@opt@headfoot@nil
333 \let\bbl@opt@layout@nil

```

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

```

334 \def\bbl@tempa#1=#2\bbl@tempa{%
335   \bbl@csarg\ifx{opt@#1}\@nil
336   \bbl@csarg\edef{opt@#1}{#2}%
337   \else
338     \bbl@error
339     {Bad option `#1=#2'. Either you have misspelled the\\%
340     key or there is a previous setting of `#1'. Valid\\%
341     keys are, among others, `shorthands', `main', `bidi',\\%
342     `strings', `config', `headfoot', `safe', `math'.}%
343     {See the manual for further details.}
344   \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.

```

345 \let\bbl@language@opts@empty
346 \DeclareOption*{%
347   \bbl@xin@{\string=}{\CurrentOption}%
348   \ifin@
349     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
350   \else
351     \bbl@add@list\bbl@language@opts{\CurrentOption}%
352   \fi}

```

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

```

353 \ProcessOptions*

```

7.4 Conditional loading of shorthands

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

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

```

354 \bbl@trace{Conditional loading of shorthands}
355 \def\bbl@sh@string#1{%
356   \ifx#1\@empty\else
357     \ifx#1t\string~%
358     \else\ifx#1c\string,%
359     \else\string#1%

```

```

360   \fi\fi
361   \expandafter\bb1@sh@string
362 \fi}
363 \ifx\bb1@opt@shorthands\@nnil
364   \def\bb1@ifshorthand#1#2#3{#2}%
365 \else\ifx\bb1@opt@shorthands\@empty
366   \def\bb1@ifshorthand#1#2#3{#3}%
367 \else

```

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

```

368 \def\bb1@ifshorthand#1{%
369   \bb1@xin@{\string#1}{\bb1@opt@shorthands}%
370   \ifin@
371     \expandafter\@firstoftwo
372   \else
373     \expandafter\@secondoftwo
374   \fi}

```

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

```

375 \edef\bb1@opt@shorthands{%
376   \expandafter\bb1@sh@string\bb1@opt@shorthands\@empty}%

```

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

```

377 \bb1@ifshorthand{'}%
378   {\PassOptionsToPackage{activeacute}{babel}}{}
379 \bb1@ifshorthand{'}%
380   {\PassOptionsToPackage{activegrave}{babel}}{}
381 \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.

```

382 \ifx\bb1@opt@headfoot\@nnil\else
383   \g@addto@macro\@resetactivechars{%
384     \set@typeset@protect
385     \expandafter\select@language@x\expandafter{\bb1@opt@headfoot}%
386     \let\protect\noexpand}
387 \fi

```

For the option `safe` we use a different approach – `\bb1@opt@safe` says which macros are redefined (B for bibs and R for refs). By default, both are set.

```

388 \ifx\bb1@opt@safe\@undefined
389   \def\bb1@opt@safe{BR}
390 \fi
391 \ifx\bb1@opt@main\@nnil\else
392   \edef\bb1@language@opts{%
393     \ifx\bb1@language@opts\@empty\else\bb1@language@opts,\fi
394     \bb1@opt@main}
395 \fi

```

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

```

396 \bb1@trace{Defining IfBabelLayout}
397 \ifx\bb1@opt@layout\@nnil
398   \newcommand\IfBabelLayout[3]{#3}%
399 \else

```

```

400 \newcommand\IfBabelLayout[1]{%
401   \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
402   \ifin@
403     \expandafter\@firstoftwo
404   \else
405     \expandafter\@secondoftwo
406   \fi}
407 \fi

```

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

```
408 \input babel.def
```

7.5 Cross referencing macros

The \LaTeX book states:

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

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

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

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

```

409 <<*More package options>> ≡
410 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
411 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
412 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
413 <</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.

```

414 \bbl@trace{Cross referencing macros}
415 \ifx\bbl@opt@safe\@empty\else
416   \def\@newl@bel#1#2#3{%
417     {\@safe@activetrue
418       \bbl@ifunset{#1#2}%
419       \relax
420       {\gdef\@multiplelabels{%
421         \@latex@warning@no@line{There were multiply-defined labels}}%
422         \@latex@warning@no@line{Label `#2' multiply defined}}%
423       \global\@namedef{#1#2}{#3}}}

```

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

```

424 \CheckCommand*\@testdef[3]{%
425   \def\reserved@a{#3}%
426   \expandafter\ifx\csname#1#2\endcsname\reserved@a
427   \else
428     \@tempwattrue
429   \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.

```

430 \def\@testdef#1#2#3{% TODO. With @samestring?
431   \@safe@activetrue
432   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
433   \def\bbl@tempb{#3}%
434   \@safe@activesfalse
435   \ifx\bbl@tempa\relax
436   \else
437     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
438   \fi
439   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
440   \ifx\bbl@tempa\bbl@tempb
441   \else
442     \@tempswatrue
443   \fi}
444 \fi

```

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

```

445 \bbl@xin@{R}\bbl@opt@safe
446 \ifin@
447   \bbl@redefineroobust\ref#1{%
448     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
449   \bbl@redefineroobust\pageref#1{%
450     \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
451 \else
452   \let\org@ref\ref
453   \let\org@pageref\pageref
454 \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.

```

455 \bbl@xin@{B}\bbl@opt@safe
456 \ifin@
457   \bbl@redefine\@citex[#1]#2{%
458     \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
459     \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.

```

460 \AtBeginDocument{%
461   \@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.)

```

462   \def\@citex[#1][#2]#3{%
463     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse

```

```

464 \org@@citex[#1][#2]{\@tempa}}%
465 }{}}

```

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

```

466 \AtBeginDocument{%
467 \ifpackageloaded{cite}{%
468 \def\@citex[#1]#2{%
469 \@safe@activestrue\org@@citex[#1][#2]\@safe@activesfalse}%
470 }{}}

```

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

```

471 \bbl@redefine\nocite#1{%
472 \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

473 \bbl@redefine\bibcite{%
474 \bbl@cite@choice
475 \bibcite}

```

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

```

476 \def\bbl@bibcite#1#2{%
477 \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.

```

478 \def\bbl@cite@choice{%
479 \global\let\bibcite\bbl@bibcite
480 \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
481 \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
482 \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.

```

483 \AtBeginDocument{\bbl@cite@choice}

```

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

```

484 \bbl@redefine\@bibitem#1{%
485 \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
486 \else
487 \let\org@nocite\nocite
488 \let\org@@citex\@citex
489 \let\org@bibcite\bibcite
490 \let\org@@bibitem\@bibitem
491 \fi

```

7.6 Marks

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

```

492 \bbl@trace{Marks}
493 \IfBabelLayout{sectioning}
494   {\ifx\bbl@opt@headfoot\@nnil
495     \g@addto@macro\@resetactivechars{%
496       \set@typeset@protect
497       \expandafter\select@language@x\expandafter{\bbl@main@language}%
498       \let\protect\noexpand
499       \edef\thepage{% TODO. Only with bidi. See also above
500         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
501       \fi}
502   {\ifbbl@single\else
503     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
504     \markright#1{%
505       \bbl@ifblank{#1}%
506       {\org@markright{}}%
507       {\toks@{#1}}%
508       \bbl@exp{%
509         \\org@markright{\\protect\\foreignlanguage{\language}%
510           {\\protect\\bbl@restore@actives\the\toks@}}}%

```

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

`\@mkboth`

```

511   \ifx\@mkboth\markboth
512     \def\bbl@tempc{\let\@mkboth\markboth}
513   \else
514     \def\bbl@tempc{}
515   \fi
516   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
517   \markboth#1#2{%
518     \protected@edef\bbl@tempb##1{%
519       \protect\foreignlanguage
520       {\language}{\protect\bbl@restore@actives##1}}%
521     \bbl@ifblank{#1}%
522     {\toks@{}}%
523     {\toks@\expandafter{\bbl@tempb{#1}}}%
524     \bbl@ifblank{#2}%
525     {\@temptokena{}}%
526     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
527     \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
528     \bbl@tempc
529   \fi} % end ifbbl@single, end \IfBabelLayout

```

7.7 Preventing clashes with other packages

7.7.1 ifthen

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

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

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

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

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

```
530 \bbl@trace{Preventing clashes with other packages}
531 \bbl@xin@{R}\bbl@opt@safe
532 \ifin@
533 \AtBeginDocument{%
534   \@ifpackageloaded{ifthen}{%
535     \bbl@redefine@long\ifthenelse#1#2#3{%
536       \let\bbl@temp@pref\pageref
537       \let\pageref\org@pageref
538       \let\bbl@temp@ref\ref
539       \let\ref\org@ref
540       \@safe@activestrue
541       \org@ifthenelse{#1}%
542         {\let\pageref\bbl@temp@pref
543          \let\ref\bbl@temp@ref
544          \@safe@activesfalse
545          #2}%
546         {\let\pageref\bbl@temp@pref
547          \let\ref\bbl@temp@ref
548          \@safe@activesfalse
549          #3}%
550     }%
551   }%
552 }
```

7.7.2 varioref

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

```
553 \AtBeginDocument{%
554   \@ifpackageloaded{varioref}{%
555     \bbl@redefine\@@vpageref#1[#2]#3{%
556       \@safe@activestrue
557       \org@@@vpageref{#1}[#2]{#3}%
558       \@safe@activesfalse}%
559     \bbl@redefine\vrefpagemum#1#2{%
560       \@safe@activestrue
561       \org\vrefpagemum{#1}{#2}%
562     }%
563   }%
564 }
```

```
562 \@safe@activesfalse}%
```

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

```
563 \expandafter\def\csname Ref \endcsname#1{%
564   \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
565   }{}%
566 }
567 \fi
```

7.7.3 `hhline`

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

```
568 \AtEndOfPackage{%
569   \AtBeginDocument{%
570     \@ifpackageloaded{hhline}%
571     {\expandafter\ifx\csname normal@char\string\endcsname\relax
572       \else
573         \makeatletter
574         \def\@currname{hhline}\input{hhline.sty}\makeatother
575       \fi}%
576     {}}}
```

7.7.4 `hyperref`

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

```
577 % \AtBeginDocument{%
578 %   \ifx\pdfstringdefDisableCommands\@undefined\else
579 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
580 %   \fi}
```

7.7.5 `fancyhdr`

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

```
581 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
582   \lowercase{\foreignlanguage{#1}}}
```

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

```
583 \def\substitutefontfamily#1#2#3{%
```

```

584 \lowercase{\immediate\openout15=#1#2.fd\relax}%
585 \immediate\write15{%
586   \string\ProvidesFile{#1#2.fd}%
587   [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
588   \space generated font description file]^{}
589   \string\DeclareFontFamily{#1}{#2}{}^{}
590   \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^{}
591   \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^{}
592   \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^{}
593   \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^{}
594   \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^{}
595   \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^{}
596   \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^{}
597   \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^{}
598   }%
599 \closeout15
600 }
601 \@onlypreamble\substitutefontfamily

```

7.8 Encoding and fonts

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

`\ensureascii`

```

602 \bbl@trace{Encoding and fonts}
603 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
604 \newcommand\BabelNonText{TS1,T3,TS3}
605 \let\org@TeX\TeX
606 \let\org@LaTeX\LaTeX
607 \let\ensureascii\@firstofone
608 \AtBeginDocument{%
609   \in@false
610   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
611     \ifin@
612       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
613     \fi}%
614   \ifin@ % if a text non-ascii has been loaded
615     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
616     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
617     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
618     \def\bbl@temp#1@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
619     \def\bbl@tempc#1ENC.DEF#2\@{\@%
620       \ifx\@empty#2\else
621         \bbl@ifunset{T#1}%
622         {}%
623         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
624       \ifin@
625         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
626         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
627       \else
628         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
629       \fi}%

```

```

630     \fi}%
631     \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
632     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
633     \ifin@else
634         \edef\ensureascii#1{%
635             \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
636     \fi
637 \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.

```

638 \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.

```

639 \AtBeginDocument{%
640     \ifpackageloaded{fontspec}%
641     {\xdef\latinencoding{%
642         \ifx\UTFfencname\@undefined
643             EU\ifcase\bbl@engine\or2\or1\fi
644         \else
645             \UTFfencname
646         \fi}}%
647     {\gdef\latinencoding{OT1}%
648         \ifx\cf@encoding\bbl@t@one
649             \xdef\latinencoding{\bbl@t@one}%
650         \else
651             \ifx\@fontenc@load@list\@undefined
652                 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
653             \else
654                 \def\@elt#1{,#1,}%
655                 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
656                 \let\@elt\relax
657                 \bbl@xin@{,T1,}\bbl@tempa
658                 \ifin@
659                     \xdef\latinencoding{\bbl@t@one}%
660                 \fi
661             \fi
662         \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.

```

663 \DeclareRobustCommand{\latintext}{%
664     \fontencoding{\latinencoding}\selectfont
665     \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.

```

666 \ifx\@undefined\DeclareTextFontCommand
667     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}

```

```

668 \else
669   \DeclareTextFontCommand{\textlatin}{\latintext}
670 \fi

```

7.9 Basic bidi support

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

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

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

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

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

```

671 \ifodd\bb1@engine
672   \def\bb1@activate@preotf{%
673     \let\bb1@activate@preotf\relax % only once
674     \directlua{
675       Babel = Babel or {}
676       %
677       function Babel.pre_otfload_v(head)
678         if Babel.numbers and Babel.digits_mapped then
679           head = Babel.numbers(head)
680         end
681         if Babel.bidi_enabled then
682           head = Babel.bidi(head, false, dir)
683         end
684         return head
685       end
686       %
687       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
688         if Babel.numbers and Babel.digits_mapped then
689           head = Babel.numbers(head)
690         end
691         if Babel.bidi_enabled then
692           head = Babel.bidi(head, false, dir)
693         end
694         return head

```



```

695     end
696     %
697     luatexbase.add_to_callback('pre_linebreak_filter',
698         Babel.pre_otfload_v,
699         'Babel.pre_otfload_v',
700         luatexbase.priority_in_callback('pre_linebreak_filter',
701             'luaotfload.node_processor') or nil)
702     %
703     luatexbase.add_to_callback('hpack_filter',
704         Babel.pre_otfload_h,
705         'Babel.pre_otfload_h',
706         luatexbase.priority_in_callback('hpack_filter',
707             'luaotfload.node_processor') or nil)
708     }}
709 \fi

```

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

```

710 \bbl@trace{Loading basic (internal) bidi support}
711 \ifodd\bbl@engine
712   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
713     \let\bbl@beforeforeign\leavevmode
714     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
715     \RequirePackage{luatexbase}
716     \bbl@activate@preotf
717     \directlua{
718       require('babel-data-bidi.lua')
719       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
720         require('babel-bidi-basic.lua')
721       \or
722         require('babel-bidi-basic-r.lua')
723     \fi}
724     % TODO - to locale_props, not as separate attribute
725     \newattribute\bbl@attr@dir
726     % TODO. I don't like it, hackish:
727     \bbl@exp{\output{\bodydir\pagedir\the\output}}
728     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
729   \fi\fi
730 \else
731   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
732     \bbl@error
733       {The bidi method `basic' is available only in\\%
734         luatex. I'll continue with `bidi=default', so\\%
735         expect wrong results}%
736       {See the manual for further details.}%
737     \let\bbl@beforeforeign\leavevmode
738     \AtEndOfPackage{%
739       \EnableBabelHook{babel-bidi}%
740       \bbl@xebidipar}
741   \fi\fi
742   \def\bbl@loadxebidi#1{%
743     \ifx\RTLfootnotetext\@undefined
744       \AtEndOfPackage{%
745         \EnableBabelHook{babel-bidi}%
746         \ifx\fontspec\@undefined
747           \usepackage{fontspec}% bidi needs fontspec
748         \fi
749         \usepackage#1{bidi}}%
750   \fi}

```

```

751 \ifnum\bb1@bidimode>200
752   \ifcase\expandafter\@gobbletwo\the\bb1@bidimode\or
753     \bb1@tentative{bidi=bidi}
754     \bb1@loadxebidi{}
755   \or
756     \bb1@tentative{bidi=bidi-r}
757     \bb1@loadxebidi{[rldocument]}
758   \or
759     \bb1@tentative{bidi=bidi-l}
760     \bb1@loadxebidi{}
761   \fi
762 \fi
763 \fi
764 \ifnum\bb1@bidimode=\@ne
765   \let\bb1@beforeforeign\leavevmode
766   \ifodd\bb1@engine
767     \newattribute\bb1@attr@dir
768     \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
769   \fi
770   \AtEndOfPackage{%
771     \EnableBabelHook{babel-bidi}%
772   \ifodd\bb1@engine\else
773     \bb1@xebidipar
774   \fi}
775 \fi

```

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

```

776 \bb1@trace{Macros to switch the text direction}
777 \def\bb1@alscripts{,Arabic,Syriac,Thaana,}
778 \def\bb1@rscripts{% TODO. Base on codes ??
779   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
780   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
781   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
782   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
783   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
784   Old South Arabian,}%
785 \def\bb1@provide@dirs#1{%
786   \bb1@xin@{\csname bbl@sname@#1\endcsname}{\bb1@alscripts\bb1@rscripts}%
787   \ifin@
788     \global\bb1@csarg\chardef{wdir@#1}\@ne
789     \bb1@xin@{\csname bbl@sname@#1\endcsname}{\bb1@alscripts}%
790   \ifin@
791     \global\bb1@csarg\chardef{wdir@#1}\tw@ % useless in xetex
792   \fi
793   \else
794     \global\bb1@csarg\chardef{wdir@#1}\z@
795   \fi
796   \ifodd\bb1@engine
797     \bb1@csarg\ifcase{wdir@#1}%
798       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
799     \or
800       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
801     \or
802       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
803     \fi
804   \fi}
805 \def\bb1@switchdir{%
806   \bb1@ifunset{bbl@lsys@\languagenam}{\bb1@provide@lsys{\languagenam}}{}}%

```

```

807 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
808 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
809 \def\bbl@setdirs#1{% TODO - math
810 \ifcase\bbl@select@type % TODO - strictly, not the right test
811 \bbl@bodydir{#1}%
812 \bbl@pardir{#1}%
813 \fi
814 \bbl@textdir{#1}}
815 % TODO. Only if \bbl@bidimode > 0?:
816 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
817 \DisableBabelHook{babel-bidi}

```

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

```

818 \ifodd\bbl@engine % luatex=1
819 \chardef\bbl@thetextdir\z@
820 \chardef\bbl@thepardir\z@
821 \def\bbl@getluadir#1{%
822 \directlua{
823 if tex.#1dir == 'TLT' then
824 tex.sprint('0')
825 elseif tex.#1dir == 'TRT' then
826 tex.sprint('1')
827 end}}
828 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=textdir.. 3=0 lr/1 rl
829 \ifcase#3\relax
830 \ifcase\bbl@getluadir{#1}\relax\else
831 #2 TLT\relax
832 \fi
833 \else
834 \ifcase\bbl@getluadir{#1}\relax
835 #2 TRT\relax
836 \fi
837 \fi}
838 \def\bbl@textdir#1{%
839 \bbl@setluadir{text}\textdir{#1}%
840 \chardef\bbl@thetextdir#1\relax
841 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
842 \def\bbl@pardir#1{%
843 \bbl@setluadir{par}\pardir{#1}%
844 \chardef\bbl@thepardir#1\relax}
845 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
846 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
847 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
848 % Sadly, we have to deal with boxes in math with basic.
849 % Activated every math with the package option bidi=:
850 \def\bbl@mathboxdir{%
851 \ifcase\bbl@thetextdir\relax
852 \everyhbox{\textdir TLT\relax}%
853 \else
854 \everyhbox{\textdir TRT\relax}%
855 \fi}
856 \frozen@everymath\expandafter{%
857 \expandafter\bbl@mathboxdir\the\frozen@everymath}
858 \frozen@everydisplay\expandafter{%
859 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
860 \else % pdftex=0, xetex=2
861 \newcount\bbl@dirlevel
862 \chardef\bbl@thetextdir\z@
863 \chardef\bbl@thepardir\z@

```

```

864 \def\bbl@textdir#1{%
865   \ifcase#1\relax
866     \chardef\bbl@thetextdir\z@
867     \bbl@textdir@i\beginL\endL
868   \else
869     \chardef\bbl@thetextdir\@ne
870     \bbl@textdir@i\beginR\endR
871   \fi}
872 \def\bbl@textdir@i#1#2{%
873   \ifhmode
874     \ifnum\currentgrouplevel>\z@
875       \ifnum\currentgrouplevel=\bbl@dirlevel
876         \bbl@error{Multiple bidi settings inside a group}%
877         {I'll insert a new group, but expect wrong results.}%
878         \bgroup\aftergroup#2\aftergroup\egroup
879       \else
880         \ifcase\currentgrouptype\or % 0 bottom
881           \aftergroup#2% 1 simple {}
882         \or
883           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
884         \or
885           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
886         \or\or\or % vbox vtop align
887         \or
888           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
889         \or\or\or\or\or\or\or % output math disc insert vcent mathchoice
890         \or
891           \aftergroup#2% 14 \begingroup
892         \else
893           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
894         \fi
895       \fi
896       \bbl@dirlevel\currentgrouplevel
897     \fi
898     #1%
899   \fi}
900 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
901 \let\bbl@bodydir\@gobble
902 \let\bbl@pagedir\@gobble
903 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

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

```

904 \def\bbl@xebidipar{%
905   \let\bbl@xebidipar\relax
906   \TeXeTstate\@ne
907   \def\bbl@xeverypar{%
908     \ifcase\bbl@thepardir
909       \ifcase\bbl@thetextdir\else\beginR\fi
910     \else
911       {\setbox\z@\lastbox\beginR\box\z@}%
912     \fi}%
913   \let\bbl@severypar\everypar
914   \newtoks\everypar
915   \everypar=\bbl@severypar
916   \bbl@severypar{\bbl@xeverypar\the\everypar}}
917 \ifnum\bbl@bidimode>200
918   \let\bbl@textdir@i\@gobbletwo

```

```

919 \let\bbl@xebidipar\@empty
920 \AddBabelHook{bidi}{foreign}{%
921   \def\bbl@tempa{\def\BabelText####1}%
922   \ifcase\bbl@thetextdir
923     \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
924   \else
925     \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
926   \fi}
927 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
928 \fi
929 \fi

```

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

```

930 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
931 \AtBeginDocument{%
932   \ifx\pdfstringdefDisableCommands\@undefined\else
933     \ifx\pdfstringdefDisableCommands\relax\else
934       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
935     \fi
936   \fi}

```

7.10 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `nor sk.cfg` will be loaded when the language definition file `nor sk.ldf` is loaded.

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

```

937 \bbl@trace{Local Language Configuration}
938 \ifx\loadlocalcfg\@undefined
939   \@ifpackagewith{babel}{noconfigs}%
940   {\let\loadlocalcfg\@gobble}%
941   {\def\loadlocalcfg#1{%
942     \InputIfFileExists{#1.cfg}%
943     {\typeout{*****^J%
944               * Local config file #1.cfg used^^J%
945               *}}}%
946     \@empty}}
947 \fi

```

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

```

948 \ifx\@unexpandable@protect\@undefined
949   \def\@unexpandable@protect{\noexpand\protect\noexpand}
950   \long\def\protected@write#1#2#3{%
951     \begingroup
952       \let\thepage\relax
953       #2%
954       \let\protect\@unexpandable@protect
955       \edef\reserved@a{\write#1{#3}}%
956       \reserved@a
957     \endgroup
958   \if@nobreak\ifvmode\nobreak\fi\fi}
959 \fi
960 %
961 % \subsection{Language options}

```

```

962 %
963 % Languages are loaded when processing the corresponding option
964 % \textit{except} if a |main| language has been set. In such a
965 % case, it is not loaded until all options has been processed.
966 % The following macro inputs the ldf file and does some additional
967 % checks (|\input| works, too, but possible errors are not caught).
968 %
969 % \begin{macrocode}
970 \bbl@trace{Language options}
971 \let\bbl@afterlang\relax
972 \let\BabelModifiers\relax
973 \let\bbl@loaded@empty
974 \def\bbl@load@language#1{%
975   \InputIfFileExists{#1.ldf}%
976   {\edef\bbl@loaded{\CurrentOption
977     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
978     \expandafter\let\expandafter\bbl@afterlang
979       \csname\CurrentOption.ldf-h@k\endcsname
980     \expandafter\let\expandafter\BabelModifiers
981       \csname bbl@mod@\CurrentOption\endcsname}%
982   {\bbl@error{%
983     Unknown option '\CurrentOption'. Either you misspelled it\\%
984     or the language definition file \CurrentOption.ldf was not found}}%
985     Valid options are: shorthands=, KeepShorthandsActive,\\%
986     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
987     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.

```

988 \def\bbl@try@load@lang#1#2#3{%
989   \IfFileExists{\CurrentOption.ldf}%
990   {\bbl@load@language{\CurrentOption}}%
991   {#1\bbl@load@language{#2}#3}}
992 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}}
993 \DeclareOption{hebrew}{%
994   \input{rlbabel.def}%
995   \bbl@load@language{hebrew}}
996 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}
997 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}
998 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}}
999 \DeclareOption{polutonikogreek}{%
1000   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1001 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}
1002 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}}
1003 \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.

```

1004 \ifx\bbl@opt@config@nnil
1005   \ifpackagewith{babel}{noconfigs}}%
1006   {\InputIfFileExists{bblopts.cfg}%
1007     {\typeout{*****^J%
1008       * Local config file bblopts.cfg used^^J%
1009       *}}%

```

```

1010     {}}%
1011 \else
1012   \InputIfFileExists{\bbl@opt@config.cfg}%
1013   {\typeout{*****^J%
1014             * Local config file \bbl@opt@config.cfg used^^J%
1015             *}}%
1016   {\bbl@error{%
1017     Local config file '\bbl@opt@config.cfg' not found}{%
1018     Perhaps you misspelled it.}}%
1019 \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages (note this list also contains the language given with `main`). If not declared above, the names of the option and the file are the same.

```

1020 \bbl@for\bbl@tempa\bbl@language@opts{%
1021   \bbl@ifunset{ds@\bbl@tempa}%
1022   {\edef\bbl@tempb{%
1023     \noexpand\DeclareOption
1024     {\bbl@tempa}%
1025     {\noexpand\bbl@load@language{\bbl@tempa}}}%
1026   \bbl@tempb}%
1027   \@empty}

```

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

```

1028 \bbl@foreach\@classoptionslist{%
1029   \bbl@ifunset{ds@#1}%
1030   {\IfFileExists{#1.ldf}%
1031    {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1032    {}}%
1033   {}

```

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

```

1034 \ifx\bbl@opt@main\@nnil\else
1035   \expandafter
1036   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1037   \DeclareOption{\bbl@opt@main}{}
1038 \fi

```

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

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

```

1039 \def\AfterBabelLanguage#1{%
1040   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
1041   \DeclareOption*{}
1042 \ProcessOptions*

```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning is raised if the main language is not the same as the last named one, or if the value of the key `main` is not a language. Then execute directly the option (because it could be used only in `main`). After loading all languages, we deactivate `\AfterBabelLanguage`.

```

1043 \bbl@trace{Option 'main'}
1044 \ifx\bbl@opt@main\@nnil

```

```

1045 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1046 \let\bbl@tempc\@empty
1047 \bbl@for\bbl@tempb\bbl@tempa{%
1048   \bbl@xin@{\,\bbl@tempb,}{\,\bbl@loaded,}%
1049   \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
1050   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1051   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1052   \ifx\bbl@tempb\bbl@tempc\else
1053     \bbl@warning{%
1054       Last declared language option is '\bbl@tempc',\%
1055       but the last processed one was '\bbl@tempb'.\%
1056       The main language cannot be set as both a global\%
1057       and a package option. Use 'main=\bbl@tempc' as\%
1058       option. Reported}%
1059   \fi
1060 \else
1061   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1062   \ExecuteOptions{\bbl@opt@main}
1063   \DeclareOption*{}
1064   \ProcessOptions*
1065 \fi
1066 \def\AfterBabelLanguage{%
1067   \bbl@error
1068   {Too late for \string\AfterBabelLanguage}%
1069   {Languages have been loaded, so I can do nothing}}

```

In order to catch the case where the user forgot to specify a language we check whether `\bbl@main@language`, has become defined. If not, no language has been loaded and an error message is displayed.

```

1070 \ifx\bbl@main@language\@undefined
1071   \bbl@info{%
1072     You haven't specified a language. I'll use 'nil'\%
1073     as the main language. Reported}
1074   \bbl@load@language{nil}
1075 \fi
1076 \</package>
1077 \<core>

```

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

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

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

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

8.1 Tools

```

1078 \ifx\ldf@quit\@undefined\else
1079 \endinput\fi % Same line!
1080 \<\<Make sure ProvidesFile is defined>\>

```



```
1081 \ProvidesFile{babel.def}[\langle date \rangle \langle version \rangle] Babel common definitions]
```

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

`\BabelModifiers` can be set too (but not sure it works).

```
1082 \ifx\AtBeginDocument\@undefined % TODO. change test.
1083   \langle Emulate LaTeX \rangle
1084   \def\language{english}%
1085   \let\bbl@opt@shorthands\@nnil
1086   \def\bbl@ifshorthand#1#2#3{#2}%
1087   \let\bbl@language@opts\@empty
1088   \ifx\babeloptionstrings\@undefined
1089     \let\bbl@opt@strings\@nnil
1090   \else
1091     \let\bbl@opt@strings\babeloptionstrings
1092   \fi
1093   \def\BabelStringsDefault{generic}
1094   \def\bbl@tempa{normal}
1095   \ifx\babeloptionmath\bbl@tempa
1096     \def\bbl@mathnormal{\noexpand\textormath}
1097   \fi
1098   \def\AfterBabelLanguage#1#2{}
1099   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1100   \let\bbl@afterlang\relax
1101   \def\bbl@opt@safe{BR}
1102   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1103   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1104   \expandafter\newif\csname ifbbl@single\endcsname
1105   \chardef\bbl@bidimode\z@
1106 \fi
```

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

```
1107 \ifx\bbl@trace\@undefined
1108   \let\LdfInit\endinput
1109   \def\ProvidesLanguage#1{\endinput}
1110 \endinput\fi % Same line!
```

And continue.

9 Multiple languages

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

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

```
1111 \langle Define core switching macros \rangle
```

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

```
1112 \def\bbl@version{\langle version \rangle}
1113 \def\bbl@date{\langle date \rangle}
1114 \def\adddialect#1#2{%
1115   \global\chardef#1#2\relax
1116   \bbl@usehooks{adddialect}{\{#1\}\{#2\}}%
1117   \begingroup
```

```

1118 \count@#1\relax
1119 \def\bbl@elt##1##2###3###4{%
1120 \ifnum\count@=##2\relax
1121 \bbl@info{\string#1 = using hyphenrules for ##1\\%
1122 (\string\language\the\count@)}%
1123 \def\bbl@elt####1####2####3####4{%
1124 \fi}%
1125 \bbl@cs{languages}%
1126 \endgroup}

```

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

```

1127 \def\bbl@fixname#1{%
1128 \begingroup
1129 \def\bbl@tempe{#1}%
1130 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1131 \bbl@tempd
1132 {\lowercase\expandafter{\bbl@tempd}%
1133 {\uppercase\expandafter{\bbl@tempd}%
1134 \@empty
1135 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1136 \uppercase\expandafter{\bbl@tempd}}}%
1137 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1138 \lowercase\expandafter{\bbl@tempd}}}%
1139 \@empty
1140 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1141 \bbl@tempd
1142 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1143 \def\bbl@iflanguage#1{%
1144 \@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.

```

1145 \def\bbl@bcpcase#1#2#3#4\@@#5{%
1146 \ifx\@empty#3%
1147 \uppercase{\def#5{#1#2}}%
1148 \else
1149 \uppercase{\def#5{#1}}%
1150 \lowercase{\edef#5{#5#2#3#4}}%
1151 \fi}
1152 \def\bbl@bcpllookup#1-#2-#3-#4\@@{%
1153 \let\bbl@bcp\relax
1154 \lowercase{\def\bbl@tempa{#1}}%
1155 \ifx\@empty#2%
1156 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1157 \else\ifx\@empty#3%
1158 \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1159 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%

```

```

1160     {\edef\bbbl@bcp{\bbbl@tempa-\bbbl@tempb}}}%
1161     {}}%
1162     \ifx\bbbl@bcp\relax
1163       \IfFileExists{babel-\bbbl@tempa.ini}{\let\bbbl@bcp\bbbl@tempa}{}%
1164     \fi
1165   \else
1166     \bbbl@bcpcase#2\@empty\@empty\@@\bbbl@tempb
1167     \bbbl@bcpcase#3\@empty\@empty\@@\bbbl@tempc
1168     \IfFileExists{babel-\bbbl@tempa-\bbbl@tempb-\bbbl@tempc.ini}%
1169       {\edef\bbbl@bcp{\bbbl@tempa-\bbbl@tempb-\bbbl@tempc}}}%
1170     {}}%
1171     \ifx\bbbl@bcp\relax
1172       \IfFileExists{babel-\bbbl@tempa-\bbbl@tempc.ini}%
1173       {\edef\bbbl@bcp{\bbbl@tempa-\bbbl@tempc}}}%
1174     {}}%
1175   \fi
1176   \ifx\bbbl@bcp\relax
1177     \IfFileExists{babel-\bbbl@tempa-\bbbl@tempc.ini}%
1178     {\edef\bbbl@bcp{\bbbl@tempa-\bbbl@tempc}}}%
1179   {}}%
1180 \fi
1181 \ifx\bbbl@bcp\relax
1182   \IfFileExists{babel-\bbbl@tempa.ini}{\let\bbbl@bcp\bbbl@tempa}{}%
1183 \fi
1184 \fi\fi}
1185 \let\bbbl@autoload@options\@empty
1186 \let\bbbl@initoload\relax
1187 \def\bbbl@provide@locale{%
1188   \ifx\babelprovide\@undefined
1189     \bbbl@error{For a language to be defined on the fly 'base'\\%
1190       is not enough, and the whole package must be\\%
1191       loaded. Either delete the 'base' option or\\%
1192       request the languages explicitly}%
1193     {See the manual for further details.}%
1194   \fi
1195 % TODO. Option to search if loaded, with \LocaleForEach
1196 \let\bbbl@auxname\language\name % Still necessary. TODO
1197 \bbbl@ifunset{bbbl@bcp@map@\language\name}{}% Move uplevel??
1198 {\edef\language\name{\@nameuse{bbbl@bcp@map@\language\name}}}%
1199 \ifbbbl@bcp@allowed
1200   \expandafter\ifx\csname date\language\name\endcsname\relax
1201     \expandafter
1202     \bbbl@bcp@lookup\language\name-\@empty-\@empty-\@empty\@@
1203     \ifx\bbbl@bcp\relax\else % Returned by \bbbl@bcp@lookup
1204       \edef\language\name{\bbbl@bcp@prefix\bbbl@bcp}%
1205       \edef\localename{\bbbl@bcp@prefix\bbbl@bcp}%
1206       \expandafter\ifx\csname date\language\name\endcsname\relax
1207         \let\bbbl@initoload\bbbl@bcp
1208         \bbbl@exp{\@babelprovide[\bbbl@autoload@bcptoptions]{\language\name}}%
1209         \let\bbbl@initoload\relax
1210       \fi
1211       \bbbl@csarg\xdef{bcp@map@\bbbl@bcp}{\localename}%
1212     \fi
1213   \fi
1214 \fi
1215 \expandafter\ifx\csname date\language\name\endcsname\relax
1216   \IfFileExists{babel-\language\name.tex}%
1217   {\bbbl@exp{\@babelprovide[\bbbl@autoload@options]{\language\name}}}%
1218   {}}%

```

```
1219 \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.

```
1220 \def\iflanguage#1{%
1221   \bbl@iflanguage{#1}{%
1222     \ifnum\csname l@#1\endcsname=\language
1223       \expandafter\@firstoftwo
1224     \else
1225       \expandafter\@secondoftwo
1226     \fi}}
```

9.1 Selecting the language

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

```
1227 \let\bbl@select@type\z@
1228 \edef\selectlanguage{%
1229   \noexpand\protect
1230   \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`.

```
1231 \ifx\@undefined\protect\let\protect\relax\fi
```

The following definition is preserved for backwards compatibility. It is related to a trick for 2.09.

```
1232 \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.

```
1233 \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`

```
1234 \def\bbl@push@language{%
1235   \ifx\language\@undefined\else
1236     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1237   \fi}
```

Retrieving information from the stack is a little bit less simple, as we need to remove the element from the stack while storing it in the macro `\language`. For this we first define a helper function.

`\bbl@pop@lang` This macro stores its first element (which is delimited by the ‘+’-sign) in `\language` and stores the rest of the string (delimited by ‘-’) in its third argument.

```
1238 \def\bbl@pop@lang#1+#2#3{%
1239   \edef\language{#1}\xdef#3{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before `\bbl@pop@lang` is executed \TeX first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a ‘+’-sign (zero language names won’t occur as this macro will only be called after something has been pushed on the stack) followed by the ‘&’-sign and finally the reference to the stack.

```
1240 \let\bbl@ifrestoring\@secondoftwo
1241 \def\bbl@pop@language{%
1242   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1243   \let\bbl@ifrestoring\@firstoftwo
1244   \expandafter\bbl@set@language\expandafter{\language}%
1245   \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).

```
1246 \chardef\localeid\z@
1247 \def\bbl@id@last{0} % No real need for a new counter
1248 \def\bbl@id@assign{%
1249   \bbl@ifunset{bbl@id@@\language}%
1250   {\count@bbl@id@last\relax
1251     \advance\count@\@ne
1252     \bbl@csarg\chardef{id@@\language}\count@
1253     \edef\bbl@id@last{\the\count@}%
1254     \ifcase\bbl@engine\or
1255       \directlua{
1256         Babel = Babel or {}
1257         Babel.locale_props = Babel.locale_props or {}
1258         Babel.locale_props[\bbl@id@last] = {}
1259         Babel.locale_props[\bbl@id@last].name = '\language'
1260       }%
1261     \fi}%
1262   }%
1263   \chardef\localeid\bbl@c{id@}}
```

The unprotected part of `\selectlanguage`.

```
1264 \expandafter\def\csname selectlanguage \endcsname#1{%
1265   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@ \fi
1266   \bbl@push@language
1267   \aftergroup\bbl@pop@language
1268   \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.

```

1269 \def\BabelContentsFiles{toc,lof,lot}
1270 \def\bbl@set@language#1{% from selectlanguage, pop@
1271 % The old buggy way. Preserved for compatibility.
1272 \edef\language{%
1273   \ifnum\escapechar=\expandafter`\string#1\@empty
1274   \else\string#1\@empty\fi}%
1275 \ifcat\relax\noexpand#1%
1276   \expandafter\ifx\csname date\language\endcsname\relax
1277   \edef\language{#1}%
1278   \let\locale\language
1279 \else
1280   \bbl@info{Using '\string\language' instead of 'language' is\\%
1281             deprecated. If what you want is to use a\\%
1282             macro containing the actual locale, make\\%
1283             sure it does not not match any language.\\%
1284             Reported}%
1285   I'll\\%
1286   try to fix '\string\locale', but I cannot promise\\%
1287   anything. Reported}%
1288   \ifx\scantokens\undefined
1289     \def\locale{??}%
1290   \else
1291     \scantokens\expandafter{\expandafter
1292       \def\expandafter\locale\expandafter{\language}}%
1293   \fi
1294 \fi
1295 \else
1296   \def\locale{#1}% This one has the correct catcodes
1297 \fi
1298 \select@language{\language}%
1299 % write to aux
1300 \expandafter\ifx\csname date\language\endcsname\relax\else
1301   \if@files
1302     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1303       \protected@write\@auxout{\string\babel@aux{\bbl@auxname}}}%
1304     \fi
1305     \bbl@usehooks{write}}}%
1306   \fi
1307 \fi}
1308 %
1309 \newif\ifbbl@bcpallowed
1310 \bbl@bcpallowedfalse
1311 \def\select@language#1{% from set@, babel@aux
1312 % set hymap
1313 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
1314 % set name
1315 \edef\language{#1}%
1316 \bbl@fixname\language
1317 % TODO. name@map must be here?
1318 \bbl@provide@locale
1319 \bbl@iflanguage\language{%
1320   \expandafter\ifx\csname date\language\endcsname\relax

```

```

1321 \bbl@error
1322 {Unknown language '\language'. Either you have\\%
1323 misspelled its name, it has not been installed,\\%
1324 or you requested it in a previous run. Fix its name,\\%
1325 install it or just rerun the file, respectively. In\\%
1326 some cases, you may need to remove the aux file}%
1327 {You may proceed, but expect wrong results}%
1328 \else
1329 % set type
1330 \let\bbl@select@type\z@
1331 \expandafter\bbl@switch\expandafter{\language}%
1332 \fi}}
1333 \def\babel@aux#1#2{%
1334 \select@language{#1}%
1335 \bbl@foreach\BabelContentsFiles{%
1336 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1337 \def\babel@toc#1#2{%
1338 \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.

```

1339 \newif\ifbbl@usedategroup
1340 \def\bbl@switch#1{% from select@, foreign@
1341 % make sure there is info for the language if so requested
1342 \bbl@ensureinfo{#1}%
1343 % restore
1344 \originalTeX
1345 \expandafter\def\expandafter\originalTeX\expandafter{%
1346 \csname noextras#1\endcsname
1347 \let\originalTeX\@empty
1348 \babel@beginsave}%
1349 \bbl@usehooks{afterreset}}}%
1350 \languageshorthands{none}%
1351 % set the locale id
1352 \bbl@id@assign
1353 % switch captions, date
1354 % No text is supposed to be added here, so we remove any
1355 % spurious spaces.
1356 \bbl@bsphack
1357 \ifcase\bbl@select@type
1358 \csname captions#1\endcsname\relax
1359 \csname date#1\endcsname\relax
1360 \else
1361 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1362 \ifin@

```

```

1363      \csname captions#1\endcsname\relax
1364      \fi
1365      \bbl@xin@{,date,}{,\bbl@select@opts,}%
1366      \ifin@ % if \foreign... within \<lang>date
1367      \csname date#1\endcsname\relax
1368      \fi
1369      \fi
1370      \bbl@esphack
1371      % switch extras
1372      \bbl@usehooks{beforeextras}{}%
1373      \csname extras#1\endcsname\relax
1374      \bbl@usehooks{afterextras}{}%
1375      % > babel-ensure
1376      % > babel-sh-<short>
1377      % > babel-bidi
1378      % > babel-fontspec
1379      % hyphenation - case mapping
1380      \ifcase\bbl@opt@hyphenmap\or
1381      \def\BabelLower##1##2{\lccode##1=##2\relax}%
1382      \ifnum\bbl@hymapsel>4\else
1383      \csname\language\name @bbl@hyphenmap\endcsname
1384      \fi
1385      \chardef\bbl@opt@hyphenmap\z@
1386      \else
1387      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1388      \csname\language\name @bbl@hyphenmap\endcsname
1389      \fi
1390      \fi
1391      \global\let\bbl@hymapsel\@cclv
1392      % hyphenation - patterns
1393      \bbl@patterns{#1}%
1394      % hyphenation - mins
1395      \babel@savevariable\lefthyphenmin
1396      \babel@savevariable\righthyphenmin
1397      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1398      \set@hyphenmins\tw@\thr@\@relax
1399      \else
1400      \expandafter\expandafter\expandafter\set@hyphenmins
1401      \csname #1hyphenmins\endcsname\relax
1402      \fi}

```

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

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

```

1403 \long\def\otherlanguage#1{%
1404   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\@fi
1405   \csname selectlanguage \endcsname{#1}%
1406   \ignorespaces}

```

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

```

1407 \long\def\endotherlanguage{%
1408   \global\@ignoretrue\ignorespaces}

```

`otherlanguage*` The `otherlanguage` environment is meant to be used when a large part of text from a

different language needs to be typeset, but without changing the translation of words such as ‘figure’. This environment makes use of `\foreign@language`.

```

1409 \expandafter\def\csname otherlanguage*\endcsname{%
1410   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1411 \def\bbl@otherlanguage@s[#1]#2{%
1412   \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
1413   \def\bbl@select@opts{#1}%
1414   \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”.

```

1415 \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.

```

1416 \providecommand\bbl@beforeforeign{}
1417 \edef\foreignlanguage{%
1418   \noexpand\protect
1419   \expandafter\noexpand\csname foreignlanguage \endcsname}
1420 \expandafter\def\csname foreignlanguage \endcsname{%
1421   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1422 \providecommand\bbl@foreign@x[3][]{%
1423   \begingroup
1424     \def\bbl@select@opts{#1}%
1425     \let\BabelText\@firstofone
1426     \bbl@beforeforeign
1427     \foreign@language{#2}%
1428     \bbl@usehooks{foreign}{}%
1429     \BabelText{#3}% Now in horizontal mode!
1430   \endgroup}
1431 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@par
1432   \begingroup
1433     {\par}%
1434     \let\BabelText\@firstofone
1435     \foreign@language{#1}%

```

```

1436 \bbl@usehooks{foreign*}{}%
1437 \bbl@dirparastext
1438 \BabelText{#2}% Still in vertical mode!
1439 {\par}%
1440 \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`.

```

1441 \def\foreign@language#1{%
1442 % set name
1443 \edef\language#1}%
1444 \ifbbl@usedategroup
1445 \bbl@add\bbl@select@opts{,date,}%
1446 \bbl@usedategroupfalse
1447 \fi
1448 \bbl@fixname\language
1449 % TODO. name@map here?
1450 \bbl@provide@locale
1451 \bbl@iflanguage\language{\%
1452 \expandafter\ifx\csname date\language\endcsname\relax
1453 \bbl@warning % TODO - why a warning, not an error?
1454 {Unknown language `#1'. Either you have\\%
1455 misspelled its name, it has not been installed,\\%
1456 or you requested it in a previous run. Fix its name,\\%
1457 install it or just rerun the file, respectively. In\\%
1458 some cases, you may need to remove the aux file.\\%
1459 I'll proceed, but expect wrong results.\\%
1460 Reported}%
1461 \fi
1462 % set type
1463 \let\bbl@select@type\@ne
1464 \expandafter\bbl@switch\expandafter{\language}}

```

`\bbl@patterns` This macro selects the hyphenation patterns by changing the `\language` register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.

It also sets hyphenation exceptions, but only once, because they are global (here `language\lccode`'s has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\babelhyphenation`, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that `:ENC` is taken into account) has been set, then use `\hyphenation` with both global and language exceptions and empty the latter to mark they must not be set again.

```

1465 \let\bbl@hyphlist\@empty
1466 \let\bbl@hyphenation@\relax
1467 \let\bbl@pttnlist\@empty
1468 \let\bbl@patterns@\relax
1469 \let\bbl@hymapsel=\@cclv
1470 \def\bbl@patterns#1{%
1471 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1472 \csname l@#1\endcsname
1473 \edef\bbl@tempa{#1}%
1474 \else
1475 \csname l@#1:\f@encoding\endcsname
1476 \edef\bbl@tempa{#1:\f@encoding}%
1477 \fi
1478 \@expandtwoargs\bbl@usehooks{patterns}{\bbl@tempa}}

```

```

1479 % > luatex
1480 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1481   \begingroup
1482     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1483     \ifin\else
1484       \@expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
1485     \hyphenation{%
1486       \bbl@hyphenation@
1487       \@ifundefined{bbl@hyphenation@#1}%
1488       \@empty
1489       {\space\csname bbl@hyphenation@#1\endcsname}}%
1490     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1491     \fi
1492   \endgroup}}

```

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

```

1493 \def\hyphenrules#1{%
1494   \edef\bbl@tempf{#1}%
1495   \bbl@fixname\bbl@tempf
1496   \bbl@iflanguage\bbl@tempf{%
1497     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1498     \languageshortands{none}%
1499     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1500       \set@hyphenmins\tw@\thr@@\relax
1501     \else
1502       \expandafter\expandafter\expandafter\set@hyphenmins
1503       \csname\bbl@tempf hyphenmins\endcsname\relax
1504     \fi}}
1505 \let\endhyphenrules\@empty

```

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

```

1506 \def\providehyphenmins#1#2{%
1507   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1508     \@namedef{#1hyphenmins}{#2}%
1509   \fi}

```

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

```

1510 \def\set@hyphenmins#1#2{%
1511   \lefthyphenmin#1\relax
1512   \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.

```

1513 \ifx\ProvidesFile\@undefined
1514   \def\ProvidesLanguage#1[#2 #3 #4]{%
1515     \wlog{Language: #1 #4 #3 <#2>}%
1516   }
1517 \else
1518   \def\ProvidesLanguage#1{%

```

```

1519 \begingroup
1520 \catcode`\ 10 %
1521 \@makeother\/%
1522 \@ifnextchar[%
1523   {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1524 \def\@provideslanguage#1[#2]{%
1525   \wlog{Language: #1 #2}%
1526   \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1527   \endgroup}
1528 \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`.

```
1529 \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.

```
1530 \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’:

```

1531 \providecommand\setlocale{%
1532   \bbl@error
1533   {Not yet available}%
1534   {Find an armchair, sit down and wait}}
1535 \let\uselocale\setlocale
1536 \let\locale\setlocale
1537 \let\selectlocale\setlocale
1538 \let\localename\setlocale
1539 \let\textlocale\setlocale
1540 \let\textlanguage\setlocale
1541 \let\language\setlocale

```

9.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.
 When the format knows about `\PackageError` it must be $\LaTeX 2_{\epsilon}$, 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.

```

1542 \edef\bbl@nulllanguage{\string\language=0}
1543 \ifx\PackageError\@undefined % TODO. Move to Plain
1544   \def\bbl@error#1#2{%
1545     \begingroup
1546       \newlinechar=`^^J
1547       \def\{^^J(babel) }%
1548       \errhelp{#2}\errmessage{\{#1}%
1549       \endgroup}

```

```

1550 \def\bbl@warning#1{%
1551   \begingroup
1552     \newlinechar=`^^J
1553     \def\{^^J(babel) }%
1554     \message{\#1}%
1555   \endgroup}
1556 \let\bbl@infowarn\bbl@warning
1557 \def\bbl@info#1{%
1558   \begingroup
1559     \newlinechar=`^^J
1560     \def\{^^J}%
1561     \wlog{#1}%
1562   \endgroup}
1563 \fi
1564 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1565 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1566   \global\@namedef{#2}{\textbf{?#1?}}%
1567   \@nameuse{#2}%
1568   \bbl@warning{%
1569     \@backslashchar#2 not set. Please, define it\\%
1570     after the language has been loaded (typically\\%
1571     in the preamble) with something like:\\%
1572     \string\renewcommand\@backslashchar#2{..}\\%
1573     Reported}}
1574 \def\bbl@tentative{\protect\bbl@tentative@i}
1575 \def\bbl@tentative@i#1{%
1576   \bbl@warning{%
1577     Some functions for '#1' are tentative.\\%
1578     They might not work as expected and their behavior\\%
1579     could change in the future.\\%
1580     Reported}}
1581 \def\@nolanerr#1{%
1582   \bbl@error
1583     {You haven't defined the language #1\space yet.\\%
1584     Perhaps you misspelled it or your installation\\%
1585     is not complete}%
1586     {Your command will be ignored, type <return> to proceed}}
1587 \def\@nopatterns#1{%
1588   \bbl@warning
1589     {No hyphenation patterns were preloaded for\\%
1590     the language '#1' into the format.\\%
1591     Please, configure your TeX system to add them and\\%
1592     rebuild the format. Now I will use the patterns\\%
1593     preloaded for \bbl@nulllanguage\space instead}}
1594 \let\bbl@usehooks\@gobbletwo
1595 \ifx\bbl@onlyswitch\@empty\endinput\fi
1596 % Here ended switch.def

Here ended switch.def.

1597 \ifx\directlua\@undefined\else
1598   \ifx\bbl@luapatterns\@undefined
1599     \input luabel.def
1600   \fi
1601 \fi
1602 <<Basic macros>>
1603 \bbl@trace{Compatibility with language.def}
1604 \ifx\bbl@languages\@undefined
1605   \ifx\directlua\@undefined
1606     \openin1 = language.def % TODO. Remove hardcoded number

```

```

1607 \ifeof1
1608 \closein1
1609 \message{I couldn't find the file language.def}
1610 \else
1611 \closein1
1612 \begingroup
1613 \def\addlanguage#1#2#3#4#5{%
1614 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1615 \global\expandafter\let\csname l@#1\expandafter\endcsname
1616 \csname lang@#1\endcsname
1617 \fi}%
1618 \def\uselanguage#1{%
1619 \input language.def
1620 \endgroup
1621 \fi
1622 \fi
1623 \chardef\l@english\z@
1624 \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.

```

1625 \def\addto#1#2{%
1626 \ifx#1\@undefined
1627 \def#1{#2}%
1628 \else
1629 \ifx#1\relax
1630 \def#1{#2}%
1631 \else
1632 {\toks@\expandafter{#1#2}%
1633 \xdef#1{\the\toks@}}%
1634 \fi
1635 \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?

```

1636 \def\bbl@withactive#1#2{%
1637 \begingroup
1638 \lccode`~=`#2\relax
1639 \lowercase{\endgroup#1~}}

```

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

```

1640 \def\bbl@redefine#1{%
1641 \edef\bbl@tempa{\bbl@stripslash#1}%
1642 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1643 \expandafter\def\csname\bbl@tempa\endcsname}
1644 \@onlypreamble\bbl@redefine

```

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

```
1645 \def\bbl@redefine@long#1{%
1646   \edef\bbl@tempa{\bbl@stripslash#1}%
1647   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1648   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1649 \@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_`.

```
1650 \def\bbl@redefineroobust#1{%
1651   \edef\bbl@tempa{\bbl@stripslash#1}%
1652   \bbl@ifunset{\bbl@tempa\space}%
1653   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1654     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1655   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1656   \namedef{\bbl@tempa\space}%
1657 \@onlypreamble\bbl@redefineroobust
```

9.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```
1658 \bbl@trace{Hooks}
1659 \newcommand\AddBabelHook[3][]{%
1660   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1661   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1662   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1663   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1664     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1665     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1666   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1667 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1668 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1669 \def\bbl@usehooks#1#2{%
1670   \def\bbl@elt##1{%
1671     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1}#2}}%
1672     \bbl@cs{ev@#1@}%
1673     \ifx\language\@undefined\else % Test required for Plain (?)
1674       \def\bbl@elt##1{%
1675         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1676         \bbl@cl{ev@#1}%
1677       \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).

```
1678 \def\bbl@evargs{% <- don't delete this comma
1679   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1680   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1681   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1682   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1683   beforestart=0,language=2}
```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro `\bbl@e@<language>` contains `\bbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1684 \bbl@trace{Defining babelensure}
1685 \newcommand\babelensure[2][{}]{% TODO - revise test files
1686   \AddBabelHook{babel-ensure}{afterextras}{%
1687     \ifcase\bbl@select@type
1688       \bbl@cl{e}%
1689     \fi}%
1690   \begingroup
1691     \let\bbl@ens@include\@empty
1692     \let\bbl@ens@exclude\@empty
1693     \def\bbl@ens@fontenc{\relax}%
1694     \def\bbl@tempb##1{%
1695       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1696     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1697     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1698     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1699     \def\bbl@tempc{\bbl@ensure}%
1700     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1701       \expandafter{\bbl@ens@include}}%
1702     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1703       \expandafter{\bbl@ens@exclude}}%
1704     \toks@\expandafter{\bbl@tempc}%
1705     \bbl@exp{%
1706   \endgroup
1707   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1708 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1709   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1710     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1711       \edef##1{\noexpand\bbl@nocaption
1712         {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1713     \fi
1714     \ifx##1\@empty\else
1715       \in@{##1}{#2}%
1716     \ifin\@else
1717       \bbl@ifunset{bbl@ensure@\language\name}%
1718       {\bbl@exp{%
1719         \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1720           \\\foreignlanguage{\language\name}%
1721           {\ifx\relax#3\else
1722             \\\fontencoding{#3}\selectfont
1723             \fi
1724             #####1}}}%
1725       }%
1726       \toks@\expandafter{##1}%
1727       \edef##1{%
1728         \bbl@csarg\noexpand{ensure@\language\name}%
1729         {\the\toks@}}%

```



```

1730     \fi
1731     \expandafter\bb1@tempb
1732   \fi}%
1733   \expandafter\bb1@tempb\bb1@captionslist\today\@empty
1734   \def\bb1@tempa##1{% elt for include list
1735     \ifx##1\@empty\else
1736       \bb1@csarg\in{\ensure@\language\expandafter}\expandafter{##1}%
1737       \ifin\@else
1738         \bb1@tempb##1\@empty
1739       \fi
1740     \expandafter\bb1@tempa
1741   \fi}%
1742   \bb1@tempa#1\@empty}
1743 \def\bb1@captionslist{%
1744   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1745   \contentsname\listfigurename\listtablename\indexname\figurename
1746   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1747   \alsoname\proofname\glossaryname}

```

9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1748 \bb1@trace{Macros for setting language files up}
1749 \def\bb1@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1750   \let\bb1@screset\@empty
1751   \let\BabelStrings\bb1@opt@string
1752   \let\BabelOptions\@empty
1753   \let\BabelLanguages\relax
1754   \ifx\originalTeX\@undefined
1755     \let\originalTeX\@empty
1756   \else
1757     \originalTeX
1758   \fi}
1759 \def\LdfInit#1#2{%
1760   \chardef\atcatcode=\catcode`\@
1761   \catcode`\@=11\relax
1762   \chardef\eqcatcode=\catcode`\=
1763   \catcode`\==12\relax

```

```

1764 \expandafter\if\expandafter\@backslashchar
1765         \expandafter\@car\string#2\@nil
1766     \ifx#2\@undefined\else
1767         \ldf@quit{#1}%
1768     \fi
1769 \else
1770     \expandafter\ifx\csname#2\endcsname\relax\else
1771         \ldf@quit{#1}%
1772     \fi
1773 \fi
1774 \bbl@ldfinit}

```

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

```

1775 \def\ldf@quit#1{%
1776     \expandafter\main@language\expandafter{#1}%
1777     \catcode`\@=\atcatcode \let\atcatcode\relax
1778     \catcode`\==\eqcatcode \let\eqcatcode\relax
1779     \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.

```

1780 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1781     \bbl@afterlang
1782     \let\bbl@afterlang\relax
1783     \let\BabelModifiers\relax
1784     \let\bbl@screset\relax}%
1785 \def\ldf@finish#1{%
1786     \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1787         \loadlocalcfg{#1}%
1788     \fi
1789     \bbl@afterldf{#1}%
1790     \expandafter\main@language\expandafter{#1}%
1791     \catcode`\@=\atcatcode \let\atcatcode\relax
1792     \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1793 \@onlypreamble\LdfInit
1794 \@onlypreamble\ldf@quit
1795 \@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.

```

1796 \def\main@language#1{%
1797     \def\bbl@main@language{#1}%
1798     \let\language\name\bbl@main@language % TODO. Set localename
1799     \bbl@id@assign
1800     \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`.

```

1801 \def\bbl@beforestart{%
1802   \bbl@usehooks{beforestart}{}%
1803   \global\let\bbl@beforestart\relax}
1804 \AtBeginDocument{%
1805   \@nameuse{bbl@beforestart}%
1806   \if@filesw
1807     \providecommand\babel@aux[2]{}%
1808     \immediate\write\@mainaux{%
1809       \string\providecommand\string\babel@aux[2]{}%
1810       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1811   \fi
1812   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1813   \ifbbl@single % must go after the line above.
1814     \renewcommand\selectlanguage[1]{}%
1815     \renewcommand\foreignlanguage[2]{#2}%
1816     \global\let\babel@aux\@gobbletwo % Also as flag
1817   \fi
1818   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1819 \def\select@language@x#1{%
1820   \ifcase\bbl@select@type
1821     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1822   \else
1823     \select@language{#1}%
1824   \fi}

```

9.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if `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.

```

1825 \bbl@trace{Shorhands}
1826 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1827   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1828   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1829   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1830     \begingroup
1831       \catcode`#1\active
1832       \nfss@catcodes
1833       \ifnum\catcode`#1=\active
1834         \endgroup
1835         \bbl@add\nfss@catcodes{\@makeother#1}%
1836       \else
1837         \endgroup
1838       \fi
1839   \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.

```

1840 \def\bbl@remove@special#1{%
1841   \begingroup
1842   \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty

```

```

1843         \else\noexpand##1\noexpand##2\fi}%
1844     \def\do{\x\do}%
1845     \def\@makeother{\x\@makeother}%
1846     \edef\x{\endgroup
1847         \def\noexpand\dospecials{\dospecials}%
1848         \expandafter\ifx\csname @sanitize\endcsname\relax\else
1849             \def\noexpand\@sanitize{\@sanitize}%
1850         \fi}%
1851     \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).

```

1852 \def\bbl@active@def#1#2#3#4{%
1853     \@namedef{#3#1}{%
1854         \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1855             \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1856         \else
1857             \bbl@afterfi\csname#2@sh@#1\endcsname
1858         \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.

```

1859     \long\@namedef{#3@arg#1}##1{%
1860         \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1861             \bbl@afterelse\csname#4#1\endcsname##1%
1862         \else
1863             \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1864         \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.

```

1865 \def\initiate@active@char#1{%
1866     \bbl@ifunset{active@char\string#1}%
1867     {\bbl@withactive
1868         {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1869     {}}

```

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).

```

1870 \def\@initiate@active@char#1#2#3{%
1871   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1872   \ifx#1\@undefined
1873     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1874   \else
1875     \bbl@csarg\let{oridef@#2}#1%
1876     \bbl@csarg\edef{oridef@#2}{%
1877       \let\noexpand#1%
1878       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1879   \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*").

```

1880   \ifx#1#3\relax
1881     \expandafter\let\csname normal@char#2\endcsname#3%
1882   \else
1883     \bbl@info{Making #2 an active character}%
1884     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1885     \@namedef{normal@char#2}{%
1886       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1887   \else
1888     \@namedef{normal@char#2}{#3}%
1889   \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).

```

1890   \bbl@restoreactive{#2}%
1891   \AtBeginDocument{%
1892     \catcode`#2\active
1893     \if@filesw
1894       \immediate\write\@mainaux{\catcode`\string#2\active}%
1895     \fi}%
1896   \expandafter\bbl@add@special\csname#2\endcsname
1897   \catcode`#2\active
1898   \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⟩).

```

1899   \let\bbl@tempa\@firstoftwo
1900   \if\string^#2%
1901     \def\bbl@tempa{\noexpand\textormath}%
1902   \else

```

```

1903 \ifx\bbl@mathnormal\@undefined\else
1904 \let\bbl@tempa\bbl@mathnormal
1905 \fi
1906 \fi
1907 \expandafter\edef\csname active@char#2\endcsname{%
1908 \bbl@tempa
1909 {\noexpand\if@safe@actives
1910 \noexpand\expandafter
1911 \expandafter\noexpand\csname normal@char#2\endcsname
1912 \noexpand\else
1913 \noexpand\expandafter
1914 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1915 \noexpand\fi}%
1916 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1917 \bbl@csarg\edef{doactive#2}{%
1918 \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!).

```

1919 \bbl@csarg\edef{active@#2}{%
1920 \noexpand\active@prefix\noexpand#1%
1921 \expandafter\noexpand\csname active@char#2\endcsname}%
1922 \bbl@csarg\edef{normal@#2}{%
1923 \noexpand\active@prefix\noexpand#1%
1924 \expandafter\noexpand\csname normal@char#2\endcsname}%
1925 \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.

```

1926 \bbl@active@def#2\user@group{user@active}{language@active}%
1927 \bbl@active@def#2\language@group{language@active}{system@active}%
1928 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1929 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1930 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1931 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1932 {\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.

```

1933 \if\string'#2%
1934 \let\prim@s\bbl@prim@s
1935 \let\active@math@prime#1%
1936 \fi
1937 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```
1938 <<*More package options>> ≡
1939 \DeclareOption{math=active}{}
1940 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1941 <</More package options>>
```

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

```
1942 \@ifpackagewith{babel}{KeepShorthandsActive}%
1943   {\let\bbl@restoreactive\@gobble}%
1944   {\def\bbl@restoreactive#1{%
1945     \bbl@exp{%
1946       \\\AfterBabelLanguage\\\CurrentOption
1947       {\catcode`#1=\the\catcode`#1\relax}%
1948       \\\AtEndOfPackage
1949       {\catcode`#1=\the\catcode`#1\relax}}}%
1950   \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.

```
1951 \def\bbl@sh@select#1#2{%
1952   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1953     \bbl@afterelse\bbl@scndcs
1954   \else
1955     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1956   \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.

```
1957 \begingroup
1958 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1959 {\gdef\active@prefix#1{%
1960   \ifx\protect\@typeset@protect
1961   \else
1962     \ifx\protect\@unexpandable@protect
1963       \noexpand#1%
1964     \else
1965       \protect#1%
1966     \fi
1967   \expandafter\@gobble
1968   \fi}}
1969 {\gdef\active@prefix#1{%
1970   \ifincsname
1971     \string#1%
1972     \expandafter\@gobble
1973   \else
1974     \ifx\protect\@typeset@protect
1975     \else
```

```

1976      \ifx\protect\@unexpandable@protect
1977      \noexpand#1%
1978      \else
1979      \protect#1%
1980      \fi
1981      \expandafter\expandafter\expandafter\@gobble
1982      \fi
1983      \fi}}
1984 \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⟩.

1985 \newif\if@safe@actives
1986 \@safe@activesfalse

\bb1@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.

1987 \def\bb1@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

\bb1@activate Both macros take one argument, like \initiate@active@char. The macro is used to
\bb1@deactivate change the definition of an active character to expand to \active@char⟨char⟩ in the case
of \bb1@activate, or \normal@char⟨char⟩ in the case of \bb1@deactivate.

1988 \def\bb1@activate#1{%
1989   \bb1@withactive{\expandafter\let\expandafter}#1%
1990   \csname bbl@active@\string#1\endcsname}
1991 \def\bb1@deactivate#1{%
1992   \bb1@withactive{\expandafter\let\expandafter}#1%
1993   \csname bbl@normal@\string#1\endcsname}

\bb1@firstcs These macros are used only as a trick when declaring shorthands.
\bb1@scndcs
1994 \def\bb1@firstcs#1#2{\csname#1\endcsname}
1995 \def\bb1@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.

1996 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1997 \def\@decl@short#1#2#3\@nil#4{%
1998   \def\bb1@tempa{#3}%
1999   \ifx\bb1@tempa\@empty
2000     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@scndcs
2001     \bb1@ifunset{#1@sh@\string#2@}{}%
2002     {\def\bb1@tempa{#4}%
2003       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bb1@tempa
2004       \else
2005         \bb1@info
2006         {Redefining #1 shorthand \string#2\\%
2007          in language \CurrentOption}%
2008         \fi}%
2009     \@namedef{#1@sh@\string#2@}{#4}%

```



```

2010 \else
2011 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbbl@firstcs
2012 \bbbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2013 {\def\bbbl@tempa{#4}%
2014 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbbl@tempa
2015 \else
2016 \bbbl@info
2017 {Redefining #1 shorthand \string#2\string#3\\%
2018 in language \CurrentOption}%
2019 \fi}%
2020 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2021 \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.

```

2022 \def\textormath{%
2023 \ifmmode
2024 \expandafter\@secondoftwo
2025 \else
2026 \expandafter\@firstoftwo
2027 \fi}

```

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

```

2028 \def\user@group{user}
2029 \def\language@group{english} % TODO. I don't like defaults
2030 \def\system@group{system}

```

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

```

2031 \def\useshorthands{%
2032 \@ifstar\bbbl@usesh@s{\bbbl@usesh@x{}}
2033 \def\bbbl@usesh@s#1{%
2034 \bbbl@usesh@x
2035 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbbl@activate{#1}}}%
2036 {#1}}
2037 \def\bbbl@usesh@x#1#2{%
2038 \bbbl@ifshorthand{#2}%
2039 {\def\user@group{user}%
2040 \initiate@active@char{#2}%
2041 #1%
2042 \bbbl@activate{#2}}%
2043 {\bbbl@error
2044 {Cannot declare a shorthand turned off (\string#2)}
2045 {Sorry, but you cannot use shorthands which have been\\%
2046 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 `\bbbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

2047 \def\user@language@group{user@\language@group}

```

```

2048 \def\bbl@set@user@generic#1#2{%
2049   \bbl@ifunset{user@generic@active#1}%
2050   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2051    \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2052    \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2053     \expandafter\noexpand\csname normal@char#1\endcsname}%
2054    \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2055     \expandafter\noexpand\csname user@active#1\endcsname}}%
2056   \@empty}
2057 \newcommand\defineshorthand[3][user]{%
2058   \edef\bbl@tempa{\zap@space#1 \@empty}%
2059   \bbl@for\bbl@tempb\bbl@tempa{%
2060     \if*\expandafter\car\bbl@tempb\@nil
2061       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2062       \@expandtwoargs
2063       \bbl@set@user@generic{\expandafter\string\car#2\@nil}\bbl@tempb
2064     \fi
2065     \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 [TODO. Unclear].

```

2066 \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`.

```

2067 \def\aliasshorthand#1#2{%
2068   \bbl@ifshorthand{#2}%
2069   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2070     \ifx\document\@notprerr
2071       \@notshorthand{#2}%
2072     \else
2073       \initiate@active@char{#2}%
2074       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2075       \csname active@char\string#1\endcsname
2076       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2077       \csname normal@char\string#1\endcsname
2078       \bbl@activate{#2}%
2079     \fi
2080   \fi}%
2081   {\bbl@error
2082     {Cannot declare a shorthand turned off (\string#2)}
2083     {Sorry, but you cannot use shorthands which have been\\%
2084      turned off in the package options}}}

```

`\@notshorthand`

```

2085 \def\@notshorthand#1{%
2086   \bbl@error{%
2087     The character '\string #1' should be made a shorthand character;\\%
2088     add the command \string\usesshorthands\string{#1\string} to
2089     the preamble.\\%
2090     I will ignore your instruction}%
2091   {You may proceed, but expect unexpected results}}

```

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

```

2092 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2093 \DeclareRobustCommand*\shorthandoff{%
2094   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2095 \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.

```

2096 \def\bbl@switch@sh#1#2{%
2097   \ifx#2\@nnil\else
2098     \bbl@ifunset{\bbl@active@\string#2}%
2099     {\bbl@error
2100      {I cannot switch '\string#2' on or off--not a shorthand}%
2101      {This character is not a shorthand. Maybe you made\\
2102       a typing mistake? I will ignore your instruction}}%
2103     {\ifcase#1%
2104      \catcode`#212\relax
2105      \or
2106      \catcode`#2\active
2107      \or
2108      \csname bbl@oricat@\string#2\endcsname
2109      \csname bbl@oridef@\string#2\endcsname
2110      \fi}%
2111     \bbl@afterfi\bbl@switch@sh#1%
2112   \fi}

```

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

```

2113 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2114 \def\bbl@putsh#1{%
2115   \bbl@ifunset{\bbl@active@\string#1}%
2116   {\bbl@putsh@i#1\@empty\@nnil}%
2117   {\csname bbl@active@\string#1\endcsname}}
2118 \def\bbl@putsh@i#1#2\@nnil{%
2119   \csname\language\@sh@\string#1@%
2120   \ifx\@empty#2\else\string#2@fi\endcsname}
2121 \ifx\bbl@opt@shorthands\@nnil\else
2122   \let\bbl@s@initiate@active@char\initiate@active@char
2123   \def\initiate@active@char#1{%
2124     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2125   \let\bbl@s@switch@sh\bbl@switch@sh
2126   \def\bbl@switch@sh#1#2{%
2127     \ifx#2\@nnil\else
2128       \bbl@afterfi
2129       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2130     \fi}
2131   \let\bbl@s@activate\bbl@activate
2132   \def\bbl@activate#1{%
2133     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2134   \let\bbl@s@deactivate\bbl@deactivate
2135   \def\bbl@deactivate#1{%

```

```

2136 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2137 \fi

```

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

```

2138 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@string#1}{#3}{#2}}

```

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

`\bbl@pr@m@s`

```

2139 \def\bbl@prim@s{%
2140 \prime\futurelet\@let@token\bbl@pr@m@s}
2141 \def\bbl@if@primes#1#2{%
2142 \ifx#1\@let@token
2143 \expandafter\@firstoftwo
2144 \else\ifx#2\@let@token
2145 \bbl@afterelse\expandafter\@firstoftwo
2146 \else
2147 \bbl@afterfi\expandafter\@secondoftwo
2148 \fi\fi}
2149 \begingroup
2150 \catcode`\^=7 \catcode`\*= \active \lccode`\*=`\^
2151 \catcode`\'=12 \catcode`\`= \active \lccode`\`=``
2152 \lowercase{%
2153 \gdef\bbl@pr@m@s{%
2154 \bbl@if@primes" "%
2155 \pr@@@s
2156 {\bbl@if@primes*^ \pr@@@t\egroup}}}
2157 \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).

```

2158 \initiate@active@char{~}
2159 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2160 \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.

`\T1dqpos`

```

2161 \expandafter\def\csname OT1dqpos\endcsname{127}
2162 \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

```

2163 \ifx\f@encoding\@undefined
2164 \def\f@encoding{OT1}
2165 \fi

```

9.6 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

`\languageattribute` The macro `\languageattribute` checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```

2166 \bbl@trace{Language attributes}
2167 \newcommand\languageattribute[2]{%
2168   \def\bbl@tempc{#1}%
2169   \bbl@fixname\bbl@tempc
2170   \bbl@iflanguage\bbl@tempc{%
2171     \bbl@vforeach{#2}{%

```

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

```

2172     \ifx\bbl@known@attribs\undefined
2173       \in@false
2174     \else
2175       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2176     \fi
2177     \ifin@
2178       \bbl@warning{%
2179         You have more than once selected the attribute '##1'\%
2180         for language #1. Reported}%
2181     \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.

```

2182       \bbl@exp{%
2183         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2184       \edef\bbl@tempa{\bbl@tempc-##1}%
2185       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2186       {\csname\bbl@tempc @attr##1\endcsname}%
2187       {\@attrerr{\bbl@tempc}{##1}}%
2188     \fi}}
2189 \@onlypreamble\languageattribute

```

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

```

2190 \newcommand*{\@attrerr}[2]{%
2191   \bbl@error
2192   {The attribute #2 is unknown for language #1.}%
2193   {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}`.

```

2194 \def\bbl@declare@ttribute#1#2#3{%
2195   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2196   \ifin@
2197     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2198   \fi
2199   \bbl@add@list\bbl@attributes{#1-#2}%
2200   \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret \TeX code based on whether a certain attribute was set. This command should appear inside the argument to

\AtBeginDocument because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far \ifin@ has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the \fi'.

```
2201 \def\bbbl@ifattributeset#1#2#3#4{%
2202   \ifx\bbbl@known@attribs\@undefined
2203     \in@false
2204   \else
2205     \bbbl@xin@{,#1-#2,}{,\bbbl@known@attribs,}%
2206   \fi
2207   \ifin@
2208     \bbbl@afterelse#3%
2209   \else
2210     \bbbl@afterfi#4%
2211   \fi
2212 }
```

`\bbbl@ifknown@ttrib` An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise.

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of \bbbl@tempa is changed. Finally we execute \bbbl@tempa.

```
2213 \def\bbbl@ifknown@ttrib#1#2{%
2214   \let\bbbl@tempa\@secondoftwo
2215   \bbbl@loopx\bbbl@tempb{#2}{%
2216     \expandafter\in@\expandafter{\expandafter,\bbbl@tempb,}{,#1,}%
2217   \ifin@
2218     \let\bbbl@tempa\@firstoftwo
2219   \else
2220     \fi}%
2221   \bbbl@tempa
2222 }
```

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

```
2223 \def\bbbl@clear@attribs{%
2224   \ifx\bbbl@attributes\@undefined\else
2225     \bbbl@loopx\bbbl@tempa{\bbbl@attributes}{%
2226       \expandafter\bbbl@clear@ttrib\bbbl@tempa.
2227     }%
2228     \let\bbbl@attributes\@undefined
2229   \fi}
2230 \def\bbbl@clear@ttrib#1-#2.{%
2231   \expandafter\let\csname#1@attr#2\endcsname\@undefined}
2232 \AtBeginDocument{\bbbl@clear@attribs}
```

9.7 Support for saving macro definitions

To save the meaning of control sequences using \babel@save, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply

use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`’ed.

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.

`\babel@beginsave` 2233 `\bbl@trace{Macros for saving definitions}`
2234 `\def\babel@beginsave{\babel@savecnt\z@}`

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

2235 `\newcount\babel@savecnt`
2236 `\babel@beginsave`

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence
`\babel@savevariable` `⟨csname⟩` to `\originalTeX`³¹. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive.

2237 `\def\babel@save#1{%`
2238 `\expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax`
2239 `\toks@\expandafter{\originalTeX\let#1=}%`
2240 `\bbl@exp{%`
2241 `\def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%`
2242 `\advance\babel@savecnt\@ne}`
2243 `\def\babel@savevariable#1{%`
2244 `\toks@\expandafter{\originalTeX #1=}%`
2245 `\bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}`

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

2246 `\def\bbl@frenchspacing{%`
2247 `\ifnum\the\sfcode\.\=@m`
2248 `\let\bbl@nonfrenchspacing\relax`
2249 `\else`
2250 `\frenchspacing`
2251 `\let\bbl@nonfrenchspacing\nonfrenchspacing`
2252 `\fi}`
2253 `\let\bbl@nonfrenchspacing\nonfrenchspacing`

9.8 Short tags

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

2254 `\bbl@trace{Short tags}`
2255 `\def\babeltags#1{%`
2256 `\edef\bbl@tempa{\zap@space#1 \@empty}%`
2257 `\def\bbl@tempb##1=##2\@{%`
2258 `\edef\bbl@tempc{%`
2259 `\noexpand\newcommand`
2260 `\expandafter\noexpand\csname ##1\endcsname{%`
2261 `\noexpand\protect`
2262 `\expandafter\noexpand\csname otherlanguage*\endcsname{##2}}}`

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

```

2263 \noexpand\newcommand
2264 \expandafter\noexpand\csname text##1\endcsname{%
2265 \noexpand\foreignlanguage{##2}}}%
2266 \bbl@tempc}%
2267 \bbl@for\bbl@tempa\bbl@tempa{%
2268 \expandafter\bbl@tempb\bbl@tempa\@@}%

```

9.9 Hyphens

`\babelhyphenation` This macro saves hyphenation exceptions. Two macros are used to store them: `\bbl@hyphenation@` for the global ones and `\bbl@hyphenation<lang>` for language ones. See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```

2269 \bbl@trace{Hyphens}
2270 \@onlypreamble\babelhyphenation
2271 \AtEndOfPackage{%
2272 \newcommand\babelhyphenation[2][\@empty]{%
2273 \ifx\bbl@hyphenation@ \relax
2274 \let\bbl@hyphenation@ \@empty
2275 \fi
2276 \ifx\bbl@hyphlist \@empty \else
2277 \bbl@warning{%
2278 You must not intermingle \string\selectlanguage\space and\%
2279 \string\babelhyphenation\space or some exceptions will not\%
2280 be taken into account. Reported}%
2281 \fi
2282 \ifx \@empty #1%
2283 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2284 \else
2285 \bbl@vforeach{#1}{%
2286 \def\bbl@tempa{##1}%
2287 \bbl@fixname\bbl@tempa
2288 \bbl@iflanguage\bbl@tempa{%
2289 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2290 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2291 \@empty
2292 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2293 #2}}}%
2294 \fi}}

```

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

```

2295 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
2296 \def\bbl@t@one{T1}
2297 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

`\babelhyphen` Macros to insert common hyphens. Note the space before `@` in `\babelhyphen`. Instead of protecting it with `\DeclareRobustCommand`, which could insert a `\relax`, we use the same procedure as shorthands, with `\active@prefix`.

```

2298 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2299 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2300 \def\bbl@hyphen{%
2301 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}%
2302 \def\bbl@hyphen@i#1#2{%
2303 \bbl@ifunset{bbl@hy@#1#2\@empty}%

```

³² $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.


```

2304 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{#2}}}%
2305 {\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.

```

2306 \def\bbl@usehyphen#1{%
2307   \leavevmode
2308   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2309   \nobreak\hskip\z@skip}
2310 \def\bbl@usehyphen#1{%
2311   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2312 \def\bbl@hyphenchar{%
2313   \ifnum\hyphenchar\font=\m@ne
2314     \babe\nullhyphen
2315   \else
2316     \char\hyphenchar\font
2317   \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.

```

2318 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2319 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2320 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2321 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2322 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2323 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2324 \def\bbl@hy@repeat{%
2325   \bbl@usehyphen%
2326   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2327 \def\bbl@hy@repeat{%
2328   \bbl@usehyphen%
2329   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2330 \def\bbl@hy@empty{\hskip\z@skip}
2331 \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.

```

2332 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

9.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

Tools But first, a couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```

2333 \bbl@trace{Multiencoding strings}
2334 \def\bbl@tglobal#1{\global\let#1#1}

```

```

2335 \def\bbl@recatcode#1{% TODO. Used only once?
2336   \@tempcnta="7F
2337   \def\bbl@tempa{%
2338     \ifnum\@tempcnta>"FF\else
2339       \catcode\@tempcnta=#1\relax
2340       \advance\@tempcnta\@ne
2341       \expandafter\bbl@tempa
2342     \fi}%
2343   \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).

```

2344 \@ifpackagewith{babel}{nocase}%
2345   {\let\bbl@patchuclc\relax}%
2346   {\def\bbl@patchuclc{%
2347     \global\let\bbl@patchuclc\relax
2348     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2349     \gdef\bbl@uclc##1{%
2350       \let\bbl@encoded\bbl@encoded@uclc
2351       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2352       {##1}%
2353       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2354         \csname\language @bbl@uclc\endcsname}%
2355       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2356     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2357     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2358 <<More package options>> ≡
2359 \DeclareOption{nocase}{}
2360 <</More package options>>

```

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

```

2361 <<More package options>> ≡
2362 \let\bbl@opt@strings\@nnil % accept strings=value
2363 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2364 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2365 \def\BabelStringsDefault{generic}
2366 <</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.

```

2367 \@onlypreamble\StartBabelCommands
2368 \def\StartBabelCommands{%
2369   \begingroup
2370   \bbl@recatcode{11}%
2371   <<Macros local to BabelCommands>>
2372   \def\bbl@provstring##1##2{%

```

```

2373 \providecommand##1{##2}%
2374 \bbl@tglobal##1}%
2375 \global\let\bbl@scafter\@empty
2376 \let\StartBabelCommands\bbl@startcmds
2377 \ifx\BabelLanguages\relax
2378 \let\BabelLanguages\CurrentOption
2379 \fi
2380 \begingroup
2381 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2382 \StartBabelCommands}
2383 \def\bbl@startcmds{%
2384 \ifx\bbl@screset\@nnil\else
2385 \bbl@usehooks{stopcommands}{}}%
2386 \fi
2387 \endgroup
2388 \begingroup
2389 \@ifstar
2390 {\ifx\bbl@opt@strings\@nnil
2391 \let\bbl@opt@strings\BabelStringsDefault
2392 \fi
2393 \bbl@startcmds@i}%
2394 \bbl@startcmds@i}
2395 \def\bbl@startcmds@i#1#2{%
2396 \edef\bbl@L{\zap@space#1 \@empty}%
2397 \edef\bbl@G{\zap@space#2 \@empty}%
2398 \bbl@startcmds@ii}
2399 \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.

```

2400 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2401 \let\SetString\@gobbletwo
2402 \let\bbl@stringdef\@gobbletwo
2403 \let\AfterBabelCommands\@gobble
2404 \ifx\@empty#1%
2405 \def\bbl@sc@label{generic}%
2406 \def\bbl@encstring##1##2{%
2407 \ProvideTextCommandDefault##1{##2}%
2408 \bbl@tglobal##1%
2409 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2410 \let\bbl@sctest\in@true
2411 \else
2412 \let\bbl@sc@charset\space % <- zapped below
2413 \let\bbl@sc@fontenc\space % <- " "
2414 \def\bbl@tempa##1=##2\@nil{%
2415 \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%
2416 \bbl@foreach{label=#1}{\bbl@tempa##1\@nil}%
2417 \def\bbl@tempa##1 ##2{% space -> comma
2418 ##1%
2419 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%

```

```

2420 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2421 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2422 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2423 \def\bbl@encstring##1##2{%
2424   \bbl@foreach\bbl@sc@fontenc{%
2425     \bbl@ifunset{T@###1}%
2426     {}%
2427     {\ProvideTextCommand##1{###1}{##2}%
2428       \bbl@tglobal##1%
2429       \expandafter
2430       \bbl@tglobal\csname###1\string##1\endcsname}}}%
2431 \def\bbl@sc@test{%
2432   \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
2433 \fi
2434 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2435 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2436   \let\AfterBabelCommands\bbl@aftercmds
2437   \let\SetString\bbl@setstring
2438   \let\bbl@stringdef\bbl@encstring
2439 \else % ie, strings=value
2440 \bbl@sc@test
2441 \ifin@
2442   \let\AfterBabelCommands\bbl@aftercmds
2443   \let\SetString\bbl@setstring
2444   \let\bbl@stringdef\bbl@provstring
2445 \fi\fi\fi
2446 \bbl@scswitch
2447 \ifx\bbl@G\@empty
2448   \def\SetString##1##2{%
2449     \bbl@error{Missing group for string \string##1}%
2450     {You must assign strings to some category, typically\\%
2451       captions or extras, but you set none}}%
2452 \fi
2453 \ifx\@empty#1%
2454   \bbl@usehooks{defaultcommands}{}%
2455 \else
2456   \@expandtwoargs
2457   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2458 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing.

The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has been loaded) .

```

2459 \def\bbl@forlang#1#2{%
2460   \bbl@for#1\bbl@L{%
2461     \bbl@xin@{,#1,}{\BabelLanguages,}%
2462     \ifin@#2\relax\fi}}
2463 \def\bbl@scswitch{%
2464   \bbl@forlang\bbl@tempa{%
2465     \ifx\bbl@G\@empty\else
2466       \ifx\SetString\@gobbletwo\else
2467         \edef\bbl@GL{\bbl@G\bbl@tempa}%

```

```

2468      \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
2469      \ifin@ \else
2470      \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2471      \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2472      \fi
2473    \fi
2474  \fi}}
2475 \AtEndOfPackage{%
2476   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2477   \let\bbl@scswitch\relax}
2478 \@onlypreamble\EndBabelCommands
2479 \def\EndBabelCommands{%
2480   \bbl@usehooks{stopcommands}{}%
2481   \endgroup
2482   \endgroup
2483   \bbl@scafter}
2484 \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.

```

2485 \def\bbl@setstring#1#2{%
2486   \bbl@forlang\bbl@tempa{%
2487     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2488     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2489     {\global\expandafter % TODO - con \bbl@exp ?
2490      \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2491       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
2492     {}}%
2493   \def\BabelString{#2}%
2494   \bbl@usehooks{stringprocess}{}%
2495   \expandafter\bbl@stringdef
2496   \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.

```

2497 \ifx\bbl@opt@strings\relax
2498   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2499   \bbl@patchuclc
2500   \let\bbl@encoded\relax
2501   \def\bbl@encoded@uclc#1{%
2502     \@inmathwarn#1%
2503     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2504       \expandafter\ifx\csname ?\string#1\endcsname\relax
2505         \TextSymbolUnavailable#1%
2506       \else
2507         \csname ?\string#1\endcsname
2508       \fi
2509     \else
2510       \csname\cf@encoding\string#1\endcsname
2511     \fi}
2512 \else

```

```

2513 \def\bbl@scset#1#2{\def#1{#2}}
2514 \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.

```

2515 <<(*Macros local to BabelCommands)>> ≡
2516 \def\SetStringLoop##1##2{%
2517   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2518   \count@\z@
2519   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2520     \advance\count@\@ne
2521     \toks@\expandafter{\bbl@tempa}%
2522     \bbl@exp{%
2523       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2524       \count@=\the\count@\relax}}}%
2525 <</Macros local to BabelCommands>>

```

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

```

2526 \def\bbl@aftercmds#1{%
2527   \toks@\expandafter{\bbl@scafter#1}%
2528   \xdef\bbl@scafter{\the\toks@}}

```

Case mapping The command `\SetCase` provides a way to change the behavior of `\MakeUppercase` and `\MakeLowercase`. `\bbl@tempa` is set by the patched `\@uclclist` to the parsing command.

```

2529 <<(*Macros local to BabelCommands)>> ≡
2530 \newcommand\SetCase[3][]{%
2531   \bbl@patchuclc
2532   \bbl@forlang\bbl@tempa{%
2533     \expandafter\bbl@encstring
2534     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2535     \expandafter\bbl@encstring
2536     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2537     \expandafter\bbl@encstring
2538     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2539 <</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.

```

2540 <<(*Macros local to BabelCommands)>> ≡
2541 \newcommand\SetHyphenMap[1]{%
2542   \bbl@forlang\bbl@tempa{%
2543     \expandafter\bbl@stringdef
2544     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2545 <</Macros local to BabelCommands>>

```

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

```

2546 \newcommand\BabelLower[2]{% one to one.
2547   \ifnum\lccode#1=#2\else
2548     \babel@savevariable{\lccode#1}%
2549     \lccode#1=#2\relax
2550   \fi}
2551 \newcommand\BabelLowerMM[4]{% many-to-many
2552   \@tempcnta=#1\relax

```

```

2553 \@tempcntb=#4\relax
2554 \def\bbl@tempa{%
2555   \ifnum\@tempcnta>#2\else
2556     \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2557     \advance\@tempcnta#3\relax
2558     \advance\@tempcntb#3\relax
2559     \expandafter\bbl@tempa
2560   \fi}%
2561 \bbl@tempa}
2562 \newcommand\BabelLowerM0[4]{% many-to-one
2563 \@tempcnta=#1\relax
2564 \def\bbl@tempa{%
2565   \ifnum\@tempcnta>#2\else
2566     \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2567     \advance\@tempcnta#3
2568     \expandafter\bbl@tempa
2569   \fi}%
2570 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2571 <<{*More package options}>> ≡
2572 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2573 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2574 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2575 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2576 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2577 <</More package options>>

```

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

```

2578 \AtEndOfPackage{%
2579   \ifx\bbl@opt@hyphenmap\undefined
2580     \bbl@xin{,}{\bbl@language@opts}%
2581     \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
2582   \fi}

```

9.11 Macros common to a number of languages

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

```

2583 \bbl@trace{Macros related to glyphs}
2584 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2585   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2586   \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.

```

2587 \def\save@sf@q#1{\leavevmode
2588   \begingroup
2589     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2590   \endgroup}

```

9.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through T1enc.def.

9.12.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```
2591 \ProvideTextCommand{\quotedblbase}{OT1}{%
2592   \save@sf@q{\set@low@box{\textquotedblright\}}%
2593   \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.

```
2594 \ProvideTextCommandDefault{\quotedblbase}{%
2595   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2596 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2597   \save@sf@q{\set@low@box{\textquoteright\}}%
2598   \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.

```
2599 \ProvideTextCommandDefault{\quotesinglbase}{%
2600   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2601 \ProvideTextCommand{\guillemetleft}{OT1}{%
2602   \ifmmode
2603     \ll
2604   \else
2605     \save@sf@q{\nobreak
2606       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2607     \fi}
2608 \ProvideTextCommand{\guillemetright}{OT1}{%
2609   \ifmmode
2610     \gg
2611   \else
2612     \save@sf@q{\nobreak
2613       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2614     \fi}
2615 \ProvideTextCommand{\guillemotleft}{OT1}{%
2616   \ifmmode
2617     \ll
2618   \else
2619     \save@sf@q{\nobreak
2620       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2621     \fi}
2622 \ProvideTextCommand{\guillemotright}{OT1}{%
2623   \ifmmode
2624     \gg
2625   \else
2626     \save@sf@q{\nobreak
2627       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2628     \fi}
```

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

```
2629 \ProvideTextCommandDefault{\guillemetleft}{%
```



```

2630 \UseTextSymbol{OT1}{\guillemetleft}}
2631 \ProvideTextCommandDefault{\guillemetright}{%
2632 \UseTextSymbol{OT1}{\guillemetright}}
2633 \ProvideTextCommandDefault{\guillemotleft}{%
2634 \UseTextSymbol{OT1}{\guillemotleft}}
2635 \ProvideTextCommandDefault{\guillemotright}{%
2636 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2637 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2638 \ifmmode
2639 <%
2640 \else
2641 \save@sf@q{\nobreak
2642 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2643 \fi}
2644 \ProvideTextCommand{\guilsinglright}{OT1}{%
2645 \ifmmode
2646 >%
2647 \else
2648 \save@sf@q{\nobreak
2649 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2650 \fi}

```

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

```

2651 \ProvideTextCommandDefault{\guilsinglleft}{%
2652 \UseTextSymbol{OT1}{\guilsinglleft}}
2653 \ProvideTextCommandDefault{\guilsinglright}{%
2654 \UseTextSymbol{OT1}{\guilsinglright}}

```

9.12.2 Letters

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

```

2655 \DeclareTextCommand{\ij}{OT1}{%
2656 i\kern-0.02em\bbl@allowhyphens j}
2657 \DeclareTextCommand{\IJ}{OT1}{%
2658 I\kern-0.02em\bbl@allowhyphens J}
2659 \DeclareTextCommand{\ij}{T1}{\char188}
2660 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2661 \ProvideTextCommandDefault{\ij}{%
2662 \UseTextSymbol{OT1}{\ij}}
2663 \ProvideTextCommandDefault{\IJ}{%
2664 \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

2665 \def\crrtic@{\hrule height0.1ex width0.3em}
2666 \def\crttic@{\hrule height0.1ex width0.33em}
2667 \def\ddj@{%
2668 \setbox0\hbox{d}\dimen@=\ht0

```

```

2669 \advance\dimen@1ex
2670 \dimen@.45\dimen@
2671 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2672 \advance\dimen@ii.5ex
2673 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2674 \def\DDJ@{%
2675 \setbox0\hbox{D}\dimen@=.55\ht0
2676 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2677 \advance\dimen@ii.15ex % correction for the dash position
2678 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2679 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2680 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2681 %
2682 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2683 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2684 \ProvideTextCommandDefault{\dj}{%
2685 \UseTextSymbol{OT1}{\dj}}
2686 \ProvideTextCommandDefault{\DJ}{%
2687 \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.

```

2688 \DeclareTextCommand{\SS}{OT1}{\SS}
2689 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

9.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with `\ProvideTextCommandDefault`, but this is very likely not required because their definitions are based on encoding-dependent macros.

`\glq` The ‘german’ single quotes.

```

\grq 2690 \ProvideTextCommandDefault{\glq}{%
2691 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2692 \ProvideTextCommand{\grq}{T1}{%
2693 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2694 \ProvideTextCommand{\grq}{TU}{%
2695 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2696 \ProvideTextCommand{\grq}{OT1}{%
2697 \save@sf@q{\kern-.0125em
2698 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2699 \kern.07em\relax}}
2700 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq 2701 \ProvideTextCommandDefault{\glqq}{%
2702 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2703 \ProvideTextCommand{\grqq}{T1}{%
2704   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2705 \ProvideTextCommand{\grqq}{TU}{%
2706   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2707 \ProvideTextCommand{\grqq}{OT1}{%
2708   \save@sf@q{\kern-.07em
2709     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2710     \kern.07em\relax}}
2711 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.

```

\frq 2712 \ProvideTextCommandDefault{\flq}{%
2713   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2714 \ProvideTextCommandDefault{\frq}{%
2715   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```

\frqq 2716 \ProvideTextCommandDefault{\flqq}{%
2717   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2718 \ProvideTextCommandDefault{\frqq}{%
2719   \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

9.12.4 Umlauts and tremas

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

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

```

2720 \def\umlauthigh{%
2721   \def\bb1@umlauta##1{\leavevmode\bggroup%
2722     \expandafter\accent\csname\fontencoding dqpos\endcsname
2723     ##1\bb1@allowhyphens\egroup}%
2724   \let\bb1@umlaute\bb1@umlauta}
2725 \def\umlautlow{%
2726   \def\bb1@umlauta{\protect\lower@umlaut}}
2727 \def\umlautelow{%
2728   \def\bb1@umlaute{\protect\lower@umlaut}}
2729 \umlauthigh

```

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

```

2730 \expandafter\ifx\csname U@D\endcsname\relax
2731   \csname newdimen\endcsname\U@D
2732 \fi

```

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

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low,

it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

2733 \def\lower@umlaut#1{%
2734   \leavevmode\bgroup
2735     \U@D 1ex%
2736     {\setbox\z@\hbox{%
2737       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2738       \dimen@ -.45ex\advance\dimen@\ht\z@
2739       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2740     \expandafter\accent\csname\fontencoding dqpos\endcsname
2741     \fontdimen5\font\U@D #1%
2742   \egroup}

```

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

```

2743 \AtBeginDocument{%
2744   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2745   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2746   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2747   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2748   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2749   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2750   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2751   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2752   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2753   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2754   \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.

```

2755 \ifx\l@english\@undefined
2756   \chardef\l@english\z@
2757 \fi
2758 % The following is used to cancel rules in ini files (see Amharic).
2759 \ifx\l@babelnohyphens\@undefined
2760   \newlanguage\l@babelnohyphens
2761 \fi

```

9.13 Layout

Work in progress.

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

```

2762 \bbl@trace{Bidi layout}
2763 \providecommand\IfBabelLayout[3]{#3}%
2764 \newcommand\BabelPatchSection[1]{%
2765   \ifundefined{#1}{-%
2766     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2767     \@namedef{#1}{%
2768       \ifstar{\bbl@presec@s{#1}}%
2769       {\@dblarg{\bbl@presec@x{#1}}}}}%

```

```

2770 \def\bbl@presec@x#1[#2]#3{%
2771   \bbl@exp{%
2772     \\\select@language@x{\bbl@main@language}%
2773     \\\bbl@cs{sspre@#1}%
2774     \\\bbl@cs{ss@#1}%
2775     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2776     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2777     \\\select@language@x{\language}}}%
2778 \def\bbl@presec@s#1#2{%
2779   \bbl@exp{%
2780     \\\select@language@x{\bbl@main@language}%
2781     \\\bbl@cs{sspre@#1}%
2782     \\\bbl@cs{ss@#1}*%
2783     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2784     \\\select@language@x{\language}}}%
2785 \IfBabelLayout{sectioning}%
2786   {\BabelPatchSection{part}%
2787    \BabelPatchSection{chapter}%
2788    \BabelPatchSection{section}%
2789    \BabelPatchSection{subsection}%
2790    \BabelPatchSection{subsubsection}%
2791    \BabelPatchSection{paragraph}%
2792    \BabelPatchSection{subparagraph}%
2793    \def\babel@toc#1{%
2794      \select@language@x{\bbl@main@language}}}%
2795 \IfBabelLayout{captions}%
2796   {\BabelPatchSection{caption}}}%

```

9.14 Load engine specific macros

```

2797 \bbl@trace{Input engine specific macros}
2798 \ifcase\bbl@engine
2799   \input txtbabel.def
2800 \or
2801   \input luababel.def
2802 \or
2803   \input xebabel.def
2804 \fi

```

9.15 Creating and modifying languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```

2805 \bbl@trace{Creating languages and reading ini files}
2806 \newcommand\babelprovide[2][{}]{%
2807   \let\bbl@savelangname\language
2808   \edef\bbl@savelocaleid{\the\localeid}%
2809   % Set name and locale id
2810   \edef\language{#2}%
2811   % \global\@namedef{\bbl@lcname@#2}{#2}%
2812   \bbl@id@assign
2813   \let\bbl@KVP@captions\@nil
2814   \let\bbl@KVP@date\@nil
2815   \let\bbl@KVP@import\@nil
2816   \let\bbl@KVP@main\@nil
2817   \let\bbl@KVP@script\@nil
2818   \let\bbl@KVP@language\@nil

```

```

2819 \let\bbl@KVP@hyphenrules\@nil % only for provide@new
2820 \let\bbl@KVP@mapfont\@nil
2821 \let\bbl@KVP@maparabic\@nil
2822 \let\bbl@KVP@mapdigits\@nil
2823 \let\bbl@KVP@intraspace\@nil
2824 \let\bbl@KVP@intrapenalty\@nil
2825 \let\bbl@KVP@onchar\@nil
2826 \let\bbl@KVP@alph\@nil
2827 \let\bbl@KVP@Alph\@nil
2828 \let\bbl@KVP@labels\@nil
2829 \bbl@csarg\let{KVP@labels*}\@nil
2830 \bbl@forkv{#1}{% TODO - error handling
2831   \in@{/}{##1}%
2832   \ifin@
2833     \bbl@renewinikey##1\@{##2}%
2834   \else
2835     \bbl@csarg\def{KVP@##1}{##2}%
2836   \fi}%
2837 % == import, captions ==
2838 \ifx\bbl@KVP@import\@nil\else
2839   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2840   {\ifx\bbl@initoload\relax
2841     \begingroup
2842       \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2843       \bbl@input@texini{#2}%
2844     \endgroup
2845   \else
2846     \xdef\bbl@KVP@import{\bbl@initoload}%
2847   \fi}%
2848 {}%
2849 \fi
2850 \ifx\bbl@KVP@captions\@nil
2851   \let\bbl@KVP@captions\bbl@KVP@import
2852 \fi
2853 % Load ini
2854 \bbl@ifunset{date#2}%
2855   {\bbl@provide@new{#2}}%
2856   {\bbl@ifblank{#1}%
2857     {\bbl@error
2858       {If you want to modify `#2' you must tell how in\\
2859         the optional argument. See the manual for the\\
2860         available options.}%
2861       {Use this macro as documented}}%
2862     {\bbl@provide@renew{#2}}}%
2863 % Post tasks
2864 \bbl@ifunset{bbl@extracaps@#2}%
2865   {\bbl@exp{\bbl@babelensure[exclude=\today]{#2}}}%
2866   {\toks\@expandafter\expandafter\expandafter
2867     {\csname bbl@extracaps@#2\endcsname}%
2868     \bbl@exp{\bbl@babelensure[exclude=\today,include=\the\toks@]{#2}}%
2869   \bbl@ifunset{bbl@ensure@\language}%
2870     {\bbl@exp{%
2871       \bbl@DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2872         \bbl@foreignlanguage{\language}%
2873         {###1}}}%
2874     }%
2875   \bbl@exp{%
2876     \bbl@toglobal\<bbl@ensure@\language>%
2877     \bbl@toglobal\<bbl@ensure@\language\space>%

```

```

2878 % At this point all parameters are defined if 'import'. Now we
2879 % execute some code depending on them. But what about if nothing was
2880 % imported? We just load the very basic parameters.
2881 \bbl@load@basic{#2}%
2882 % == script, language ==
2883 % Override the values from ini or defines them
2884 \ifx\bbl@KVP@script\@nil\else
2885   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2886 \fi
2887 \ifx\bbl@KVP@language\@nil\else
2888   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2889 \fi
2890 % == onchar ==
2891 \ifx\bbl@KVP@onchar\@nil\else
2892   \bbl@luahyphenate
2893   \directlua{
2894     if Babel.locale_mapped == nil then
2895       Babel.locale_mapped = true
2896       Babel.linebreaking.add_before(Babel.locale_map)
2897       Babel.loc_to_scr = {}
2898       Babel.chr_to_loc = Babel.chr_to_loc or {}
2899     end}%
2900   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2901   \ifin@
2902     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2903       \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2904     \fi
2905     \bbl@exp{\\ \bbl@add\\ \bbl@starthyphens
2906       {\ \bbl@patterns@lua{\language\name}}}%
2907     % TODO - error/warning if no script
2908     \directlua{
2909       if Babel.script_blocks['\bbl@cl{sbc}'] then
2910         Babel.loc_to_scr[\the\localeid] =
2911           Babel.script_blocks['\bbl@cl{sbc}']
2912         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2913         Babel.locale_props[\the\localeid].lg = \the\@nameuse{1@\language\name}\space
2914       end
2915     }%
2916   \fi
2917   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2918   \ifin@
2919     \bbl@ifunset{bbl@lsys\language\name}{\bbl@provide@lsys\language\name}{}%
2920     \bbl@ifunset{bbl@wdir\language\name}{\bbl@provide@dirs\language\name}{}%
2921     \directlua{
2922       if Babel.script_blocks['\bbl@cl{sbc}'] then
2923         Babel.loc_to_scr[\the\localeid] =
2924           Babel.script_blocks['\bbl@cl{sbc}']
2925       end}%
2926     \ifx\bbl@mapselect\@undefined
2927       \AtBeginDocument{%
2928         \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
2929         {\selectfont}}%
2930       \def\bbl@mapselect{%
2931         \let\bbl@mapselect\relax
2932         \edef\bbl@prefontid{\fontid\font}}%
2933       \def\bbl@mapdir##1{%
2934         {\def\language{##1}%
2935          \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2936          \bbl@switchfont

```

```

2937         \directlua{
2938             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2939                 ['\bbl@prefontid'] = \fontid\font\space}}}%
2940     \fi
2941     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2942 \fi
2943 % TODO - catch non-valid values
2944 \fi
2945 % == mapfont ==
2946 % For bidi texts, to switch the font based on direction
2947 \ifx\bbl@KVP@mapfont\@nil\else
2948     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
2949     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2950         mapfont. Use 'direction'.%
2951         {See the manual for details.}}}%
2952     \bbl@ifunset{\bbl@sys@\language}{\bbl@provide@sys{\language}}}%
2953     \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
2954     \ifx\bbl@mapselect\@undefined
2955         \AtBeginDocument{%
2956             \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2957             {\selectfont}}%
2958         \def\bbl@mapselect{%
2959             \let\bbl@mapselect\relax
2960             \edef\bbl@prefontid{\fontid\font}}%
2961         \def\bbl@mapdir##1{%
2962             {\def\language{##1}%
2963             \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2964             \bbl@switchfont
2965             \directlua{Babel.fontmap
2966                 [\the\csname bbl@wdir@##1\endcsname]%
2967                 [\bbl@prefontid]=\fontid\font}}}%
2968         \fi
2969         \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2970 \fi
2971 % == intraspace, intrapenalty ==
2972 % For CJK, East Asian, Southeast Asian, if interspace in ini
2973 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2974     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2975 \fi
2976 \bbl@provide@intraspace
2977 % == hyphenate.other.locale ==
2978 \bbl@ifunset{\bbl@hyotl@\language}{}%
2979     {\bbl@csarg\bbl@replace{\hyotl@\language}{ }{ },}%
2980     \bbl@startcommands*{\language}{}%
2981     \bbl@csarg\bbl@foreach{\hyotl@\language}{%
2982         \ifcase\bbl@engine
2983             \ifnum##1<257
2984                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2985             \fi
2986             \else
2987                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2988             \fi}%
2989     \bbl@endcommands}%
2990 % == hyphenate.other.script ==
2991 \bbl@ifunset{\bbl@hyots@\language}{}%
2992     {\bbl@csarg\bbl@replace{\hyots@\language}{ }{ },}%
2993     \bbl@csarg\bbl@foreach{\hyots@\language}{%
2994         \ifcase\bbl@engine
2995             \ifnum##1<257

```



```

2996         \global\lccode##1=##1\relax
2997     \fi
2998 \else
2999     \global\lccode##1=##1\relax
3000 \fi}}%
3001 % == maparabic ==
3002 % Native digits, if provided in ini (TeX level, xe and lua)
3003 \ifcase\bbbl@engine\else
3004     \bbbl@ifunset{\bbbl@dgnat@\language\name}{}%
3005     {\expandafter\ifx\csname \bbbl@dgnat@\language\name\endcsname\@empty\else
3006         \expandafter\expandafter\expandafter
3007         \bbbl@setdigits\csname \bbbl@dgnat@\language\name\endcsname
3008         \ifx\bbbl@KVP@maparabic\@nil\else
3009             \ifx\bbbl@latinarabic\@undefined
3010                 \expandafter\let\expandafter\@arabic
3011                 \csname \bbbl@counter@\language\name\endcsname
3012             \else % ie, if layout=counters, which redefines \@arabic
3013                 \expandafter\let\expandafter\bbbl@latinarabic
3014                 \csname \bbbl@counter@\language\name\endcsname
3015             \fi
3016         \fi
3017     \fi}%
3018 \fi
3019 % == mapdigits ==
3020 % Native digits (lua level).
3021 \ifodd\bbbl@engine
3022     \ifx\bbbl@KVP@mapdigits\@nil\else
3023         \bbbl@ifunset{\bbbl@dgnat@\language\name}{}%
3024         {\RequirePackage{luatexbase}%
3025         \bbbl@activate@preotf
3026         \directlua{
3027             Babel = Babel or {} %%% -> presets in luababel
3028             Babel.digits_mapped = true
3029             Babel.digits = Babel.digits or {}
3030             Babel.digits[\the\localeid] =
3031                 table.pack(string.utfvalue('\bbbl@cldgnat'))
3032             if not Babel.numbers then
3033                 function Babel.numbers(head)
3034                     local LOCALE = luatexbase.registernumber'\bbbl@attr@locale'
3035                     local GLYPH = node.id'glyph'
3036                     local inmath = false
3037                     for item in node.traverse(head) do
3038                         if not inmath and item.id == GLYPH then
3039                             local temp = node.get_attribute(item, LOCALE)
3040                             if Babel.digits[temp] then
3041                                 local chr = item.char
3042                                 if chr > 47 and chr < 58 then
3043                                     item.char = Babel.digits[temp][chr-47]
3044                                 end
3045                             end
3046                         elseif item.id == node.id'math' then
3047                             inmath = (item.subtype == 0)
3048                         end
3049                     end
3050                     return head
3051                 end
3052             end
3053         }}}}
3054 \fi

```

```

3055 \fi
3056 % == alph, Alph ==
3057 % What if extras<lang> contains a \babel@save\@alph? It won't be
3058 % restored correctly when exiting the language, so we ignore
3059 % this change with the \bbl@alph@saved trick.
3060 \ifx\bbl@KVP@alph\@nil\else
3061   \toks@\expandafter\expandafter\expandafter{%
3062     \csname extras\language\endcsname}%
3063   \bbl@exp{%
3064     \def\<extras\language>{%
3065       \let\\bbl@alph@saved\\@alph
3066       \the\toks@
3067       \let\\@alph\\bbl@alph@saved
3068       \\babel@save\\@alph
3069       \let\\@alph<bbl@cntr@bbl@KVP@alph @\language>}}%
3070   \fi
3071 \ifx\bbl@KVP@Alph\@nil\else
3072   \toks@\expandafter\expandafter\expandafter{%
3073     \csname extras\language\endcsname}%
3074   \bbl@exp{%
3075     \def\<extras\language>{%
3076       \let\\bbl@Alph@saved\\@Alph
3077       \the\toks@
3078       \let\\@Alph\\bbl@Alph@saved
3079       \\babel@save\\@Alph
3080       \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language>}}%
3081   \fi
3082 % == require.babel in ini ==
3083 % To load or reload the babel-*.tex, if require.babel in ini
3084 \bbl@ifunset{bbl@rqtex@\language}{}%
3085   {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3086     \let\BabelBeforeIni@gobbletwo
3087     \chardef\atcatcode=\catcode\@
3088     \catcode\@=11\relax
3089     \bbl@input@texini{\bbl@cs{rqtex@\language}}%
3090     \catcode\@=\atcatcode
3091     \let\atcatcode\relax
3092   \fi}%
3093 % == main ==
3094 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3095   \let\language\bbl@savelangname
3096   \chardef\localeid\bbl@savelocaleid\relax
3097 \fi}

```

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

```

3098% TODO. Merge with \localenumeral:
3099% \newcommand\localedigits{\@nameuse{\language digits}}
3100\def\bbl@setdigits#1#2#3#4#5{%
3101  \bbl@exp{%
3102    \def\<\language digits>####1{%      ie, \langdigits
3103      \<bbl@digits@\language>####1\\@nil}%
3104      \let\<bbl@cntr@digits@\language>\<\language digits>%
3105      \def\<\language counter>####1{%    ie, \langcounter
3106        \\expandafter\<bbl@counter@\language>%
3107        \\csname c@####1\endcsname}%
3108      \def\<bbl@counter@\language>####1{% ie, \bbl@counter@lang
3109        \\expandafter\<bbl@digits@\language>%

```

[illegible]

```

3131 \def\bbl@provide@new#1{%
3132   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3133   \@namedef{extras#1}{}%
3134   \@namedef{noextras#1}{}%
3135   \bbl@startcommands*{#1}{captions}%
3136     \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3137       \def\bbl@tempb##1{% elt for \bbl@captionslist
3138         \ifx##1\@empty\else
3139           \bbl@exp{%
3140             \\SetString\\##1{%
3141               \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3142             \expandafter\bbl@tempb
3143             \fi}%
3144           \expandafter\bbl@tempb\bbl@captionslist\@empty
3145         \else
3146           \ifx\bbl@initoload\relax
3147             \bbl@read@ini{\bbl@KVP@captions}0% Here letters cat = 11
3148           \else
3149             \bbl@read@ini{\bbl@initoload}0% Here all letters cat = 11
3150           \fi
3151           \bbl@after@ini
3152           \bbl@savestrings
3153         \fi
3154       \StartBabelCommands*{#1}{date}%
3155       \ifx\bbl@KVP@import\@nil
3156         \bbl@exp{%
3157           \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
3158       \else
3159         \bbl@savetoday
3160         \bbl@savedate
3161       \fi
3162     \bbl@endcommands
3163     \bbl@load@basic{#1}%
3164     \bbl@exp{%
3165       \gdef\<#1hyphenmins>{%
3166         {\bbl@ifunset\bbl@lftm{#1}{2}{\bbl@cs{lftm{#1}}}%

```

```

3167     {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3168 \bbl@provide@hyphens{#1}%
3169 \ifx\bbl@KVP@main\@nil\else
3170     \expandafter\main@language\expandafter{#1}%
3171 \fi}
3172 \def\bbl@provide@renew#1{%
3173     \ifx\bbl@KVP@captions\@nil\else
3174         \StartBabelCommands*{#1}{captions}%
3175         \bbl@read@ini{\bbl@KVP@captions}0%   Here all letters cat = 11
3176         \bbl@after@ini
3177         \bbl@savestrings
3178         \EndBabelCommands
3179 \fi
3180 \ifx\bbl@KVP@import\@nil\else
3181     \StartBabelCommands*{#1}{date}%
3182     \bbl@savetoday
3183     \bbl@savedate
3184     \EndBabelCommands
3185 \fi
3186 % == hyphenrules ==
3187 \bbl@provide@hyphens{#1}}
3188 % Load the basic parameters (ids, typography, counters, and a few
3189 % more), while captions and dates are left out. But it may happen some
3190 % data has been loaded before automatically, so we first discard the
3191 % saved values.
3192 \def\bbl@load@basic#1{%
3193     \bbl@ifunset{\bbl@inidata@\languagename}{}%
3194     {\getlocaleproperty\bbl@tempa{\languagename}{identification/load.level}%
3195     \ifcase\bbl@tempa\else
3196         \bbl@csarg\let{lname@\languagename}\relax
3197     \fi}%
3198 \bbl@ifunset{\bbl@lname@#1}%
3199 {\def\BabelBeforeIni##1##2{%
3200     \begingroup
3201     \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3202     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14
3203     \let\bbl@ini@captions@aux\@gobbletwo
3204     \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3205     \bbl@read@ini{##1}0%
3206     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3207     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3208     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3209     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3210     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3211     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3212     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3213     \bbl@exportkey{intsp}{typography.intraspace}{}%
3214     \bbl@exportkey{chrng}{characters.ranges}{}%
3215     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3216     \ifx\bbl@initoload\relax\endinput\fi
3217 \endgroup}%
3218 \begingroup % boxed, to avoid extra spaces:
3219 \ifx\bbl@initoload\relax
3220     \bbl@input@texini{#1}%
3221 \else
3222     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3223 \fi
3224 \endgroup}%
3225 {}}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3226 \def\bbbl@provide@hyphens#1{%
3227   \let\bbbl@tempa\relax
3228   \ifx\bbbl@KVP@hyphenrules\@nil\else
3229     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3230     \bbbl@foreach\bbbl@KVP@hyphenrules{%
3231       \ifx\bbbl@tempa\relax % if not yet found
3232         \bbbl@ifsamestring{##1}{+}%
3233         {\bbbl@exp{\addlanguage\<l@##1>}}}%
3234         {}%
3235         \bbbl@ifunset{l@##1}%
3236         {}%
3237         {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}}%
3238     \fi}%
3239 \fi
3240 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
3241   \ifx\bbbl@KVP@import\@nil
3242     \ifx\bbbl@initoload\relax\else
3243       \bbbl@exp{% and hyphenrules is not empty
3244         \bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
3245         {}%
3246         {\let\bbbl@tempa\<l@bbbl@cl{hyphr}>}}}%
3247     \fi
3248   \else % if importing
3249     \bbbl@exp{% and hyphenrules is not empty
3250       \bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
3251       {}%
3252       {\let\bbbl@tempa\<l@bbbl@cl{hyphr}>}}}%
3253   \fi
3254 \fi
3255 \bbbl@ifunset{bbbl@tempa}% ie, relax or undefined
3256 {\bbbl@ifunset{l@#1}% no hyphenrules found - fallback
3257   {\bbbl@exp{\adddialect\<l@#1>\language}}}%
3258   {}% so, l@<lang> is ok - nothing to do
3259   {\bbbl@exp{\adddialect\<l@#1>\bbbl@tempa}}}% found in opt list or ini
3260

```

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

```

3261 \ifx\bbbl@readstream\@undefined
3262   \csname newread\endcsname\bbbl@readstream
3263 \fi
3264 \def\bbbl@input@texini#1{%
3265   \bbbl@bsphack
3266   \bbbl@exp{%
3267     \catcode\%%=14
3268     \lowercase{\InputIfFileExists{babel-#1.tex}}{}}%
3269     \catcode\%%=\the\catcode\%\relax}%
3270   \bbbl@esphack}
3271 \def\bbbl@inipreread#1=#2\@{%
3272   \bbbl@trim@def\bbbl@tempa{#1}% Redundant below !!
3273   \bbbl@trim\toks@{#2}%
3274   % Move trims here ??
3275   \bbbl@ifunset{bbbl@KVP@\bbbl@section/\bbbl@tempa}%
3276   {\bbbl@exp{%
3277     \g@addto@macro\bbbl@inidata{%
3278       \bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3279     \expandafter\bbbl@inireader\bbbl@tempa=#2\@}%

```

```

3280     {}}%
3281 \def\bbl@read@ini#1#2{%
3282   \bbl@csarg\edef{lini@language}{#1}%
3283   \openin\bbl@readstream=babel-#1.ini
3284   \ifeof\bbl@readstream
3285     \bbl@error
3286     {There is no ini file for the requested language\\%
3287      (#1). Perhaps you misspelled it or your installation\\%
3288      is not complete.}%
3289     {Fix the name or reinstall babel.}%
3290   \else
3291     \bbl@exp{\def\\bbl@inidata{%
3292       \\bbl@elt{identification}{tag.ini}{#1}%
3293       \\bbl@elt{identification}{load.level}{#2}}}%
3294     \let\bbl@section\@empty
3295     \let\bbl@savestrings\@empty
3296     \let\bbl@savetoday\@empty
3297     \let\bbl@savedate\@empty
3298     \let\bbl@inireader\bbl@iniskip
3299     \bbl@info{Importing
3300               \ifcase#2 \or font and identification \or basic \fi
3301               data for \language\\%
3302               from babel-#1.ini. Reported}%
3303     \loop
3304     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3305       \endlinechar\m@ne
3306       \read\bbl@readstream to \bbl@line
3307       \endlinechar\^^M
3308       \ifx\bbl@line\@empty\else
3309         \expandafter\bbl@iniline\bbl@line\bbl@iniline
3310       \fi
3311     \repeat
3312     \bbl@foreach\bbl@renewlist{%
3313       \bbl@ifunset{bbl@renew@##1}{\bbl@inisec[##1]\@}%
3314     \global\let\bbl@renewlist\@empty
3315     % Ends last section. See \bbl@inisec
3316     \def\bbl@elt##1##2{\bbl@inireader##1=##2\@}%
3317     \bbl@cs{renew@\bbl@section}%
3318     \global\bbl@csarg\let{renew@\bbl@section}\relax
3319     \bbl@cs{secpost@\bbl@section}%
3320     \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3321     \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
3322     \bbl@tglobal\bbl@ini@loaded
3323   \fi}
3324 \def\bbl@iniline#1\bbl@iniline{%
3325   \@ifnextchar[\bbl@inisec{\@ifnextchar\bbl@iniskip\bbl@inipreread}#1\@}% ]

```

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

```

3326 \def\bbl@iniskip#1\@{%           if starts with ;
3327 \def\bbl@inisec[#1]#2\@{%       if starts with opening bracket
3328   \def\bbl@elt##1##2{%
3329     \expandafter\toks@\expandafter{%
3330       \expandafter{\bbl@section}{##1}{##2}}%
3331     \bbl@exp{%
3332       \\g@addto@macro\\bbl@inidata{\\bbl@elt\the\toks@}}%

```

```

3333 \bbl@inireader##1=##2\@}%
3334 \bbl@cs{renew@bbl@section}%
3335 \global\bbl@csarg\let{renew@bbl@section}\relax
3336 \bbl@cs{secpost@bbl@section}%
3337 % The previous code belongs to the previous section.
3338 % -----
3339 % Now start the current one.
3340 \in@{=date.}{=#1}%
3341 \ifin@
3342 \lowercase{\def\bbl@tempa{=#1}}%
3343 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3344 \bbl@replace\bbl@tempa{=date.}{}%
3345 \in@{.licr=}{#1}%
3346 \ifin@
3347 \ifcase\bbl@engine
3348 \bbl@replace\bbl@tempa{.licr=}{}%
3349 \else
3350 \let\bbl@tempa\relax
3351 \fi
3352 \fi
3353 \ifx\bbl@tempa\relax\else
3354 \bbl@replace\bbl@tempa{=}{}%
3355 \bbl@exp{%
3356 \def\bbl@inikv@#1>####1=####2\\@%
3357 \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}%
3358 \fi
3359 \fi
3360 \def\bbl@section{#1}%
3361 \def\bbl@elt##1##2{%
3362 \@namedef{bbl@KVP@#1/##1}{}}%
3363 \bbl@cs{renew@#1}%
3364 \bbl@cs{secpre@#1}% pre-section 'hook'
3365 \bbl@ifunset{bbl@inikv@#1}%
3366 {\let\bbl@inireader\bbl@iniskip}%
3367 {\bbl@exp{\let\\bbl@inireader<bbl@inikv@#1>}}
3368 \let\bbl@renewlist@empty
3369 \def\bbl@renewinikv#1/#2\@#3{%
3370 \bbl@ifunset{bbl@renew@#1}%
3371 {\bbl@add@list\bbl@renewlist{#1}}%
3372 {}}%
3373 \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}

```

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

```

3374 \def\bbl@inikv#1=#2\@% key=value
3375 \bbl@trim@def\bbl@tempa{#1}%
3376 \bbl@trim\toks@{#2}%
3377 \bbl@csarg\edef{kv@\bbl@section.\bbl@tempa}{\the\toks@}

```

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

```

3378 \def\bbl@exportkey#1#2#3{%
3379 \bbl@ifunset{bbl@kv@#2}%
3380 {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3381 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3382 \bbl@csarg\gdef{#1@\languagename}{#3}}%
3383 \else
3384 \bbl@exp{\global\let<bbl@#1@\languagename>\bbl@kv@#2}%
3385 \fi}}

```

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

```

3386 \def\bbl@iniwarning#1{%
3387   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3388   {\bbl@warning{%
3389     From babel-\bbl@cs{lini@language}.ini:\%
3390     \bbl@cs{@kv@identification.warning#1}\%
3391     Reported }}}
3392 \let\bbl@inikv@identification\bbl@inikv
3393 \def\bbl@secpost@identification{%
3394   \bbl@iniwarning{%
3395     \ifcase\bbl@engine
3396       \bbl@iniwarning{.pdflatex}%
3397     \or
3398       \bbl@iniwarning{.lualatex}%
3399     \or
3400       \bbl@iniwarning{.xelatex}%
3401     \fi%
3402     \bbl@exportkey{elname}{identification.name.english}{}%
3403     \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3404       {\csname bbl@elname@language\endcsname}}%
3405     \bbl@exportkey{lbcpl}{identification.tag.bcp47}{% TODO
3406     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3407     \bbl@exportkey{esname}{identification.script.name}{}%
3408     \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3409       {\csname bbl@esname@language\endcsname}}%
3410     \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{%
3411     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3412     \ifbbl@bcptoname
3413       \bbl@csarg\xdef{bcp@map@bbl@cl{lbcpl}}{language}%
3414     \fi}
3415 \let\bbl@inikv@typography\bbl@inikv
3416 \let\bbl@inikv@characters\bbl@inikv
3417 \let\bbl@inikv@numbers\bbl@inikv
3418 \def\bbl@inikv@counters#1=#2@@{%
3419   \bbl@ifsamestring{#1}{digits}%
3420   {\bbl@error{The counter name 'digits' is reserved for mapping\%
3421     decimal digits}%
3422     {Use another name.}}%
3423   {%
3424   \def\bbl@tempc{#1}%
3425   \bbl@trim@def{\bbl@tempb*}{#2}%
3426   \in@{.1$}{#1$}%
3427   \ifin@
3428     \bbl@replace\bbl@tempc{.1}{}%
3429     \bbl@csarg\protected@xdef{cntr@bbl@tempc @language}{%
3430       \noexpand\bbl@alphnumeral{\bbl@tempb*}}%
3431   \fi
3432   \in@{.F.}{#1}%
3433   \ifin@else\in@{.S.}{#1}\fi
3434   \ifin@
3435     \bbl@csarg\protected@xdef{cntr@#1@language}{\bbl@tempb*}%
3436   \else
3437     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3438     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \ \
3439     \bbl@csarg{\global\expandafter\let}{cntr@#1@language}\bbl@tempa

```



```

3440 \fi}
3441 \def\bbl@after@ini{%
3442 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3443 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3444 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3445 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3446 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3447 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3448 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3449 \bbl@exportkey{intsp}{typography.intraspace}{}%
3450 \bbl@exportkey{jstfy}{typography.justify}{w}%
3451 \bbl@exportkey{chrng}{characters.ranges}{}%
3452 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3453 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3454 \bbl@tglobal\bbl@savetoday
3455 \bbl@tglobal\bbl@savestate}

```

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

```

3456 \ifcase\bbl@engine
3457 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3458 \bbl@ini@captions@aux{#1}{#2}}
3459 \else
3460 \def\bbl@inikv@captions#1=#2\@@{%
3461 \bbl@ini@captions@aux{#1}{#2}}
3462 \fi

```

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

```

3463 \def\bbl@ini@captions@aux#1#2{%
3464 \bbl@trim\def\bbl@tempa{#1}%
3465 \bbl@ifblank{#2}%
3466 {\bbl@exp{%
3467 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3468 {\bbl@trim\toks@{#2}}}%
3469 \bbl@exp{%
3470 \bbl@add\bbl@savestrings{%
3471 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3472 \toks@\expandafter{\bbl@captionslist}%
3473 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3474 \ifin\else
3475 \bbl@exp{%
3476 \bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3477 \bbl@tglobal\<\bbl@extracaps@\language>}%
3478 \fi}

```

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

```

3479 \def\bbl@bktoname#1\@@{\csname#1name\endcsname} % TODO - ugly
3480 \def\bbl@bktothe#1\@@{\csname the#1\endcsname}
3481 \def\bbl@list@the{%
3482 part,chapter,section,subsection,subsubsection,paragraph,%
3483 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3484 table,page,footnote,mpfootnote,mpfn} % Include \thempfn?
3485 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3486 \bbl@ifunset{bbl@map@#1@\language}%
3487 {\@nameuse{#1}}%
3488 {\@nameuse{bbl@map@#1@\language}}}
3489 \def\bbl@inikv@labels#1=#2\@@{%

```

```

3490 \in@{map.}{#1}%
3491 \ifin@
3492 \def\bbl@tempc{#1}%
3493 \bbl@replace\bbl@tempc{map.}{}%
3494 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,}%
3495 \bbl@exp{%
3496 \gdef\<bbl@map@\bbl@tempc @\language\name>%
3497 {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3498 \bbl@foreach\bbl@list@the{%
3499 \bbl@exp{\let\\bbl@tempd\<the##1>}%
3500 \bbl@exp{%
3501 \\bbl@sreplace\<the##1>%
3502 {\<\bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3503 \\bbl@sreplace\<the##1>%
3504 {\<\empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3505 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3506 \toks@\expandafter\expandafter\expandafter{%
3507 \csname the##1\endcsname}
3508 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3509 \fi}%
3510 %
3511 \else
3512 \def\bbl@toreplace{#2}%
3513 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3514 \bbl@replace\bbl@toreplace{[ ]}{\bbl@bktoname}%
3515 \bbl@replace\bbl@toreplace{[ ]}{\bbl@bktothe}%
3516 \bbl@replace\bbl@toreplace{[ ]}{\@@}%
3517 \bbl@replace\bbl@toreplace{[ ]}{\@@}%
3518 \in@{,#1,}{,chapter,}%
3519 \ifin@
3520 \bbl@patchchapter
3521 \global\bbl@csarg\let{chapfmt@\language\name}\bbl@toreplace
3522 \fi
3523 \in@{,#1,}{,appendix,}%
3524 \ifin@
3525 \bbl@patchchapter
3526 \global\bbl@csarg\let{appxfmt@\language\name}\bbl@toreplace
3527 \fi
3528 \in@{,#1,}{,part,}%
3529 \ifin@
3530 \bbl@patchpart
3531 \global\bbl@csarg\let{partfmt@\language\name}\bbl@toreplace
3532 \fi
3533 \in@{,#1,}{,figure,table,}%
3534 \ifin@
3535 \toks@\expandafter{\bbl@toreplace}%
3536 \bbl@exp{\gdef\<fnum@#1>{\the\toks@}}%
3537 \fi
3538 %
3539 \in@{enumerate.}{#1}%
3540 \ifin@
3541 \def\bbl@tempa{#1}%
3542 \bbl@replace\bbl@tempa{enumerate.}{}%
3543 \toks@\expandafter{\bbl@toreplace}%
3544 \bbl@exp{%
3545 \\bbl@add\<extras\language\name>{%
3546 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3547 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3548 \\bbl@tglobal\<extras\language\name>}%

```

```

3549 \fi
3550 \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.

```

3551 \def\bbl@chapttype{chap}
3552 \ifx\@makechapterhead\@undefined
3553 \let\bbl@patchchapter\relax
3554 \else
3555 \def\bbl@patchchapter{%
3556 \global\let\bbl@patchchapter\relax
3557 \bbl@add\appendix{\def\bbl@chapttype{appx}}}% Not harmful, I hope
3558 \bbl@tglobal\appendix
3559 \bbl@sreplace\@makechapterhead
3560 {\@chapapp\space\thechapter}%
3561 {\bbl@chapterformat}%
3562 \bbl@tglobal\@makechapterhead
3563 \gdef\bbl@chapterformat{%
3564 \bbl@ifunset{\bbl@bbl@chapttype fmt@\languagename}%
3565 {\@chapapp\space\thechapter}
3566 {\@nameuse{\bbl@bbl@chapttype fmt@\languagename}}}}
3567 \fi
3568 \ifx\@part\@undefined
3569 \let\bbl@patchpart\relax
3570 \else
3571 \def\bbl@patchpart{%
3572 \global\let\bbl@patchpart\relax
3573 \bbl@sreplace\@part
3574 {\partname\nobreakspace\thepart}%
3575 {\bbl@partformat}%
3576 \bbl@tglobal\@part
3577 \gdef\bbl@partformat{%
3578 \bbl@ifunset{\bbl@partfmt@\languagename}%
3579 {\partname\nobreakspace\thepart}
3580 {\@nameuse{\bbl@partfmt@\languagename}}}}
3581 \fi

```

Date. TODO. Document

```

3582 % Arguments are _not_ protected.
3583 \let\bbl@calendar\@empty
3584 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3585 \def\bbl@cased{% TODO. Move
3586 \ifx\oe\OE
3587 \expandafter\in@\expandafter
3588 {\expandafter\OE\expandafter}\expandafter{\oe}%
3589 \ifin@
3590 \bbl@afterelse\expandafter\MakeUppercase
3591 \else
3592 \bbl@afterfi\expandafter\MakeLowercase
3593 \fi
3594 \else
3595 \expandafter\@firstofone
3596 \fi}
3597 \def\bbl@localedate#1#2#3#4{%
3598 \begingroup
3599 \ifx\@empty#1\@empty\else
3600 \let\bbl@ld@calendar\@empty

```

```

3601 \let\bbl@ld@variant\@empty
3602 \edef\bbl@tempa{\zap@space#1 \@empty}%
3603 \def\bbl@tempb##1==##2\@@{\@namedef\bbl@ld@##1}{##2}}%
3604 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3605 \edef\bbl@calendar{%
3606   \bbl@ld@calendar
3607   \ifx\bbl@ld@variant\@empty\else
3608     .\bbl@ld@variant
3609   \fi}%
3610 \bbl@replace\bbl@calendar{gregorian}{}%
3611 \fi
3612 \bbl@cased
3613 {\@nameuse\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}}%
3614 \endgroup}
3615 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3616 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3617   \bbl@trim@def\bbl@tempa{#1.#2}%
3618   \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3619   {\bbl@trim@def\bbl@tempa{#3}%
3620     \bbl@trim\toks@{#5}%
3621     \@temptokena\expandafter{\bbl@savedate}%
3622     \bbl@exp{% Reverse order - in ini last wins
3623       \def\\bbl@savedate{%
3624         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3625         \the\@temptokena}}}%
3626   {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3627     {\lowercase{\def\bbl@tempb{#6}}%
3628       \bbl@trim@def\bbl@toreplace{#5}%
3629       \bbl@TG@@date
3630       \bbl@ifunset\bbl@date@\language @}%
3631       {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3632         % TODO. Move to a better place.
3633         \bbl@exp{%
3634           \gdef\<\language date>{\protect\<\language date >}%
3635           \gdef\<\language date >####1####2####3{%
3636             \\bbl@usedategrouptrue
3637             \<bbl@ensure@\language >{%
3638               \\localedate{####1}{####2}{####3}}}%
3639             \\bbl@add\\bbl@savetoday{%
3640               \\SetString\\today{%
3641                 \<\language date>%
3642                 {\the\year}{\the\month}{\the\day}}}}}%
3643         {}%
3644       \ifx\bbl@tempb\@empty\else
3645         \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3646       \fi}%
3647     {}}}}

```

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.

```

3648 \let\bbl@calendar\@empty
3649 \newcommand\BabelDateSpace{\nobreakspace}
3650 \newcommand\BabelDateDot{\.} % TODO. \let instead of repeating
3651 \newcommand\BabelDated[1]{\number#1}
3652 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3653 \newcommand\BabelDateM[1]{\number#1}
3654 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3655 \newcommand\BabelDateMMMM[1]{%

```

```

3656 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3657 \newcommand\BabelDatey[1]{\number#1}%
3658 \newcommand\BabelDateyy[1]{%
3659 \ifnum#1<10 0\number#1 %
3660 \else\ifnum#1<100 \number#1 %
3661 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3662 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3663 \else
3664 \bbl@error
3665 {Currently two-digit years are restricted to the\
3666 range 0-9999.}%
3667 {There is little you can do. Sorry.}%
3668 \fi\fi\fi\fi}}
3669 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3670 \def\bbl@replace@finish@iii#1{%
3671 \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
3672 \def\bbl@TG@date{%
3673 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3674 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot{}}%
3675 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3676 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3677 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3678 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3679 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3680 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3681 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3682 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3683 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3684 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3685 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3686 % Note after \bbl@replace \toks@ contains the resulting string.
3687 % TODO - Using this implicit behavior doesn't seem a good idea.
3688 \bbl@replace@finish@iii\bbl@toreplace}
3689 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3690 \def\bbl@xdatecctr[#1|#2]{\localnumeral{#2}{#1}}

```

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

```

3691 \def\bbl@provide@lsys#1{%
3692 \bbl@ifunset{bbl@lname@#1}%
3693 {\bbl@ini@basic{#1}}%
3694 {}%
3695 \bbl@csarg\let{lsys@#1}\@empty
3696 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3697 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3698 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3699 \bbl@ifunset{bbl@lname@#1}{%
3700 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3701 \ifcase\bbl@engine\or\or
3702 \bbl@ifunset{bbl@prehc@#1}{}%
3703 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3704 {}%
3705 {\ifx\bbl@xenoxyph\@undefined
3706 \let\bbl@xenoxyph\bbl@xenoxyph@d
3707 \ifx\AtBeginDocument\@notprerr
3708 \expandafter\@secondoftwo % to execute right now
3709 \fi
3710 \AtBeginDocument{%
3711 \expandafter\bbl@add

```

```

3712         \csname selectfont \endcsname{\bbl@xenoHYph}%
3713         \expandafter\selectlanguage\expandafter{\languageName}%
3714         \expandafter\bbl@tOglobal\csname selectfont \endcsname}%
3715     \fi}}%
3716 \fi
3717 \bbl@csarg\bbl@tOglobal{lsys@#1}}
3718 \def\bbl@ifset#1#2#3{%    TODO. Move to the correct place.
3719     \bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{#1}{#3}{#2}}}%
3720 \def\bbl@xenoHYph@d{%
3721     \bbl@ifset{bbl@prehc@languageName}%
3722     {\ifnum\hyphenchar\font=\defaultHyphenchar
3723         \iffontchar\font\bbl@cL{prehc}\relax
3724         \hyphenchar\font\bbl@cL{prehc}\relax
3725         \else\iffontchar\font"200B
3726         \hyphenchar\font"200B
3727         \else
3728             \bbl@error
3729             {Neither 0 nor ZERO WIDTH SPACE are available\\%
3730              in the current font, and therefore the hyphen\\%
3731              will be printed. Try with 'HyphenChar', but be\\%
3732              aware this setting is not safe (see the manual).}%
3733             {See the manual.}%
3734             \hyphenchar\font\defaultHyphenchar
3735         \fi\fi
3736     \fi}%
3737     {\hyphenchar\font\defaultHyphenchar}}
3738 % \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).

```

3739 \def\bbl@ini@basic#1{%
3740     \def\BabelBeforeIni##1##2{%
3741         \begingroup
3742         \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
3743         \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3744         \catcode`\;=12 \catcode`\|=12 \catcode`\%=14
3745         \bbl@read@ini{##1}1%
3746         \endinput           % babel- .tex may contain onlypreamble's
3747         \endgroup}%         boxed, to avoid extra spaces:
3748     {\bbl@input@texini{#1}}}

```

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

```

3749 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3750     \ifx\\#1%             % \ before, in case #1 is multiletter
3751         \bbl@exp{%
3752             \def\\bbl@tempa####1{%
3753                 \ifcase>####1\space\the\toks@<\else>\\@ctrerr<\fi>}}%
3754         \else
3755             \toks@\expandafter{\the\toks@or #1}%
3756             \expandafter\bbl@buildifcase
3757         \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the

subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3758 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\language}\{#2}}
3759 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3760 \newcommand\localecounter[2]{%
3761   \expandafter\bbl@localecntr
3762   \expandafter{\number\csname c@#2\endcsname}\{#1}}
3763 \def\bbl@alphnumeral#1#2{%
3764   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3765 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3766   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
3767     \bbl@alphnumeral@ii{#9}000000#1\or
3768     \bbl@alphnumeral@ii{#9}00000#1#2\or
3769     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3770     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3771     \bbl@alphnum@invalid{>9999}%
3772   \fi}
3773 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3774   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3775     {\bbl@cs{cntr@#1.4@\language}\{#5}}
3776     {\bbl@cs{cntr@#1.3@\language}\{#6}}
3777     {\bbl@cs{cntr@#1.2@\language}\{#7}}
3778     {\bbl@cs{cntr@#1.1@\language}\{#8}}
3779     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3780       \bbl@ifunset{bbl@cntr@#1.S.321@\language}\{#5}}
3781       {\bbl@cs{cntr@#1.S.321@\language}\{#5}}
3782     \fi}%
3783   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}
3784 \def\bbl@alphnum@invalid#1{%
3785   \bbl@error{Alphabetic numeral too large (#1)}%
3786   {Currently this is the limit.}}

```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

3787 \newcommand\localeinfo[1]{%
3788   \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\language}%
3789     {\bbl@error{I've found no info for the current locale.\%
3790       The corresponding ini file has not been loaded\%
3791       Perhaps it doesn't exist}%
3792     {\See the manual for details.}}%
3793   {\bbl@cs{\csname bbl@info@#1\endcsname @\language}}
3794 % \@namedef{bbl@info@name.locale}\{lname}
3795 \@namedef{bbl@info@tag.ini}\{lini}
3796 \@namedef{bbl@info@name.english}\{elname}
3797 \@namedef{bbl@info@name.opentype}\{lname}
3798 \@namedef{bbl@info@tag.bcp47}\{lbc} % TODO
3799 \@namedef{bbl@info@tag.opentype}\{lotf}
3800 \@namedef{bbl@info@script.name}\{esname}
3801 \@namedef{bbl@info@script.name.opentype}\{sname}
3802 \@namedef{bbl@info@script.tag.bcp47}\{sbcp}
3803 \@namedef{bbl@info@script.tag.opentype}\{sotf}
3804 \let\bbl@ensureinfo\@gobble
3805 \newcommand\BabelEnsureInfo{%
3806   \ifx\InputIfFileExists\undefined\else
3807     \def\bbl@ensureinfo##1{%
3808       \bbl@ifunset{bbl@lname@##1}\{bbl@ini@basic{##1}}\{}}%
3809   \fi
3810   \bbl@foreach\bbl@loaded{%

```

```

3811 \def\language{##1}%
3812 \bbl@ensureinfo{##1}}}}

More general, but non-expandable, is \getlocaleproperty. To inspect every possible
loaded ini, we define \LocaleForEach, where \bbl@ini@loaded is a comma-separated list
of locales, built by \bbl@read@ini.

3813 \newcommand\getlocaleproperty{%
3814 \ifstar\bbl@getproperty@s\bbl@getproperty@x%
3815 \def\bbl@getproperty@s#1#2#3{%
3816 \let#1\relax
3817 \def\bbl@elt##1##2##3{%
3818 \bbl@ifsamestring{##1/##2}{##3}%
3819 {\providecommand#1{##3}%
3820 \def\bbl@elt####1####2####3{}}}%
3821 {}}%
3822 \bbl@cs{inidata@#2}}%
3823 \def\bbl@getproperty@x#1#2#3{%
3824 \bbl@getproperty@s{#1}{#2}{#3}%
3825 \ifx#1\relax
3826 \bbl@error
3827 {Unknown key for locale '#2':\%
3828 #3\%
3829 \string#1 will be set to \relax}%
3830 {Perhaps you misspelled it.}%
3831 \fi}
3832 \let\bbl@ini@loaded\@empty
3833 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel behavior

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

```

3834 \newcommand\babeladjust[1]{% TODO. Error handling.
3835 \bbl@forkv{#1}{%
3836 \bbl@ifunset{\bbl@ADJ@##1@##2}%
3837 {\bbl@cs{ADJ@##1}{##2}}%
3838 {\bbl@cs{ADJ@##1@##2}}}
3839 %
3840 \def\bbl@adjust@lua#1#2{%
3841 \ifvmode
3842 \ifnum\currentgrouplevel=\z@
3843 \directlua{ Babel.#2 }%
3844 \expandafter\expandafter\expandafter\@gobble
3845 \fi
3846 \fi
3847 {\bbl@error % The error is gobbled if everything went ok.
3848 {Currently, #1 related features can be adjusted only\%
3849 in the main vertical list.}%
3850 {Maybe things change in the future, but this is what it is.}}}
3851 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3852 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3853 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3854 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3855 \@namedef{\bbl@ADJ@bidi.text@on}{%
3856 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3857 \@namedef{\bbl@ADJ@bidi.text@off}{%
3858 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3859 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%

```



```

3860 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3861 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3862 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3863 %
3864 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3865 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3866 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3867 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3868 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3869 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3870 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3871 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3872 %
3873 \def\bbl@adjust@layout#1{%
3874 \ifvmode
3875 #1%
3876 \expandafter\@gobble
3877 \fi
3878 {\bbl@error % The error is gobbled if everything went ok.
3879 {Currently, layout related features can be adjusted only\\%
3880 in vertical mode.}%
3881 {Maybe things change in the future, but this is what it is.}}}
3882 \@namedef{bbl@ADJ@layout.tabular@on}{%
3883 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3884 \@namedef{bbl@ADJ@layout.tabular@off}{%
3885 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3886 \@namedef{bbl@ADJ@layout.lists@on}{%
3887 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3888 \@namedef{bbl@ADJ@layout.lists@off}{%
3889 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3890 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3891 \bbl@activateposthyphen}
3892 %
3893 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3894 \bbl@bcpallowedtrue}
3895 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3896 \bbl@bcpallowedfalse}
3897 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3898 \def\bbl@bcp@prefix{#1}}
3899 \def\bbl@bcp@prefix{bcp47-}
3900 \@namedef{bbl@ADJ@autoload.options#1}{%
3901 \def\bbl@autoload@options{#1}}
3902 \let\bbl@autoload@bcptoptions\@empty
3903 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3904 \def\bbl@autoload@bcptoptions{#1}}
3905 \newif\ifbbl@bcptoname
3906 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3907 \bbl@bcptonametrue}
3908 \BabelEnsureInfo}
3909 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3910 \bbl@bcptonamefalse}
3911 % TODO: use babel name, override
3912 %
3913 % As the final task, load the code for lua.
3914 %
3915 \ifx\directlua\@undefined\else
3916 \ifx\bbl@luapatterns\@undefined
3917 \input luababel.def
3918 \fi

```

```

3919 \fi
3920 </core>

A proxy file for switch.def

3921 <*kernel>
3922 \let\bbl@onlyswitch\@empty
3923 \input babel.def
3924 \let\bbl@onlyswitch\@undefined
3925 </kernel>
3926 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by $\text{iniT}_\text{E}\text{X}$ because it should instruct $\text{T}_\text{E}\text{X}$ to read hyphenation patterns. To this end the `docstrip` option `patterns` can be used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

To make sure that $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```

3927 <<Make sure ProvidesFile is defined>>
3928 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
3929 \xdef\bbl@format{\jobname}
3930 \def\bbl@version{\<version>}
3931 \def\bbl@date{\<date>}
3932 \ifx\AtBeginDocument\@undefined
3933   \def\@empty{}
3934   \let\orig@dump\dump
3935   \def\dump{%
3936     \ifx\@ztryfc\@undefined
3937     \else
3938       \toks0=\expandafter{\@preamblecmds}%
3939       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3940       \def\@begindocumenthook{}%
3941     \fi
3942     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3943 \fi
3944 <<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.

```

3945 \def\process@line#1#2 #3 #4 {%
3946   \ifx=#1%
3947     \process@synonym{#2}%
3948   \else
3949     \process@language{#1#2}{#3}{#4}%
3950   \fi
3951   \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.

```

3952 \toks@{}
3953 \def\bbl@languages{}

```

When no languages have been loaded yet, the name following the = will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The \relax just helps to the \if below catching synonyms without a language.)

Otherwise the name will be a synonym for the language loaded last.

We also need to copy the hyphenmin parameters for the synonym.

```

3954 \def\process@synonym#1{%
3955   \ifnum\last@language=\m@ne
3956     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3957   \else
3958     \expandafter\chardef\csname l@#1\endcsname\last@language
3959     \wlog{\string\l@#1=\string\language\the\last@language}%
3960     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3961       \csname\language\hyphenmins\endcsname
3962     \let\bbl@elt\relax
3963     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
3964   \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` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

```

3965 \def\process@language#1#2#3{%
3966   \expandafter\addlanguage\csname l@#1\endcsname
3967   \expandafter\language\csname l@#1\endcsname
3968   \edef\language{#1}%

```

```

3969 \bbl@hook@everylanguage{#1}%
3970 % > luatex
3971 \bbl@get@enc#1::\@@@
3972 \begingroup
3973 \leftthyphenmin\m@ne
3974 \bbl@hook@loadpatterns{#2}%
3975 % > luatex
3976 \ifnum\leftthyphenmin=\m@ne
3977 \else
3978 \expandafter\xdef\csname #1hyphenmins\endcsname{%
3979 \the\leftthyphenmin\the\rightthyphenmin}%
3980 \fi
3981 \endgroup
3982 \def\bbl@tempa{#3}%
3983 \ifx\bbl@tempa\@empty\else
3984 \bbl@hook@loadexceptions{#3}%
3985 % > luatex
3986 \fi
3987 \let\bbl@elt\relax
3988 \edef\bbl@languages{%
3989 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3990 \ifnum\the\language=\z@
3991 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3992 \set@hyphenmins\tw@\thr@@\relax
3993 \else
3994 \expandafter\expandafter\expandafter\set@hyphenmins
3995 \csname #1hyphenmins\endcsname
3996 \fi
3997 \the\toks@
3998 \toks@{}}%
3999 \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.

```

4000 \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.

```

4001 \def\bbl@hook@everylanguage#1{}
4002 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4003 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4004 \def\bbl@hook@loadkernel#1{%
4005 \def\addlanguage{\csname newlanguage\endcsname}%
4006 \def\adddialect##1##2{%
4007 \global\chardef##1##2\relax
4008 \wlog{\string##1 = a dialect from \string\language##2}}%
4009 \def\iflanguage#1{%
4010 \expandafter\ifx\csname l@##1\endcsname\relax
4011 \@nolanerr{##1}%
4012 \else
4013 \ifnum\csname l@##1\endcsname=\language
4014 \expandafter\expandafter\expandafter\@firstoftwo
4015 \else
4016 \expandafter\expandafter\expandafter\@secondoftwo
4017 \fi
4018 \fi}%
4019 \def\providehyphenmins##1##2{%

```

```

4020 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4021 \namedef{##1hyphenmins}{##2}%
4022 \fi}%
4023 \def\set@hyphenmins##1##2{%
4024 \leftthyphenmin##1\relax
4025 \rightthyphenmin##2\relax}%
4026 \def\selectlanguage{%
4027 \errhelp{Selecting a language requires a package supporting it}%
4028 \errmessage{Not loaded}}%
4029 \let\foreignlanguage\selectlanguage
4030 \let\otherlanguage\selectlanguage
4031 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4032 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4033 \def\setlocale{%
4034 \errhelp{Find an armchair, sit down and wait}%
4035 \errmessage{Not yet available}}%
4036 \let\uselocale\setlocale
4037 \let\locale\setlocale
4038 \let\selectlocale\setlocale
4039 \let\localename\setlocale
4040 \let\textlocale\setlocale
4041 \let\textlanguage\setlocale
4042 \let\languagegettext\setlocale}
4043 \begingroup
4044 \def\AddBabelHook#1#2{%
4045 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4046 \def\next{\toks1}%
4047 \else
4048 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4049 \fi
4050 \next}
4051 \ifx\directlua\@undefined
4052 \ifx\XeTeXinputencoding\@undefined\else
4053 \input xebabel.def
4054 \fi
4055 \else
4056 \input luababel.def
4057 \fi
4058 \openin1 = babel-\bbl@format.cfg
4059 \ifeof1
4060 \else
4061 \input babel-\bbl@format.cfg\relax
4062 \fi
4063 \closein1
4064 \endgroup
4065 \bbl@hook@loadkernel{switch.def}

```

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

```

4066 \openin1 = language.dat

```

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

```

4067 \def\languagename{english}%
4068 \ifeof1
4069 \message{I couldn't find the file language.dat,\space
4070 I will try the file hyphen.tex}
4071 \input hyphen.tex\relax
4072 \chardef\l@english\z@
4073 \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`.

```
4074 \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.

```
4075 \loop
4076   \endlinechar\m@ne
4077   \read1 to \bbl@line
4078   \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.

```
4079   \if T\ifeof1F\fi T\relax
4080   \ifx\bbl@line\@empty\else
4081     \edef\bbl@line{\bbl@line\space\space\space}%
4082     \expandafter\process@line\bbl@line\relax
4083   \fi
4084 \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.

```
4085 \begingroup
4086   \def\bbl@elt#1#2#3#4{%
4087     \global\language=#2\relax
4088     \gdef\language#1}%
4089   \def\bbl@elt##1##2##3##4{}}%
4090   \bbl@languages
4091 \endgroup
4092 \fi
4093 \closein1
```

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

```
4094 \if/\the\toks@/\else
4095   \errhelp{language.dat loads no language, only synonyms}
4096   \errmessage{Orphan language synonym}
4097 \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.

```
4098 \let\bbl@line\@undefined
4099 \let\process@line\@undefined
4100 \let\process@synonym\@undefined
4101 \let\process@language\@undefined
4102 \let\bbl@get@enc\@undefined
4103 \let\bbl@hyph@enc\@undefined
4104 \let\bbl@tempa\@undefined
4105 \let\bbl@hook@loadkernel\@undefined
4106 \let\bbl@hook@everylanguage\@undefined
4107 \let\bbl@hook@loadpatterns\@undefined
4108 \let\bbl@hook@loadexceptions\@undefined
4109 \</patterns>
```

Here the code for `iniTEX` ends.

12 Font handling with fontspec

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

```
4110 <<*More package options>> ≡
4111 \chardef\bbl@bidimode\z@
4112 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4113 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4114 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4115 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4116 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4117 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4118 <</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`.

```
4119 <<*Font selection>> ≡
4120 \bbl@trace{Font handling with fontspec}
4121 \@onlypreamble\babelfont
4122 \newcommand\babelfont[2][% 1=langs/scripts 2=fam
4123   \bbl@foreach{#1}{%
4124     \expandafter\ifx\csname date##1\endcsname\relax
4125     \IfFileExists{babel-##1.tex}%
4126       {\babelprovide{##1}}%
4127       {}%
4128     \fi}%
4129 \edef\bbl@tempa{#1}%
4130 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4131 \ifx\fontspec\undefined
4132   \usepackage{fontspec}%
4133 \fi
4134 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4135 \bbl@bblfont}
4136 \newcommand\bbl@bblfont[2][% 1=features 2=fontname, @font=rm|sf|tt
4137   \bbl@ifunset{\bbl@tempb family}%
4138     {\bbl@providedefam{\bbl@tempb}}%
4139     {\bbl@exp{%
4140       \\\bbl@sreplace\<\bbl@tempb family >%
4141       {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4142 % For the default font, just in case:
4143 \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}}%
4144 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4145   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4146   \bbl@exp{%
4147     \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4148     \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4149     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4150   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4151     \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}}}%
4152 \def\bbl@providedefam#1{%
4153   \bbl@exp{%
4154     \\\newcommand\<#1default>{}% Just define it
4155     \\\bbl@add@list\<\bbl@font@fams{#1}%
4156     \\\DeclareRobustCommand\<#1family>{%
```

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

```
4152 \def\bbl@providedefam#1{%
4153   \bbl@exp{%
4154     \\\newcommand\<#1default>{}% Just define it
4155     \\\bbl@add@list\<\bbl@font@fams{#1}%
4156     \\\DeclareRobustCommand\<#1family>{%
```

```

4157      \\not@math@alphabet\<#1family>\relax
4158      \\fontfamily\<#1default>\\selectfont}%
4159      \\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.

```

4160 \def\bbl@nostdfont#1{%
4161   \bbl@ifunset{\bbl@WFF@\f@family}%
4162   {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
4163   \bbl@infowarn{The current font is not a babel standard family:\\%
4164     #1%
4165     \fontname\font\\%
4166     There is nothing intrinsically wrong with this warning, and\\%
4167     you can ignore it altogether if you do not need these\\%
4168     families. But if they are used in the document, you should be\\%
4169     aware 'babel' will no set Script and Language for them, so\\%
4170     you may consider defining a new family with \string\babelfont.\\%
4171     See the manual for further details about \string\babelfont.\\%
4172     Reported}}
4173   {}}%
4174 \gdef\bbl@switchfont{%
4175   \bbl@ifunset{\bbl@lsys@\language\name}{\bbl@provide@lsys{\language\name}}}%
4176   \bbl@exp{% eg Arabic -> arabic
4177     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4178   \bbl@foreach\bbl@font@fams{%
4179     \bbl@ifunset{\bbl@##1dflt@\language\name}%      (1) language?
4180     {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}%      (2) from script?
4181       {\bbl@ifunset{\bbl@##1dflt@}%                2=F - (3) from generic?
4182         {}}%                                         123=F - nothing!
4183         {\bbl@exp{%                                3=T - from generic
4184           \global\let\<bbl@##1dflt@\language\name>%
4185             \<bbl@##1dflt@>}}}%
4186         {\bbl@exp{%                                2=T - from script
4187           \global\let\<bbl@##1dflt@\language\name>%
4188             \<bbl@##1dflt@*\bbl@tempa>}}}}}%
4189     {}}%                                           1=T - language, already defined
4190   \def\bbl@tempa{\bbl@nostdfont{}}}%
4191   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4192     \bbl@ifunset{\bbl@##1dflt@\language\name}%
4193     {\bbl@cs{famrst@##1}%
4194       \global\bbl@csarg\let{famrst@##1}\relax}%
4195     {\bbl@exp{% order is relevant
4196       \\bbl@add\\originalTeX{%
4197         \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4198         \<##1default>\<##1family>{##1}}}%
4199       \\bbl@font@set\<bbl@##1dflt@\language\name>% the main part!
4200       \<##1default>\<##1family>}}}}}%
4201   \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.

```

4202 \ifx\f@family\undefined\else % if latex
4203   \ifcase\bbl@engine % if pdftex
4204     \let\bbl@ckeckstdfonts\relax
4205   \else
4206     \def\bbl@ckeckstdfonts{%
4207       \begingroup
4208       \global\let\bbl@ckeckstdfonts\relax
4209       \let\bbl@tempa\@empty

```



```

4210 \bbl@foreach\bbl@font@fams{%
4211 \bbl@ifunset\bbl@##1dflt@}%
4212 {\@nameuse{##1family}}%
4213 \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4214 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \f@family\\}%
4215 \space\space\fontname\font\\}%
4216 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4217 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4218 }%
4219 \ifx\bbl@tempa\@empty\else
4220 \bbl@infowarn{The following font families will use the default\\%
4221 settings for all or some languages:\\%
4222 \bbl@tempa
4223 There is nothing intrinsically wrong with it, but\\%
4224 'babel' will no set Script and Language, which could\\%
4225 be relevant in some languages. If your document uses\\%
4226 these families, consider redefining them with \string\babelfont.\\%
4227 Reported}%
4228 \fi
4229 \endgroup}
4230 \fi
4231 \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.

```

4232 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4233 \bbl@xin@{<>}{#1}%
4234 \ifin@
4235 \bbl@exp{\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4236 \fi
4237 \bbl@exp{%
4238 \def\\#2#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4239 \\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\bbl@tempa\relax}}}%
4240 % TODO - next should be global?, but even local does its job. I'm
4241 % still not sure -- must investigate:
4242 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4243 \let\bbl@tempe\bbl@mapselect
4244 \let\bbl@mapselect\relax
4245 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4246 \let#4\@empty % Make sure \renewfontfamily is valid
4247 \bbl@exp{%
4248 \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4249 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4250 {\newfontscript\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4251 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4252 {\newfontlanguage\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4253 \\renewfontfamily\\#4%
4254 [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{.}{#3}
4255 \begingroup
4256 #4%
4257 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4258 \endgroup
4259 \let#4\bbl@temp@fam
4260 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4261 \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.

```
4262 \def\bbbl@font@rst#1#2#3#4{%
4263   \bbbl@csarg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}
```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```
4264 \def\bbbl@font@fams{rm,sf,tt}
```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```
4265 \newcommand\babelFSstore[2][{%
4266   \bbbl@ifblank{#1}%
4267   {\bbbl@csarg\def{sname@#2}{Latin}}%
4268   {\bbbl@csarg\def{sname@#2}{#1}}%
4269   \bbbl@provide@dirs{#2}%
4270   \bbbl@csarg\ifnum{wdir@#2}>\z@
4271     \let\bbbl@beforeforeign\leavevmode
4272     \EnableBabelHook{babel-bidi}%
4273   \fi
4274   \bbbl@foreach{#2}{%
4275     \bbbl@FSstore{##1}{rm}\rmdefault\bbbl@save@rmdefault
4276     \bbbl@FSstore{##1}{sf}\sfdefault\bbbl@save@sfdefault
4277     \bbbl@FSstore{##1}{tt}\ttdefault\bbbl@save@ttdefault}}
4278 \def\bbbl@FSstore#1#2#3#4{%
4279   \bbbl@csarg\edef{#2default#1}{#3}%
4280   \expandafter\addto\csname extras#1\endcsname{%
4281     \let#4#3%
4282     \ifx#3\f@family
4283       \edef#3{\csname bbl@#2default#1\endcsname}%
4284       \fontfamily{#3}\selectfont
4285     \else
4286       \edef#3{\csname bbl@#2default#1\endcsname}%
4287       \fi}%
4288   \expandafter\addto\csname noextras#1\endcsname{%
4289     \ifx#3\f@family
4290       \fontfamily{#4}\selectfont
4291     \fi
4292     \let#3#4}}
4293 \let\bbbl@langfeatures\@empty
4294 \def\babelFSfeatures{% make sure \fontspec is redefined once
4295   \let\bbbl@ori@fontspec\fontspec
4296   \renewcommand\fontspec[1][{%
4297     \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
4298   \let\babelFSfeatures\bbbl@FSfeatures
4299   \babelFSfeatures}
4300 \def\bbbl@FSfeatures#1#2{%
4301   \expandafter\addto\csname extras#1\endcsname{%
4302     \babel@save\bbbl@langfeatures
4303     \edef\bbbl@langfeatures{#2,}}
4304 <</Font selection>>
```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

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

```
4305 <<{*Footnote changes}>> ≡
4306 \bbl@trace{Bidi footnotes}
4307 \ifnum\bbl@bidimode>\z@
4308   \def\bbl@footnote#1#2#3{%
4309     \ifnextchar[%
4310       {\bbl@footnote@o{#1}{#2}{#3}}%
4311       {\bbl@footnote@x{#1}{#2}{#3}}}
4312   \def\bbl@footnote@x#1#2#3#4{%
4313     \bgroup
4314     \select@language@x{\bbl@main@language}%
4315     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4316     \egroup}
4317   \def\bbl@footnote@o#1#2#3[#4]#5{%
4318     \bgroup
4319     \select@language@x{\bbl@main@language}%
4320     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4321     \egroup}
4322   \def\bbl@footnotetext#1#2#3{%
4323     \ifnextchar[%
4324       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4325       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4326   \def\bbl@footnotetext@x#1#2#3#4{%
4327     \bgroup
4328     \select@language@x{\bbl@main@language}%
4329     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4330     \egroup}
4331   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4332     \bgroup
4333     \select@language@x{\bbl@main@language}%
4334     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4335     \egroup}
4336   \def\BabelFootnote#1#2#3#4{%
4337     \ifx\bbl@fn@footnote\@undefined
4338       \let\bbl@fn@footnote\footnote
4339     \fi
4340     \ifx\bbl@fn@footnotetext\@undefined
4341       \let\bbl@fn@footnotetext\footnotetext
4342     \fi
4343     \bbl@ifblank{#2}%
4344     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4345      \@namedef{\bbl@stripslash#1text}%
4346      {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4347     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4348      \@namedef{\bbl@stripslash#1text}%
4349      {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4350   \fi
4351 <</Footnote changes>>
```

Now, the code.

```
4352 <{*xetex}>
4353 \def\BabelStringsDefault{unicode}
4354 \let\xebbl@stop\relax
```

```

4355 \AddBabelHook{xetex}{encodedcommands}{%
4356   \def\bbl@tempa{#1}%
4357   \ifx\bbl@tempa\@empty
4358     \XeTeXinputencoding"bytes"%
4359   \else
4360     \XeTeXinputencoding"#1"%
4361   \fi
4362   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4363 \AddBabelHook{xetex}{stopcommands}{%
4364   \xebbl@stop
4365   \let\xebbl@stop\relax}
4366 \def\bbl@intraspace#1 #2 #3@@{%
4367   \bbl@csarg\gdef{\xeisp@language}%
4368   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4369 \def\bbl@intrapenalty#1@@{%
4370   \bbl@csarg\gdef{\xeipn@language}%
4371   {\XeTeXlinebreakpenalty #1\relax}}
4372 \def\bbl@provide@intraspace{%
4373   \bbl@xin@{\bbl@cl{\lnbrk}}{s}%
4374   \ifin@else\bbl@xin@{\bbl@cl{\lnbrk}}{c}\fi
4375   \ifin@
4376     \bbl@ifunset{\bbl@intsp@language}{}%
4377     {\expandafter\ifx\csname\bbl@intsp@language\endcsname\@empty\else
4378       \ifx\bbl@KVP@intraspace\@nil
4379         \bbl@exp{%
4380           \\bbl@intraspace\bbl@cl{intsp}\\@@}%
4381         \fi
4382         \ifx\bbl@KVP@intrapenalty\@nil
4383           \bbl@intrapenalty0@@
4384         \fi
4385       \fi
4386       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4387         \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4388       \fi
4389       \ifx\bbl@KVP@intrapenalty\@nil\else
4390         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4391       \fi
4392       \bbl@exp{%
4393         \\bbl@add\<extras\language>{%
4394           \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
4395           \<bbl@xeisp@language>%
4396           \<bbl@xeipn@language>%
4397           \\bbl@toglobal\<extras\language>%
4398           \\bbl@add\<noextras\language>{%
4399             \XeTeXlinebreaklocale "en"%
4400             \\bbl@toglobal\<noextras\language>}}%
4401       \ifx\bbl@ispace\@undefined
4402         \gdef\bbl@ispace{\bbl@cl{\xeisp}}%
4403       \ifx\AtBeginDocument\@notprerr
4404         \expandafter\@secondoftwo % to execute right now
4405       \fi
4406       \AtBeginDocument{%
4407         \expandafter\bbl@add
4408         \csname selectfont \endcsname{\bbl@ispace}%
4409         \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4410       \fi}%
4411   \fi}
4412 \ifx\DisableBabelHook\@undefined\endinput\fi
4413 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}

```

```

4414 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4415 \DisableBabelHook{babel-fontspec}
4416 <<Font selection>>
4417 \input txtbabel.def
4418 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titles, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the 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.

```

4419 (*texxet)
4420 \providecommand\bbl@provide@intraspace{}
4421 \bbl@trace{Redefinitions for bidi layout}
4422 \def\bbl@sspre@caption{%
4423   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}%
4424   \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4425 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4426 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4427 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4428   \def\hangfrom#1{%
4429     \setbox\@tempboxa\hbox{#1}%
4430     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4431     \noindent\box\@tempboxa}
4432 \def\raggedright{%
4433   \let\@centercr
4434   \bbl@startskip\z@skip
4435   \@rightskip\@flushglue
4436   \bbl@endskip\@rightskip
4437   \parindent\z@
4438   \parfillskip\bbl@startskip}
4439 \def\raggedleft{%
4440   \let\@centercr
4441   \bbl@startskip\@flushglue
4442   \bbl@endskip\z@skip
4443   \parindent\z@
4444   \parfillskip\bbl@endskip}
4445 \fi
4446 \IfBabelLayout{lists}
4447   {\bbl@sreplace\list
4448     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4449     \def\bbl@listleftmargin{%
4450       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4451     \ifcase\bbl@engine
4452       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4453       \def\p@enumiii{\p@enumii}\theenumii{}
4454     \fi
4455     \bbl@sreplace\@verbatim
4456       {\leftskip\@totalleftmargin}%
4457       {\bbl@startskip\textwidth
4458         \advance\bbl@startskip-\linewidth}%
4459     \bbl@sreplace\@verbatim

```

```

4460      {\rightskip\z@skip}%
4461      {\bbl@endskip\z@skip}}%
4462  {}
4463  \IfBabelLayout{contents}%
4464  {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4465   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4466  {}
4467  \IfBabelLayout{columns}%
4468  {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4469   \def\bbl@outputbox#1{%
4470     \hb@xt@\textwidth{%
4471       \hskip\columnwidth
4472       \hfil
4473       {\normalcolor\vrule \@width\columnseprule}%
4474       \hfil
4475       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4476       \hskip-\textwidth
4477       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4478       \hskip\columnsep
4479       \hskip\columnwidth}}}%
4480  {}
4481  <<Footnote changes>>
4482  \IfBabelLayout{footnotes}%
4483  {\BabelFootnote\footnote\language\{}}%
4484  \BabelFootnote\localfootnote\language\{}}%
4485  \BabelFootnote\mainfootnote\{}}%
4486  {}
4487  \IfBabelLayout{counters}%
4488  {\let\bbl@latinarabic=\@arabic
4489   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4490   \let\bbl@asciroman=\@roman
4491   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4492   \let\bbl@asciiRoman=\@Roman
4493   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
4494  </texet>

```

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.

13.3 LuaTeX

The loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4495 (*luatex)
4496 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4497 \bbl@trace{Read language.dat}
4498 \ifx\bbl@readstream\undefined
4499   \csname newread\endcsname\bbl@readstream
4500 \fi
4501 \begingroup
4502   \toks@{}
4503   \count@ \z@ % 0=start, 1=0th, 2=normal
4504   \def\bbl@process@line#1#2 #3 #4 {%
4505     \ifx=#1%
4506       \bbl@process@synonym{#2}%
4507     \else
4508       \bbl@process@language{#1#2}{#3}{#4}%
4509     \fi
4510     \ignorespaces}
4511   \def\bbl@manylang{%
4512     \ifnum\bbl@last>\@ne
4513       \bbl@info{Non-standard hyphenation setup}%
4514     \fi
4515     \let\bbl@manylang\relax}
4516   \def\bbl@process@language#1#2#3{%
4517     \ifcase\count@
4518       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4519     \or
4520       \count@\tw@
4521     \fi
4522     \ifnum\count@=\tw@
4523       \expandafter\addlanguage\csname l@#1\endcsname
4524       \language\allocationnumber
4525       \chardef\bbl@last\allocationnumber
4526       \bbl@manylang
4527       \let\bbl@elt\relax
4528       \xdef\bbl@languages{%
4529         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4530     \fi
4531     \the\toks@
4532     \toks@{}}

```

```

4533 \def\bbl@process@synonym@aux#1#2{%
4534   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4535   \let\bbl@elt\relax
4536   \xdef\bbl@languages{%
4537     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4538 \def\bbl@process@synonym#1{%
4539   \ifcase\count@
4540     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4541   \or
4542     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4543   \else
4544     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4545   \fi}
4546 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4547   \chardef\l@english\z@
4548   \chardef\l@USenglish\z@
4549   \chardef\bbl@last\z@
4550   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4551   \gdef\bbl@languages{%
4552     \bbl@elt{english}{0}{hyphen.tex}}%
4553     \bbl@elt{USenglish}{0}{}}
4554 \else
4555   \global\let\bbl@languages@format\bbl@languages
4556   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4557     \ifnum#2>\z@\else
4558       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4559     \fi}%
4560   \xdef\bbl@languages{\bbl@languages}%
4561 \fi
4562 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4563 \bbl@languages
4564 \openin\bbl@readstream=language.dat
4565 \ifeof\bbl@readstream
4566   \bbl@warning{I couldn't find language.dat. No additional\\%
4567     patterns loaded. Reported}%
4568 \else
4569   \loop
4570     \endlinechar\m@ne
4571     \read\bbl@readstream to \bbl@line
4572     \endlinechar\^^M
4573     \if T\ifeof\bbl@readstream F\fi T\relax
4574     \ifx\bbl@line\@empty\else
4575       \edef\bbl@line{\bbl@line\space\space\space}%
4576       \expandafter\bbl@process@line\bbl@line\relax
4577     \fi
4578   \repeat
4579 \fi
4580 \endgroup
4581 \bbl@trace{Macros for reading patterns files}
4582 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4583 \ifx\babelcatcodetablenum\@undefined
4584   \ifx\newcatcodetable\@undefined
4585     \def\babelcatcodetablenum{5211}
4586     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4587   \else
4588     \newcatcodetable\babelcatcodetablenum
4589     \newcatcodetable\bbl@pattcodes
4590   \fi
4591 \else

```



```

4592 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4593 \fi
4594 \def\bbl@luapatterns#1#2{%
4595   \bbl@get@enc#1::\@@@
4596   \setbox\z@\hbox\bgroup
4597     \begingroup
4598       \savecatcodetable\babelcatcodetablenum\relax
4599       \initcatcodetable\bbl@pattcodes\relax
4600       \catcodetable\bbl@pattcodes\relax
4601       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4602       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4603       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4604       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4605       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4606       \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4607       \input #1\relax
4608       \catcodetable\babelcatcodetablenum\relax
4609     \endgroup
4610   \def\bbl@tempa{#2}%
4611   \ifx\bbl@tempa@empty\else
4612     \input #2\relax
4613   \fi
4614 \egroup}%
4615 \def\bbl@patterns@lua#1{%
4616   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4617     \csname l@#1\endcsname
4618     \edef\bbl@tempa{#1}%
4619   \else
4620     \csname l@#1:\f@encoding\endcsname
4621     \edef\bbl@tempa{#1:\f@encoding}%
4622   \fi\relax
4623   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4624   \@ifundefined{bbl@hyphendata@the\language}%
4625   {\def\bbl@elt##1##2##3##4{%
4626     \ifnum##2=\csname l@#1:\f@encoding\endcsname % #2=spanish, dutch:OT1...
4627     \def\bbl@tempb{##3}%
4628     \ifx\bbl@tempb@empty\else % if not a synonymous
4629       \def\bbl@tempc{##3}{##4}%
4630     \fi
4631     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4632   \fi}%
4633   \bbl@languages
4634   \@ifundefined{bbl@hyphendata@the\language}%
4635   {\bbl@info{No hyphenation patterns were set for\%
4636     language '\bbl@tempa'. Reported}}%
4637   {\expandafter\expandafter\expandafter\bbl@luapatterns
4638     \csname bbl@hyphendata@the\language\endcsname}}}%
4639 \endinput\fi
4640 % Here ends \ifx\AddBabelHook\undefined
4641 % A few lines are only read by hyphen.cfg
4642 \ifx\DisableBabelHook\undefined
4643   \AddBabelHook{luatex}{everylanguage}{%
4644     \def\process@language##1##2##3{%
4645       \def\process@line####1####2 ####3 ####4 {}}%
4646   \AddBabelHook{luatex}{loadpatterns}{%
4647     \input #1\relax
4648     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4649       {##1}{}}%
4650   \AddBabelHook{luatex}{loadexceptions}{%

```

```

4651 \input #1\relax
4652 \def\bbl@tempb##1##2{{##1}{##2}}%
4653 \expandafter\edef\csname bbl@hyphendata@the\language\endcsname
4654 {\expandafter\expandafter\expandafter\bbl@tempb
4655 \csname bbl@hyphendata@the\language\endcsname}}
4656 \endinput\fi
4657 % Here stops reading code for hyphen.cfg
4658 % The following is read the 2nd time it's loaded
4659 \begingroup
4660 \catcode`\%=12
4661 \catcode`\'=12
4662 \catcode`\%=12
4663 \catcode`\:=12
4664 \directlua{
4665   Babel = Babel or {}
4666   function Babel.bytes(line)
4667     return line:gsub(".",
4668       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4669   end
4670   function Babel.begin_process_input()
4671     if luatexbase and luatexbase.add_to_callback then
4672       luatexbase.add_to_callback('process_input_buffer',
4673         Babel.bytes, 'Babel.bytes')
4674     else
4675       Babel.callback = callback.find('process_input_buffer')
4676       callback.register('process_input_buffer', Babel.bytes)
4677     end
4678   end
4679   function Babel.end_process_input ()
4680     if luatexbase and luatexbase.remove_from_callback then
4681       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4682     else
4683       callback.register('process_input_buffer', Babel.callback)
4684     end
4685   end
4686   function Babel.addpatterns(pp, lg)
4687     local lg = lang.new(lg)
4688     local pats = lang.patterns(lg) or ''
4689     lang.clear_patterns(lg)
4690     for p in pp:gmatch('[^%s]+') do
4691       ss = ''
4692       for i in string.utfcharacters(p:gsub('%d', '')) do
4693         ss = ss .. '%d?' .. i
4694       end
4695       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4696       ss = ss:gsub('%.%%d%?$', '%%.')
4697       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4698       if n == 0 then
4699         tex.sprint(
4700           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4701           .. p .. [[]])
4702         pats = pats .. ' ' .. p
4703       else
4704         tex.sprint(
4705           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4706           .. p .. [[]])
4707       end
4708     end
4709     lang.patterns(lg, pats)

```

```

4710 end
4711 }
4712 \endgroup
4713 \ifx\newattribute\undefined\else
4714 \newattribute\bbbl@attr@locale
4715 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbbl@attr@locale'}
4716 \AddBabelHook{luatex}{beforeextras}{%
4717 \setattribute\bbbl@attr@locale\localeid}
4718 \fi
4719 \def\BabelStringsDefault{unicode}
4720 \let\luabbbl@stop\relax
4721 \AddBabelHook{luatex}{encodedcommands}{%
4722 \def\bbbl@tempa{utf8}\def\bbbl@tempb{#1}%
4723 \ifx\bbbl@tempa\bbbl@tempb\else
4724 \directlua{Babel.begin_process_input()}%
4725 \def\luabbbl@stop{%
4726 \directlua{Babel.end_process_input()}}%
4727 \fi}%
4728 \AddBabelHook{luatex}{stopcommands}{%
4729 \luabbbl@stop
4730 \let\luabbbl@stop\relax}
4731 \AddBabelHook{luatex}{patterns}{%
4732 \@ifundefined{bbbl@hyphendata@the\language}%
4733 {\def\bbbl@elt##1##2##3##4{%
4734 \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
4735 \def\bbbl@tempb{##3}%
4736 \ifx\bbbl@tempb\empty\else % if not a synonymous
4737 \def\bbbl@tempc{##3}{##4}%
4738 \fi
4739 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
4740 \fi}%
4741 \bbbl@languages
4742 \@ifundefined{bbbl@hyphendata@the\language}%
4743 {\bbbl@info{No hyphenation patterns were set for\%
4744 language '#2'. Reported}}%
4745 {\expandafter\expandafter\expandafter\bbbl@luapatterns
4746 \csname bbl@hyphendata@the\language\endcsname}}}%
4747 \@ifundefined{bbbl@patterns@}{}%
4748 \begingroup
4749 \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
4750 \ifin\else
4751 \ifx\bbbl@patterns@\empty\else
4752 \directlua{ Babel.addpatterns(
4753 [[\bbbl@patterns@]], \number\language) }%
4754 \fi
4755 \@ifundefined{bbbl@patterns@#1}%
4756 \empty
4757 {\directlua{ Babel.addpatterns(
4758 [[\space\csname bbl@patterns@#1\endcsname]],
4759 \number\language) }}%
4760 \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
4761 \fi
4762 \endgroup}%
4763 \bbbl@exp{%
4764 \bbbl@ifunset{bbbl@prehc\languagename}{}%
4765 {\bbbl@ifblank{\bbbl@cs{prehc\languagename}}}%
4766 {\prehyphenchar=\bbbl@cl{prehc}\relax}}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbbl@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.

```

4767 \@onlypreamble\babelpatterns
4768 \AtEndOfPackage{%
4769   \newcommand\babelpatterns[2][\@empty]{%
4770     \ifx\bbl@patterns@relax
4771       \let\bbl@patterns@\@empty
4772     \fi
4773     \ifx\bbl@pttnlist\@empty\else
4774       \bbl@warning{%
4775         You must not intermingle \string\selectlanguage\space and\%
4776         \string\babelpatterns\space or some patterns will not\%
4777         be taken into account. Reported}%
4778     \fi
4779     \ifx\@empty#1%
4780       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4781     \else
4782       \edef\bbl@tempb{\zap@space#1 \@empty}%
4783       \bbl@for\bbl@tempa\bbl@tempb{%
4784         \bbl@fixname\bbl@tempa
4785         \bbl@iflanguage\bbl@tempa{%
4786           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4787             \ifundefined{bbl@patterns@\bbl@tempa}%
4788               \@empty
4789               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4790             #2}}}%
4791     \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

In progress. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4792 \directlua{
4793   Babel = Babel or {}
4794   Babel.linebreaking = Babel.linebreaking or {}
4795   Babel.linebreaking.before = {}
4796   Babel.linebreaking.after = {}
4797   Babel.locale = {} % Free to use, indexed with \localeid
4798   function Babel.linebreaking.add_before(func)
4799     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4800     table.insert(Babel.linebreaking.before , func)
4801   end
4802   function Babel.linebreaking.add_after(func)
4803     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4804     table.insert(Babel.linebreaking.after, func)
4805   end
4806 }
4807 \def\bbl@intraspace#1 #2 #3\@@{%
4808   \directlua{
4809     Babel = Babel or {}
4810     Babel.intraspaces = Babel.intraspaces or {}
4811     Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
4812       {b = #1, p = #2, m = #3}
4813     Babel.locale_props[\the\localeid].intraspace = %

```

```

4814         {b = #1, p = #2, m = #3}
4815     }}
4816 \def\bbl@intrapenalty#1\@@{%
4817     \directlua{
4818         Babel = Babel or {}
4819         Babel.intrapenalties = Babel.intrapenalties or {}
4820         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4821         Babel.locale_props[\the\localeid].intrapenalty = #1
4822     }}
4823 \begingroup
4824 \catcode`\%=12
4825 \catcode`\^=14
4826 \catcode`\'=12
4827 \catcode`\~=12
4828 \gdef\bbl@seaintraspace{^
4829     \let\bbl@seaintraspace\relax
4830     \directlua{
4831         Babel = Babel or {}
4832         Babel.sea_enabled = true
4833         Babel.sea_ranges = Babel.sea_ranges or {}
4834         function Babel.set_chranges (script, chrng)
4835             local c = 0
4836             for s, e in string.gmatch(chrng..' ', '(-)%%.(-)%s') do
4837                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4838                 c = c + 1
4839             end
4840         end
4841         function Babel.sea_disc_to_space (head)
4842             local sea_ranges = Babel.sea_ranges
4843             local last_char = nil
4844             local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4845             for item in node.traverse(head) do
4846                 local i = item.id
4847                 if i == node.id'glyph' then
4848                     last_char = item
4849                 elseif i == 7 and item.subtype == 3 and last_char
4850                     and last_char.char > 0x0C99 then
4851                     quad = font.getfont(last_char.font).size
4852                     for lg, rg in pairs(sea_ranges) do
4853                         if last_char.char > rg[1] and last_char.char < rg[2] then
4854                             lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyl1
4855                             local intraspace = Babel.intraspaces[lg]
4856                             local intrapenalty = Babel.intrapenalties[lg]
4857                             local n
4858                             if intrapenalty ~= 0 then
4859                                 n = node.new(14, 0) ^^ penalty
4860                                 n.penalty = intrapenalty
4861                                 node.insert_before(head, item, n)
4862                             end
4863                             n = node.new(12, 13) ^^ (glue, spaceskip)
4864                             node.setglue(n, intraspace.b * quad,
4865                                 intraspace.p * quad,
4866                                 intraspace.m * quad)
4867                             node.insert_before(head, item, n)
4868                             node.remove(head, item)
4869                         end
4870                     end
4871                 end
4872             end

```

```

4873     end
4874 }^^
4875 \bbl@luahyphenate}
4876 \catcode`\%=14
4877 \gdef\bbl@cjkintraspacespace{%
4878   \let\bbl@cjkintraspacespace\relax
4879   \directlua{
4880     Babel = Babel or {}
4881     require'babel-data-cjk.lua'
4882     Babel.cjk_enabled = true
4883     function Babel.cjk_linebreak(head)
4884       local GLYPH = node.id'glyph'
4885       local last_char = nil
4886       local quad = 655360      % 10 pt = 655360 = 10 * 65536
4887       local last_class = nil
4888       local last_lang = nil
4889
4890       for item in node.traverse(head) do
4891         if item.id == GLYPH then
4892
4893           local lang = item.lang
4894
4895           local LOCALE = node.get_attribute(item,
4896             luatexbase.registernumber'bbl@attr@locale')
4897           local props = Babel.locale_props[LOCALE]
4898
4899           local class = Babel.cjk_class[item.char].c
4900
4901           if class == 'cp' then class = 'cl' end % ]) as CL
4902           if class == 'id' then class = 'I' end
4903
4904           local br = 0
4905           if class and last_class and Babel.cjk_breaks[last_class][class] then
4906             br = Babel.cjk_breaks[last_class][class]
4907           end
4908
4909           if br == 1 and props.linebreak == 'c' and
4910             lang ~= \the\l@nohyphenation\space and
4911             last_lang ~= \the\l@nohyphenation then
4912             local intrapenalty = props.intrapenalty
4913             if intrapenalty ~= 0 then
4914               local n = node.new(14, 0)      % penalty
4915               n.penalty = intrapenalty
4916               node.insert_before(head, item, n)
4917             end
4918             local intraspacespace = props.intraspacespace
4919             local n = node.new(12, 13)      % (glue, spaceskip)
4920             node.setglue(n, intraspacespace.b * quad,
4921               intraspacespace.p * quad,
4922               intraspacespace.m * quad)
4923             node.insert_before(head, item, n)
4924           end
4925
4926           quad = font.getfont(item.font).size
4927           last_class = class
4928           last_lang = lang
4929         else % if penalty, glue or anything else
4930           last_class = nil
4931         end

```

```

4932     end
4933     lang.hyphenate(head)
4934 end
4935 }%
4936 \bbl@luahyphenate}
4937 \gdef\bbl@luahyphenate{%
4938 \let\bbl@luahyphenate\relax
4939 \directlua{
4940   luatexbase.add_to_callback('hyphenate',
4941   function (head, tail)
4942     if Babel.linebreaking.before then
4943       for k, func in ipairs(Babel.linebreaking.before) do
4944         func(head)
4945       end
4946     end
4947     if Babel.cjk_enabled then
4948       Babel.cjk_linebreak(head)
4949     end
4950     lang.hyphenate(head)
4951     if Babel.linebreaking.after then
4952       for k, func in ipairs(Babel.linebreaking.after) do
4953         func(head)
4954       end
4955     end
4956     if Babel.sea_enabled then
4957       Babel.sea_disc_to_space(head)
4958     end
4959   end,
4960   'Babel.hyphenate')
4961 }
4962 }
4963 \endgroup
4964 \def\bbl@provide@intraspace{%
4965 \bbl@ifunset{\bbl@intsp@{language}}{%
4966   {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
4967     \bbl@xin@{\bbl@cl{lbrk}}{c}%
4968     \ifin@           % cjk
4969     \bbl@cjk_intraspace
4970     \directlua{
4971       Babel = Babel or {}
4972       Babel.locale_props = Babel.locale_props or {}
4973       Babel.locale_props[\the\localeid].linebreak = 'c'
4974     }%
4975     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4976     \ifx\bbl@KVP@intrapenalty\@nil
4977       \bbl@intrapenalty0\@@
4978     \fi
4979   \else           % sea
4980     \bbl@sea_intraspace
4981     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4982     \directlua{
4983       Babel = Babel or {}
4984       Babel.sea_ranges = Babel.sea_ranges or {}
4985       Babel.set_chranges('\bbl@cl{sbc} ',
4986         '\bbl@cl{chrng}')
4987     }%
4988     \ifx\bbl@KVP@intrapenalty\@nil
4989       \bbl@intrapenalty0\@@
4990     \fi

```

```

4991      \fi
4992      \fi
4993      \ifx\bbbl@KVP@intrapenalty\@nil\else
4994      \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
4995      \fi}}

```

13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used.

There is a separate file, defined below.

Work in progress.

Common stuff.

```

4996 \AddBabelHook{babel-fontspec}{afterextras}{\bbbl@switchfont}
4997 \AddBabelHook{babel-fontspec}{beforestart}{\bbbl@ckeckstdfonts}
4998 \DisableBabelHook{babel-fontspec}
4999 <<Font selection>>

```

13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5000 \directlua{
5001 Babel.script_blocks = {
5002   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5003             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5004   ['Armn'] = {{0x0530, 0x058F}},
5005   ['Beng'] = {{0x0980, 0x09FF}},
5006   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5007   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5008   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5009             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5010   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5011   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5012             {0xAB00, 0xAB2F}},
5013   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5014   % Don't follow strictly Unicode, which places some Coptic letters in
5015   % the 'Greek and Coptic' block
5016   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5017   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5018             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5019             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5020             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5021             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5022             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5023   ['Hebr'] = {{0x0590, 0x05FF}},

```



```

5024 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5025             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5026 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5027 ['Knda'] = {{0x0C80, 0x0CFF}},
5028 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5029             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5030             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5031 ['Lao'] = {{0x0E80, 0x0EFF}},
5032 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5033             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5034             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5035 ['Mahj'] = {{0x11150, 0x1117F}},
5036 ['Mlym'] = {{0x0D00, 0x0D7F}},
5037 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5038 ['Orya'] = {{0x0B00, 0x0B7F}},
5039 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5040 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5041 ['Taml'] = {{0x0B80, 0x0BFF}},
5042 ['Telu'] = {{0x0C00, 0x0C7F}},
5043 ['Tfng'] = {{0x2D30, 0x2D7F}},
5044 ['Thai'] = {{0x0E00, 0x0E7F}},
5045 ['Tibt'] = {{0x0F00, 0x0FFF}},
5046 ['Vaii'] = {{0xA500, 0xA63F}},
5047 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5048 }
5049
5050 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5051 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5052 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5053
5054 function Babel.locale_map(head)
5055   if not Babel.locale_mapped then return head end
5056
5057   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5058   local GLYPH = node.id('glyph')
5059   local inmath = false
5060   local toloc_save
5061   for item in node.traverse(head) do
5062     local toloc
5063     if not inmath and item.id == GLYPH then
5064       % Optimization: build a table with the chars found
5065       if Babel.chr_to_loc[item.char] then
5066         toloc = Babel.chr_to_loc[item.char]
5067       else
5068         for lc, maps in pairs(Babel.loc_to_scr) do
5069           for _, rg in pairs(maps) do
5070             if item.char >= rg[1] and item.char <= rg[2] then
5071               Babel.chr_to_loc[item.char] = lc
5072               toloc = lc
5073               break
5074             end
5075           end
5076         end
5077       end
5078       % Now, take action, but treat composite chars in a different
5079       % fashion, because they 'inherit' the previous locale. Not yet
5080       % optimized.
5081       if not toloc and
5082         (item.char >= 0x0300 and item.char <= 0x036F) or

```

```

5083         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5084         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5085             toloc = toloc_save
5086         end
5087         if toloc and toloc > -1 then
5088             if Babel.locale_props[toloc].lg then
5089                 item.lang = Babel.locale_props[toloc].lg
5090                 node.set_attribute(item, LOCALE, toloc)
5091             end
5092             if Babel.locale_props[toloc]['/'..item.font] then
5093                 item.font = Babel.locale_props[toloc]['/'..item.font]
5094             end
5095             toloc_save = toloc
5096         end
5097         elseif not inmath and item.id == 7 then
5098             item.replace = item.replace and Babel.locale_map(item.replace)
5099             item.pre      = item.pre and Babel.locale_map(item.pre)
5100             item.post     = item.post and Babel.locale_map(item.post)
5101         elseif item.id == node.id'math' then
5102             inmath = (item.subtype == 0)
5103         end
5104     end
5105     return head
5106 end
5107 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

5108 \newcommand\babelcharproperty[1]{%
5109   \count@=#1\relax
5110   \ifvmode
5111     \expandafter\bbl@chprop
5112   \else
5113     \bbl@error{\string\babelcharproperty\space can be used only in\%
5114       vertical mode (preamble or between paragraphs)}%
5115     {See the manual for futher info}%
5116   \fi}
5117 \newcommand\bbl@chprop[3][\the\count@]{%
5118   \@tempcnta=#1\relax
5119   \bbl@ifunset{\bbl@chprop@#2}%
5120   {\bbl@error{No property named '#2'. Allowed values are\%
5121     direction (bc), mirror (bmg), and linebreak (lb)}%
5122     {See the manual for futher info}}%
5123   }%
5124   \loop
5125     \bbl@cs{chprop@#2}{#3}%
5126   \ifnum\count@<\@tempcnta
5127     \advance\count@\@ne
5128   \repeat}
5129 \def\bbl@chprop@direction#1{%
5130   \directlua{
5131     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5132     Babel.characters[\the\count@]['d'] = '#1'
5133   }}
5134 \let\bbl@chprop@bc\bbl@chprop@direction
5135 \def\bbl@chprop@mirror#1{%
5136   \directlua{
5137     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5138     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

5139 }}
5140 \let\bbl@chprop@bmg\bbl@chprop@mirror
5141 \def\bbl@chprop@linebreak#1{%
5142   \directlua{
5143     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5144     Babel.cjk_characters[\the\count@]['c'] = '#1'
5145   }}
5146 \let\bbl@chprop@lb\bbl@chprop@linebreak
5147 \def\bbl@chprop@locale#1{%
5148   \directlua{
5149     Babel.chr_to_loc = Babel.chr_to_loc or {}
5150     Babel.chr_to_loc[\the\count@] =
5151       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5152   }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

5153 \begingroup
5154 \catcode`\#=12
5155 \catcode`\%=12
5156 \catcode`\&=14
5157 \directlua{
5158   Babel.linebreaking.post_replacements = {}
5159   Babel.linebreaking.pre_replacements = {}
5160
5161   function Babel.str_to_nodes(fn, matches, base)
5162     local n, head, last
5163     if fn == nil then return nil end
5164     for s in string.utfvalues(fn(matches)) do
5165       if base.id == 7 then
5166         base = base.replace
5167       end
5168       n = node.copy(base)
5169       n.char = s
5170       if not head then
5171         head = n
5172       else
5173         last.next = n
5174       end
5175       last = n
5176     end
5177     return head
5178   end
5179
5180   function Babel.fetch_word(head, funct)
5181     local word_string = ''
5182     local word_nodes = {}

```

```

5183     local lang
5184     local item = head
5185     local inmath = false
5186
5187     while item do
5188
5189         if item.id == 29
5190             and not(item.char == 124) && ie, not |
5191             and not(item.char == 61) && ie, not =
5192             and not inmath
5193             and (item.lang == lang or lang == nil) then
5194             lang = lang or item.lang
5195             word_string = word_string .. unicode.utf8.char(item.char)
5196             word_nodes[#word_nodes+1] = item
5197
5198         elseif item.id == 7 and item.subtype == 2 and not inmath then
5199             word_string = word_string .. '='
5200             word_nodes[#word_nodes+1] = item
5201
5202         elseif item.id == 7 and item.subtype == 3 and not inmath then
5203             word_string = word_string .. '|'
5204             word_nodes[#word_nodes+1] = item
5205
5206         elseif item.id == 11 and item.subtype == 0 then
5207             inmath = true
5208
5209         elseif word_string == '' then
5210             && pass
5211
5212         else
5213             return word_string, word_nodes, item, lang
5214         end
5215
5216         item = item.next
5217     end
5218 end
5219
5220 function Babel.post_hyphenate_replace(head)
5221     local u = unicode.utf8
5222     local lbkr = Babel.linebreaking.post_replacements
5223     local word_head = head
5224
5225     while true do
5226         local w, wn, nw, lang = Babel.fetch_word(word_head)
5227         if not lang then return head end
5228
5229         if not lbkr[lang] then
5230             break
5231         end
5232
5233         for k=1, #lbkr[lang] do
5234             local p = lbkr[lang][k].pattern
5235             local r = lbkr[lang][k].replace
5236
5237             while true do
5238                 local matches = { u.match(w, p) }
5239                 if #matches < 2 then break end
5240
5241                 local first = table.remove(matches, 1)

```

```

5242     local last = table.remove(matches, #matches)
5243
5244     %% Fix offsets, from bytes to unicode.
5245     first = u.len(w:sub(1, first-1)) + 1
5246     last = u.len(w:sub(1, last-1))
5247
5248     local new %% used when inserting and removing nodes
5249     local changed = 0
5250
5251     %% This loop traverses the replace list and takes the
5252     %% corresponding actions
5253     for q = first, last do
5254         local crep = r[q-first+1]
5255         local char_node = wn[q]
5256         local char_base = char_node
5257
5258         if crep and crep.data then
5259             char_base = wn[crep.data+first-1]
5260         end
5261
5262         if crep == {} then
5263             break
5264         elseif crep == nil then
5265             changed = changed + 1
5266             node.remove(head, char_node)
5267         elseif crep and (crep.pre or crep.no or crep.post) then
5268             changed = changed + 1
5269             d = node.new(7, 0) %% (disc, discretionary)
5270             d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5271             d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5272             d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5273             d.attr = char_base.attr
5274             if crep.pre == nil then %% TeXbook p96
5275                 d.penalty = crep.penalty or tex.hyphenpenalty
5276             else
5277                 d.penalty = crep.penalty or tex.exhyphenpenalty
5278             end
5279             head, new = node.insert_before(head, char_node, d)
5280             node.remove(head, char_node)
5281             if q == 1 then
5282                 word_head = new
5283             end
5284         elseif crep and crep.string then
5285             changed = changed + 1
5286             local str = crep.string(matches)
5287             if str == '' then
5288                 if q == 1 then
5289                     word_head = char_node.next
5290                 end
5291                 head, new = node.remove(head, char_node)
5292             elseif char_node.id == 29 and u.len(str) == 1 then
5293                 char_node.char = string.utfvalue(str)
5294             else
5295                 local n
5296                 for s in string.utfvalues(str) do
5297                     if char_node.id == 7 then
5298                         log('Automatic hyphens cannot be replaced, just removed.')
5299                     else
5300                         n = node.copy(char_base)

```

```

5301         end
5302         n.char = s
5303         if q == 1 then
5304             head, new = node.insert_before(head, char_node, n)
5305             word_head = new
5306         else
5307             node.insert_before(head, char_node, n)
5308         end
5309     end
5310
5311     node.remove(head, char_node)
5312     end %% string length
5313     end %% if char and char.string
5314     end %% for char in match
5315     if changed > 20 then
5316         texio.write('Too many changes. Ignoring the rest.')
5317     elseif changed > 0 then
5318         w, wn, nw = Babel.fetch_word(word_head)
5319     end
5320
5321     end %% for match
5322     end %% for patterns
5323     word_head = nw
5324     end %% for words
5325     return head
5326 end
5327
5328 %%%
5329 %% Preliminary code for \babelprehyphenation
5330 %% TODO. Copypaste pattern. Merge with fetch_word
5331 function Babel.fetch_subtext(head, funct)
5332     local word_string = ''
5333     local word_nodes = {}
5334     local lang
5335     local item = head
5336     local inmath = false
5337
5338     while item do
5339
5340         if item.id == 29 then
5341             local locale = node.get_attribute(item, Babel.attr_locale)
5342
5343             if not(item.char == 124) %% ie, not | = space
5344                 and not inmath
5345                 and (locale == lang or lang == nil) then
5346                 lang = lang or locale
5347                 word_string = word_string .. unicode.utf8.char(item.char)
5348                 word_nodes[#word_nodes+1] = item
5349             end
5350
5351             if item == node.tail(head) then
5352                 item = nil
5353                 return word_string, word_nodes, item, lang
5354             end
5355
5356             elseif item.id == 12 and item.subtype == 13 and not inmath then
5357                 word_string = word_string .. '|'
5358                 word_nodes[#word_nodes+1] = item
5359

```

```

5360         if item == node.tail(head) then
5361             item = nil
5362             return word_string, word_nodes, item, lang
5363         end
5364
5365         elseif item.id == 11 and item.subtype == 0 then
5366             inmath = true
5367
5368         elseif word_string == '' then
5369             %% pass
5370
5371         else
5372             return word_string, word_nodes, item, lang
5373         end
5374
5375         item = item.next
5376     end
5377 end
5378
5379 &% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5380 function Babel.pre_hyphenate_replace(head)
5381     local u = unicode.utf8
5382     local lbkr = Babel.linebreaking.pre_replacements
5383     local word_head = head
5384
5385     while true do
5386         local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5387         if not lang then return head end
5388
5389         if not lbkr[lang] then
5390             break
5391         end
5392
5393         for k=1, #lbkr[lang] do
5394             local p = lbkr[lang][k].pattern
5395             local r = lbkr[lang][k].replace
5396
5397             while true do
5398                 local matches = { u.match(w, p) }
5399                 if #matches < 2 then break end
5400
5401                 local first = table.remove(matches, 1)
5402                 local last = table.remove(matches, #matches)
5403
5404                 &% Fix offsets, from bytes to unicode.
5405                 first = u.len(w:sub(1, first-1)) + 1
5406                 last = u.len(w:sub(1, last-1))
5407
5408                 local new &% used when inserting and removing nodes
5409                 local changed = 0
5410
5411                 &% This loop traverses the replace list and takes the
5412                 &% corresponding actions
5413                 for q = first, last do
5414                     local crep = r[q-first+1]
5415                     local char_node = wn[q]
5416                     local char_base = char_node
5417
5418                     if crep and crep.data then

```

```

5419         char_base = wn[crep.data+first-1]
5420     end
5421
5422     if crep == {} then
5423         break
5424     elseif crep == nil then
5425         changed = changed + 1
5426         node.remove(head, char_node)
5427     elseif crep and crep.string then
5428         changed = changed + 1
5429         local str = crep.string(matches)
5430         if str == '' then
5431             if q == 1 then
5432                 word_head = char_node.next
5433             end
5434             head, new = node.remove(head, char_node)
5435         elseif char_node.id == 29 and u.len(str) == 1 then
5436             char_node.char = string.utfvalue(str)
5437         else
5438             local n
5439             for s in string.utfvalues(str) do
5440                 if char_node.id == 7 then
5441                     log('Automatic hyphens cannot be replaced, just removed.')
5442                 else
5443                     n = node.copy(char_base)
5444                 end
5445                 n.char = s
5446                 if q == 1 then
5447                     head, new = node.insert_before(head, char_node, n)
5448                     word_head = new
5449                 else
5450                     node.insert_before(head, char_node, n)
5451                 end
5452             end
5453
5454             node.remove(head, char_node)
5455         end &% string length
5456     end &% if char and char.string
5457 end &% for char in match
5458 if changed > 20 then
5459     texio.write('Too many changes. Ignoring the rest.')
5460 elseif changed > 0 then
5461     &% For one-to-one can we modify directly the
5462     &% values without re-fetching? Very likely.
5463     w, wn, nw = Babel.fetch_subtext(word_head)
5464 end
5465
5466     end &% for match
5467 end &% for patterns
5468 word_head = nw
5469 end &% for words
5470 return head
5471 end
5472 &%% end of preliminary code for \babelprehyphenation
5473
5474 &% The following functions belong to the next macro
5475
5476 &% This table stores capture maps, numbered consecutively
5477 Babel.capture_maps = {}

```



```

5478
5479 function Babel.capture_func(key, cap)
5480   local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
5481   ret = ret:gsub('{{([0-9])|([^\]|+)|(-.)}}', Babel.capture_func_map)
5482   ret = ret:gsub("%[%[%]%%.%.%", '')
5483   ret = ret:gsub("%.%.%[%[%]%%", '')
5484   return key .. "[=function(m) return ]] .. ret .. [[ end]]
5485 end
5486
5487 function Babel.capt_map(from, mapno)
5488   return Babel.capture_maps[mapno][from] or from
5489 end
5490
5491 &% Handle the {n|abc|ABC} syntax in captures
5492 function Babel.capture_func_map(capno, from, to)
5493   local froms = {}
5494   for s in string.utfcharacters(from) do
5495     table.insert(froms, s)
5496   end
5497   local cnt = 1
5498   table.insert(Babel.capture_maps, {})
5499   local mlen = table.getn(Babel.capture_maps)
5500   for s in string.utfcharacters(to) do
5501     Babel.capture_maps[mlen][froms[cnt]] = s
5502     cnt = cnt + 1
5503   end
5504   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5505     (mlen) .. ").. " .. "["
5506 end
5507 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the {*n*} syntax. For example, `pre={1}{1}`- becomes `function(m) return m[1]..m[1]..'-' end`, where *m* are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to *m*[1]. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

```

5508 \catcode`\#=6
5509 \gdef\babelposthyphenation#1#2#3{&%
5510   \bbl@activateposthyphen
5511   \begingroup
5512     \def\babeltempa{\bbl@add@list\babeltempb}&%
5513     \let\babeltempb\@empty
5514     \bbl@foreach{#3}{&%
5515       \bbl@ifsamestring{##1}{remove}&%
5516       {\bbl@add@list\babeltempb{nil}}&%
5517       {\directlua{
5518         local rep = [[##1]]
5519         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5520         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5521         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5522         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5523         tex.print([[string\babeltempa{[]] .. rep .. [[]]])
5524       }}}&%

```

```

5525 \directlua{
5526     local lbkr = Babel.linebreaking.post_replacements
5527     local u = unicode.utf8
5528     %% Convert pattern:
5529     local patt = string.gsub(#[#2]=, '%s', '')
5530     if not u.find(patt, '()', nil, true) then
5531         patt = '()' .. patt .. '()'
5532     end
5533     patt = u.gsub(patt, '{(.)}',
5534         function (n)
5535             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5536         end)
5537     lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5538     table.insert(lbkr[\the\csname l@#1\endcsname],
5539         { pattern = patt, replace = { \babeltempb } })
5540 }%%
5541 \endgroup}
5542 % TODO. Working !!! Copypaste pattern.
5543 \gdef\babelprehyphenation#1#2#3{%%
5544     \bbl@activateprehyphen
5545     \begingroup
5546         \def\babeltempa{\bbl@add@list\babeltempb}%%
5547         \let\babeltempb\@empty
5548         \bbl@foreach{#3}{%%
5549             \bbl@ifsamestring{##1}{remove}%%
5550             {\bbl@add@list\babeltempb{nil}}%%
5551             {\directlua{
5552                 local rep = [#{##1}]
5553                 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5554                 tex.print([[\string\babeltempa{[]} .. rep .. [{}]])
5555             }}%%
5556         \directlua{
5557             local lbkr = Babel.linebreaking.pre_replacements
5558             local u = unicode.utf8
5559             %% Convert pattern:
5560             local patt = string.gsub(#[#2]=, '%s', '')
5561             if not u.find(patt, '()', nil, true) then
5562                 patt = '()' .. patt .. '()'
5563             end
5564             patt = u.gsub(patt, '{(.)}',
5565                 function (n)
5566                     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5567                 end)
5568             lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5569             table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5570                 { pattern = patt, replace = { \babeltempb } })
5571         }%%
5572     \endgroup}
5573 \endgroup
5574 \def\bbl@activateposthyphen{%
5575     \let\bbl@activateposthyphen\relax
5576     \directlua{
5577         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5578     }}
5579 % TODO. Working !!!
5580 \def\bbl@activateprehyphen{%
5581     \let\bbl@activateprehyphen\relax
5582     \directlua{
5583         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)

```

```
5584   }}
```

13.7 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```
5585 \bbl@trace{Redefinitions for bidi layout}
5586 \ifx\@eqnnum\undefined\else
5587   \ifx\bbl@attr@dir\undefined\else
5588     \edef\@eqnnum{%
5589       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5590       \unexpanded\expandafter{\@eqnnum}}%
5591   \fi
5592 \fi
5593 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5594 \ifnum\bbl@bidimode>\z@
5595   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5596     \bbl@exp{%
5597       \mathdir\the\bodydir
5598       #1%           Once entered in math, set boxes to restore values
5599       \<ifmmode>%
5600       \everyvbox{%
5601         \the\everyvbox
5602         \bodydir\the\bodydir
5603         \mathdir\the\mathdir
5604         \everyhbox{\the\everyhbox}%
5605         \everyvbox{\the\everyvbox}}%
5606       \everyhbox{%
5607         \the\everyhbox
5608         \bodydir\the\bodydir
5609         \mathdir\the\mathdir
5610         \everyhbox{\the\everyhbox}%
5611         \everyvbox{\the\everyvbox}}%
5612       \<fi>}}%
5613   \def\@hangfrom#1{%
5614     \setbox\@tempboxa\hbox{{#1}}%
5615     \hangindent\wd\@tempboxa
5616     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5617       \shapemode\@ne
5618     \fi
5619     \noindent\box\@tempboxa}
5620 \fi
5621 \IfBabelLayout{tabular}
5622   {\let\bbl@OL@tabular\@tabular
5623     \bbl@replace\@tabular{$}\bbl@nextfake$}%
5624   \let\bbl@NL@tabular\@tabular
5625   \AtBeginDocument{%
```

```

5626 \ifx\bb1@NL@@tabular\@tabular\else
5627 \bb1@replace\@tabular{$}\bb1@nextfake$}%
5628 \let\bb1@NL@@tabular\@tabular
5629 \fi}}
5630 {}
5631 \IfBabelLayout{lists}
5632 {\let\bb1@OL@list\list
5633 \bb1@sreplace\list{\parshape}\bb1@listparshape}%
5634 \let\bb1@NL@list\list
5635 \def\bb1@listparshape#1#2#3{%
5636 \parshape #1 #2 #3 %
5637 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5638 \shapemode\tw@
5639 \fi}}
5640 {}
5641 \IfBabelLayout{graphics}
5642 {\let\bb1@pictresetdir\relax
5643 \def\bb1@pictsetdir{%
5644 \ifcase\bb1@thetextdir
5645 \let\bb1@pictresetdir\relax
5646 \else
5647 \textdir TLT\relax
5648 \def\bb1@pictresetdir{\textdir TRT\relax}%
5649 \fi}%
5650 \let\bb1@OL@@picture\@picture
5651 \let\bb1@OL@put\put
5652 \bb1@sreplace\@picture{\hskip-}\bb1@pictsetdir\hskip-}%
5653 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5654 \@killglue
5655 \raise#2\unitlength
5656 \hb@xt@z@{\kern#1\unitlength\bb1@pictresetdir#3}\hss}}%
5657 \AtBeginDocument
5658 {\ifx\tikz@atbegin@node\undefined\else
5659 \let\bb1@OL@pgfpicture\pgfpicture
5660 \bb1@sreplace\pgfpicture{\pgfpicturetrue}\bb1@pictsetdir\pgfpicturetrue}%
5661 \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir}%
5662 \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5663 \fi}}
5664 {}

```

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.

```

5665 \IfBabelLayout{counters}%
5666 {\let\bb1@OL@@textsuperscript\@textsuperscript
5667 \bb1@sreplace\@textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5668 \let\bb1@latinarabic=\@arabic
5669 \let\bb1@OL@@arabic\@arabic
5670 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
5671 \@ifpackagewith{babel}{bidi=default}%
5672 {\let\bb1@asciroman=\@roman
5673 \let\bb1@OL@@roman\@roman
5674 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
5675 \let\bb1@asciiRoman=\@Roman
5676 \let\bb1@OL@@roman\@Roman
5677 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}%
5678 \let\bb1@OL@labelenumii\labelenumii
5679 \def\labelenumii{}\theenumii}%
5680 \let\bb1@OL@p@enumiii\p@enumiii

```

```

5681 \def\p@enumiii{\p@enumii}\theenumii{}{}{}{}
5682 <<Footnote changes>>
5683 \IfBabelLayout{footnotes}%
5684 {\let\bbl@OL@footnote\footnote
5685 \BabelFootnote\footnote\language{}{}%
5686 \BabelFootnote\localfootnote\language{}{}%
5687 \BabelFootnote\mainfootnote{}{}{}{}
5688 {}

```

Some \LaTeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5689 \IfBabelLayout{extras}%
5690 {\let\bbl@OL@underline\underline
5691 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5692 \let\bbl@OL@LaTeX2e\LaTeX2e
5693 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5694 \if b\expandafter\@car\@series\@nil\boldmath\fi
5695 \babelsublr}%
5696 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
5697 {}
5698 </luatex>

```

13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```

5699 (*basic-r)
5700 Babel = Babel or {}
5701
5702 Babel.bidi_enabled = true
5703
5704 require('babel-data-bidi.lua')
5705
5706 local characters = Babel.characters
5707 local ranges = Babel.ranges
5708
5709 local DIR = node.id("dir")
5710
5711 local function dir_mark(head, from, to, outer)
5712   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5713   local d = node.new(DIR)
5714   d.dir = '+' .. dir
5715   node.insert_before(head, from, d)
5716   d = node.new(DIR)
5717   d.dir = '-' .. dir
5718   node.insert_after(head, to, d)
5719 end
5720
5721 function Babel.bidi(head, ispar)
5722   local first_n, last_n          -- first and last char with nums
5723   local last_es                 -- an auxiliary 'last' used with nums
5724   local first_d, last_d         -- first and last char in L/R block
5725   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):

```

5726   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5727   local strong_lr = (strong == 'l') and 'l' or 'r'
5728   local outer = strong
5729
5730   local new_dir = false
5731   local first_dir = false
5732   local inmath = false
5733
5734   local last_lr
5735
5736   local type_n = ''
5737
5738   for item in node.traverse(head) do
5739
5740     -- three cases: glyph, dir, otherwise
5741     if item.id == node.id'glyph'
5742       or (item.id == 7 and item.subtype == 2) then
5743
5744       local itemchar

```

```

5745     if item.id == 7 and item.subtype == 2 then
5746         itemchar = item.replace.char
5747     else
5748         itemchar = item.char
5749     end
5750     local chardata = characters[itemchar]
5751     dir = chardata and chardata.d or nil
5752     if not dir then
5753         for nn, et in ipairs(ranges) do
5754             if itemchar < et[1] then
5755                 break
5756             elseif itemchar <= et[2] then
5757                 dir = et[3]
5758                 break
5759             end
5760         end
5761     end
5762     dir = dir or 'l'
5763     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).

```

5764     if new_dir then
5765         attr_dir = 0
5766         for at in node.traverse(item.attr) do
5767             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5768                 attr_dir = at.value % 3
5769             end
5770         end
5771         if attr_dir == 1 then
5772             strong = 'r'
5773         elseif attr_dir == 2 then
5774             strong = 'al'
5775         else
5776             strong = 'l'
5777         end
5778         strong_lr = (strong == 'l') and 'l' or 'r'
5779         outer = strong_lr
5780         new_dir = false
5781     end
5782
5783     if dir == 'nsm' then dir = strong end -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

5784     dir_real = dir -- We need dir_real to set strong below
5785     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:

```

5786     if strong == 'al' then
5787         if dir == 'en' then dir = 'an' end -- W2
5788         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5789         strong_lr = 'r' -- W3
5790     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

5791 elseif item.id == node.id'dir' and not inmath then
5792     new_dir = true
5793     dir = nil
5794 elseif item.id == node.id'math' then
5795     inmath = (item.subtype == 0)
5796 else
5797     dir = nil          -- Not a char
5798 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>.

```

5799 if dir == 'en' or dir == 'an' or dir == 'et' then
5800     if dir ~= 'et' then
5801         type_n = dir
5802     end
5803     first_n = first_n or item
5804     last_n = last_es or item
5805     last_es = nil
5806 elseif dir == 'es' and last_n then -- W3+W6
5807     last_es = item
5808 elseif dir == 'cs' then          -- it's right - do nothing
5809     if first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5810         if strong_lr == 'r' and type_n ~= '' then
5811             dir_mark(head, first_n, last_n, 'r')
5812         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5813             dir_mark(head, first_n, last_n, 'r')
5814             dir_mark(head, first_d, last_d, outer)
5815             first_d, last_d = nil, nil
5816         elseif strong_lr == 'l' and type_n ~= '' then
5817             last_d = last_n
5818         end
5819         type_n = ''
5820         first_n, last_n = nil, nil
5821     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:

```

5822 if dir == 'l' or dir == 'r' then
5823     if dir ~= outer then
5824         first_d = first_d or item
5825         last_d = item
5826     elseif first_d and dir ~= strong_lr then
5827         dir_mark(head, first_d, last_d, outer)
5828         first_d, last_d = nil, nil
5829     end
5830 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.

```

5831   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5832       item.char = characters[item.char] and
5833           characters[item.char].m or item.char
5834   elseif (dir or new_dir) and last_lr ~= item then
5835       local mir = outer .. strong_lr .. (dir or outer)
5836       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5837           for ch in node.traverse(node.next(last_lr)) do
5838               if ch == item then break end
5839               if ch.id == node.id'glyph' and characters[ch.char] then
5840                   ch.char = characters[ch.char].m or ch.char
5841               end
5842           end
5843       end
5844   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).

```

5845   if dir == 'l' or dir == 'r' then
5846       last_lr = item
5847       strong = dir_real          -- Don't search back - best save now
5848       strong_lr = (strong == 'l') and 'l' or 'r'
5849   elseif new_dir then
5850       last_lr = nil
5851   end
5852 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

5853   if last_lr and outer == 'r' then
5854       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5855           if characters[ch.char] then
5856               ch.char = characters[ch.char].m or ch.char
5857           end
5858       end
5859   end
5860   if first_n then
5861       dir_mark(head, first_n, last_n, outer)
5862   end
5863   if first_d then
5864       dir_mark(head, first_d, last_d, outer)
5865   end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

5866   return node.prev(head) or head
5867 end
5868 </basic-r>

```

And here the Lua code for bidi=basic:

```

5869 <(*basic)
5870 Babel = Babel or {}
5871
5872 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5873
5874 Babel.fontmap = Babel.fontmap or {}
5875 Babel.fontmap[0] = {}          -- 1

```

```

5876 Babel.fontmap[1] = {}      -- r
5877 Babel.fontmap[2] = {}      -- al/an
5878
5879 Babel.bidi_enabled = true
5880 Babel.mirroring_enabled = true
5881
5882 require('babel-data-bidi.lua')
5883
5884 local characters = Babel.characters
5885 local ranges = Babel.ranges
5886
5887 local DIR = node.id('dir')
5888 local GLYPH = node.id('glyph')
5889
5890 local function insert_implicit(head, state, outer)
5891   local new_state = state
5892   if state.sim and state.eim and state.sim ~= state.eim then
5893     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5894     local d = node.new(DIR)
5895     d.dir = '+' .. dir
5896     node.insert_before(head, state.sim, d)
5897     local d = node.new(DIR)
5898     d.dir = '-' .. dir
5899     node.insert_after(head, state.eim, d)
5900   end
5901   new_state.sim, new_state.eim = nil, nil
5902   return head, new_state
5903 end
5904
5905 local function insert_numeric(head, state)
5906   local new
5907   local new_state = state
5908   if state.san and state.ean and state.san ~= state.ean then
5909     local d = node.new(DIR)
5910     d.dir = '+TLT'
5911     _, new = node.insert_before(head, state.san, d)
5912     if state.san == state.sim then state.sim = new end
5913     local d = node.new(DIR)
5914     d.dir = '-TLT'
5915     _, new = node.insert_after(head, state.ean, d)
5916     if state.ean == state.eim then state.eim = new end
5917   end
5918   new_state.san, new_state.ean = nil, nil
5919   return head, new_state
5920 end
5921
5922 -- TODO - \hbox with an explicit dir can lead to wrong results
5923 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5924 -- was s made to improve the situation, but the problem is the 3-dir
5925 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5926 -- well.
5927
5928 function Babel.bidi(head, ispar, hdir)
5929   local d -- d is used mainly for computations in a loop
5930   local prev_d = ''
5931   local new_d = false
5932
5933   local nodes = {}
5934   local outer_first = nil

```

```

5935 local inmath = false
5936
5937 local glue_d = nil
5938 local glue_i = nil
5939
5940 local has_en = false
5941 local first_et = nil
5942
5943 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5944
5945 local save_outer
5946 local temp = node.get_attribute(head, ATDIR)
5947 if temp then
5948     temp = temp % 3
5949     save_outer = (temp == 0 and 'l') or
5950                 (temp == 1 and 'r') or
5951                 (temp == 2 and 'al')
5952 elseif ispar then -- Or error? Shouldn't happen
5953     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5954 else -- Or error? Shouldn't happen
5955     save_outer = ('TRT' == hdir) and 'r' or 'l'
5956 end
5957 -- when the callback is called, we are just _after_ the box,
5958 -- and the textdir is that of the surrounding text
5959 -- if not ispar and hdir ~= tex.textdir then
5960 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5961 -- end
5962 local outer = save_outer
5963 local last = outer
5964 -- 'al' is only taken into account in the first, current loop
5965 if save_outer == 'al' then save_outer = 'r' end
5966
5967 local fontmap = Babel.fontmap
5968
5969 for item in node.traverse(head) do
5970
5971     -- In what follows, #node is the last (previous) node, because the
5972     -- current one is not added until we start processing the neutrals.
5973
5974     -- three cases: glyph, dir, otherwise
5975     if item.id == GLYPH
5976         or (item.id == 7 and item.subtype == 2) then
5977
5978         local d_font = nil
5979         local item_r
5980         if item.id == 7 and item.subtype == 2 then
5981             item_r = item.replace -- automatic discs have just 1 glyph
5982         else
5983             item_r = item
5984         end
5985         local chardata = characters[item_r.char]
5986         d = chardata and chardata.d or nil
5987         if not d or d == 'nsm' then
5988             for nn, et in ipairs(ranges) do
5989                 if item_r.char < et[1] then
5990                     break
5991                 elseif item_r.char <= et[2] then
5992                     if not d then d = et[3]
5993                     elseif d == 'nsm' then d_font = et[3]

```

```

5994         end
5995         break
5996     end
5997 end
5998 end
5999 d = d or 'l'
6000
6001 -- A short 'pause' in bidi for mapfont
6002 d_font = d_font or d
6003 d_font = (d_font == 'l' and 0) or
6004           (d_font == 'nsm' and 0) or
6005           (d_font == 'r' and 1) or
6006           (d_font == 'al' and 2) or
6007           (d_font == 'an' and 2) or nil
6008 if d_font and fontmap and fontmap[d_font][item_r.font] then
6009     item_r.font = fontmap[d_font][item_r.font]
6010 end
6011
6012 if new_d then
6013     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6014     if inmath then
6015         attr_d = 0
6016     else
6017         attr_d = node.get_attribute(item, ATDIR)
6018         attr_d = attr_d % 3
6019     end
6020     if attr_d == 1 then
6021         outer_first = 'r'
6022         last = 'r'
6023     elseif attr_d == 2 then
6024         outer_first = 'r'
6025         last = 'al'
6026     else
6027         outer_first = 'l'
6028         last = 'l'
6029     end
6030     outer = last
6031     has_en = false
6032     first_et = nil
6033     new_d = false
6034 end
6035
6036 if glue_d then
6037     if (d == 'l' and 'l' or 'r') ~= glue_d then
6038         table.insert(nodes, {glue_i, 'on', nil})
6039     end
6040     glue_d = nil
6041     glue_i = nil
6042 end
6043
6044 elseif item.id == DIR then
6045     d = nil
6046     new_d = true
6047
6048 elseif item.id == node.id'glue' and item.subtype == 13 then
6049     glue_d = d
6050     glue_i = item
6051     d = nil
6052

```

```

6053     elseif item.id == node.id'math' then
6054         inmath = (item.subtype == 0)
6055
6056     else
6057         d = nil
6058     end
6059
6060     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6061     if last == 'al' and d == 'en' then
6062         d = 'an'          -- W3
6063     elseif last == 'al' and (d == 'et' or d == 'es') then
6064         d = 'on'          -- W6
6065     end
6066
6067     -- EN + CS/ES + EN      -- W4
6068     if d == 'en' and #nodes >= 2 then
6069         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6070             and nodes[#nodes-1][2] == 'en' then
6071             nodes[#nodes][2] = 'en'
6072         end
6073     end
6074
6075     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6076     if d == 'an' and #nodes >= 2 then
6077         if (nodes[#nodes][2] == 'cs')
6078             and nodes[#nodes-1][2] == 'an' then
6079             nodes[#nodes][2] = 'an'
6080         end
6081     end
6082
6083     -- ET/EN                  -- W5 + W7->l / W6->on
6084     if d == 'et' then
6085         first_et = first_et or (#nodes + 1)
6086     elseif d == 'en' then
6087         has_en = true
6088         first_et = first_et or (#nodes + 1)
6089     elseif first_et then      -- d may be nil here !
6090         if has_en then
6091             if last == 'l' then
6092                 temp = 'l'    -- W7
6093             else
6094                 temp = 'en'   -- W5
6095             end
6096         else
6097             temp = 'on'       -- W6
6098         end
6099         for e = first_et, #nodes do
6100             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6101         end
6102         first_et = nil
6103         has_en = false
6104     end
6105
6106     if d then
6107         if d == 'al' then
6108             d = 'r'
6109             last = 'al'
6110         elseif d == 'l' or d == 'r' then
6111             last = d

```

```

6112     end
6113     prev_d = d
6114     table.insert(nodes, {item, d, outer_first})
6115 end
6116
6117     outer_first = nil
6118
6119 end
6120
6121 -- TODO -- repeated here in case EN/ET is the last node. Find a
6122 -- better way of doing things:
6123 if first_et then      -- dir may be nil here !
6124     if has_en then
6125         if last == 'l' then
6126             temp = 'l'      -- W7
6127         else
6128             temp = 'en'     -- W5
6129         end
6130     else
6131         temp = 'on'        -- W6
6132     end
6133     for e = first_et, #nodes do
6134         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6135     end
6136 end
6137
6138 -- dummy node, to close things
6139 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6140
6141 ----- NEUTRAL -----
6142
6143 outer = save_outer
6144 last = outer
6145
6146 local first_on = nil
6147
6148 for q = 1, #nodes do
6149     local item
6150
6151     local outer_first = nodes[q][3]
6152     outer = outer_first or outer
6153     last = outer_first or last
6154
6155     local d = nodes[q][2]
6156     if d == 'an' or d == 'en' then d = 'r' end
6157     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6158
6159     if d == 'on' then
6160         first_on = first_on or q
6161     elseif first_on then
6162         if last == d then
6163             temp = d
6164         else
6165             temp = outer
6166         end
6167         for r = first_on, q - 1 do
6168             nodes[r][2] = temp
6169             item = nodes[r][1]      -- MIRRORING
6170             if Babel.mirroring_enabled and item.id == GLYPH

```

```

6171         and temp == 'r' and characters[item.char] then
6172             local font_mode = font.fonts[item.font].properties.mode
6173             if font_mode ~= 'harf' and font_mode ~= 'plug' then
6174                 item.char = characters[item.char].m or item.char
6175             end
6176         end
6177     end
6178     first_on = nil
6179 end
6180
6181 if d == 'r' or d == 'l' then last = d end
6182 end
6183
6184 ----- IMPLICIT, REORDER -----
6185
6186 outer = save_outer
6187 last = outer
6188
6189 local state = {}
6190 state.has_r = false
6191
6192 for q = 1, #nodes do
6193     local item = nodes[q][1]
6194
6195     outer = nodes[q][3] or outer
6196
6197     local d = nodes[q][2]
6198
6199     if d == 'nsm' then d = last end          -- W1
6200     if d == 'en' then d = 'an' end
6201     local isdir = (d == 'r' or d == 'l')
6202
6203     if outer == 'l' and d == 'an' then
6204         state.san = state.san or item
6205         state.ean = item
6206     elseif state.san then
6207         head, state = insert_numeric(head, state)
6208     end
6209
6210     if outer == 'l' then
6211         if d == 'an' or d == 'r' then      -- im -> implicit
6212             if d == 'r' then state.has_r = true end
6213             state.sim = state.sim or item
6214             state.eim = item
6215         elseif d == 'l' and state.sim and state.has_r then
6216             head, state = insert_implicit(head, state, outer)
6217         elseif d == 'l' then
6218             state.sim, state.eim, state.has_r = nil, nil, false
6219         end
6220     else
6221         if d == 'an' or d == 'l' then
6222             if nodes[q][3] then -- nil except after an explicit dir
6223                 state.sim = item -- so we move sim 'inside' the group
6224             else
6225                 state.sim = state.sim or item
6226             end
6227             state.eim = item
6228         elseif d == 'r' and state.sim then

```

```

6230         head, state = insert_implicit(head, state, outer)
6231     elseif d == 'r' then
6232         state.sim, state.eim = nil, nil
6233     end
6234 end
6235
6236 if isdir then
6237     last = d          -- Don't search back - best save now
6238 elseif d == 'on' and state.san then
6239     state.san = state.san or item
6240     state.ean = item
6241 end
6242
6243 end
6244
6245 return node.prev(head) or head
6246 end
6247 </basic>

```

14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},

```

For the meaning of these codes, see the Unicode standard.

15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```

6248 <nil>
6249 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6250 \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.

```

6251 \ifx\l@nil\@undefined
6252   \newlanguage\l@nil
6253   \@namedef{bbl@hyphendata@the\l@nil}{\{}}{\{}}% Remove warning
6254   \let\bbl@elt\relax
6255   \edef\bbl@languages{% Add it to the list of languages
6256     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}{\{}}
6257 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

6258 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```


The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 6259 \let\captionnil\@empty
6260 \let\datenil\@empty
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
6261 \ldf@finish{nil}
6262 \</nil>
```

16 Support for Plain \TeX (plain.def)

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based \TeX -format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `locallyhyphen.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 \TeX , 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 \TeX sees, we need to set some category codes just to be able to change the definition of `\input`.

```
6263 (*bplain | blplain)
6264 \catcode`\{=1 % left brace is begin-group character
6265 \catcode`\}=2 % right brace is end-group character
6266 \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).

```
6267 \openin 0 hyphen.cfg
6268 \ifeof0
6269 \else
6270 \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.

```
6271 \def\input #1 {%
6272 \let\input\input
6273 \a hyphen.cfg
6274 \let\input\undefined
6275 }
6276 \fi
6277 \</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`.

```
6278 \bplain\la plain.tex
6279 \bplain\la 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.

```
6280 \bplain\def\fmtname{babel-plain}
6281 \bplain\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of $\text{\LaTeX} 2_{\epsilon}$ that are needed for `babel`.

```
6282 \langle *Emulate LaTeX \rangle \equiv
6283 % == Code for plain ==
6284 \def\@empty{}
6285 \def\loadlocalcfg#1{%
6286   \openin0#1.cfg
6287   \ifeof0
6288     \closein0
6289   \else
6290     \closein0
6291     {\immediate\write16{*****}%
6292      \immediate\write16{* Local config file #1.cfg used}%
6293      \immediate\write16{*}%
6294     }
6295     \input #1.cfg\relax
6296   \fi
6297   \@endoflfd}
```

16.3 General tools

A number of \LaTeX macro's that are needed later on.

```
6298 \long\def\@firstofone#1{#1}
6299 \long\def\@firstoftwo#1#2{#1}
6300 \long\def\@secondoftwo#1#2{#2}
6301 \def\@nnil{\@nil}
6302 \def\@gobbletwo#1#2{}
6303 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6304 \def\@star@or@long#1{%
6305   \@ifstar
6306   {\let\l@ngrel@x\relax#1}%
6307   {\let\l@ngrel@x\long#1}}
6308 \let\l@ngrel@x\relax
6309 \def\@car#1#2\@nil{#1}
6310 \def\@cdr#1#2\@nil{#2}
6311 \let\@typeset@protect\relax
6312 \let\protected@edef\edef
6313 \long\def\@gobble#1{}
6314 \edef\@backslashchar{\expandafter\@gobble\string\}
6315 \def\strip@prefix#1>{}
6316 \def\g@addto@macro#1#2{{%
6317   \toks@\expandafter{#1#2}%
6318   \xdef#1{\the\toks@}}}
```

```

6319 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6320 \def\@nameuse#1{\csname #1\endcsname}
6321 \def\@ifundefined#1{%
6322   \expandafter\ifx\csname#1\endcsname\relax
6323     \expandafter\@firstoftwo
6324   \else
6325     \expandafter\@secondoftwo
6326   \fi}
6327 \def\@expandtwoargs#1#2#3{%
6328   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6329 \def\zap@space#1 #2{%
6330   #1%
6331   \ifx#2\@empty\else\expandafter\zap@space\fi
6332   #2}
6333 \let\bbl@trace\@gobble

```

$\text{\LaTeX 2}_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

6334 \ifx\@preamblecmds\@undefined
6335   \def\@preamblecmds{}
6336 \fi
6337 \def\@onlypreamble#1{%
6338   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6339     \@preamblecmds\do#1}}
6340 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

6341 \def\begin{document}{%
6342   \@begin{document}hook
6343   \global\let\@begin{document}hook\@undefined
6344   \def\do##1{\global\let##1\@undefined}%
6345   \@preamblecmds
6346   \global\let\do\noexpand}
6347 \ifx\@begin{document}hook\@undefined
6348   \def\@begin{document}hook{}
6349 \fi
6350 \@onlypreamble\@begin{document}hook
6351 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

6352 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6353 \@onlypreamble\AtEndOfPackage
6354 \def\@endofldf{}
6355 \@onlypreamble\@endofldf
6356 \let\bbl@afterlang\@empty
6357 \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.

```

6358 \catcode`\&=\z@
6359 \ifx&\if@files\@undefined
6360   \expandafter\let\csname if@files\endcsname
6361     \csname iffalse\endcsname
6362 \fi
6363 \catcode`\&=4

```

Mimick L^AT_EX's commands to define control sequences.

```

6364 \def\newcommand{\@star@or@long\new@command}
6365 \def\new@command#1{%
6366   \@testopt{\@newcommand#1}0}
6367 \def\@newcommand#1[#2]{%
6368   \@ifnextchar [{\@xargdef#1[#2]]%
6369               {\@argdef#1[#2]}}
6370 \long\def\@argdef#1[#2]#3{%
6371   \@yargdef#1\@ne{#2}{#3}}
6372 \long\def\@xargdef#1[#2][#3]#4{%
6373   \expandafter\def\expandafter#1\expandafter{%
6374     \expandafter\@protected@testopt\expandafter #1%
6375     \csname\string#1\expandafter\endcsname{#3}}%
6376   \expandafter\@yargdef \csname\string#1\endcsname
6377   \tw@{#2}{#4}}
6378 \long\def\@yargdef#1#2#3{%
6379   \@tempcnta#3\relax
6380   \advance \@tempcnta \@ne
6381   \let\@hash@\relax
6382   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6383   \@tempcntb #2%
6384   \@whilenum\@tempcntb <\@tempcnta
6385   \do{%
6386     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6387     \advance\@tempcntb \@ne}%
6388   \let\@hash@###
6389   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
6390 \def\providecommand{\@star@or@long\provide@command}
6391 \def\provide@command#1{%
6392   \begingroup
6393     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
6394   \endgroup
6395   \expandafter\ifundefined\@gtempa
6396     {\def\reserved@a{\new@command#1}}%
6397     {\let\reserved@a\relax
6398     \def\reserved@a{\new@command\reserved@a}}%
6399   \reserved@a}%
6400 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6401 \def\declare@robustcommand#1{%
6402   \edef\reserved@a{\string#1}%
6403   \def\reserved@b{#1}%
6404   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6405   \edef#1{%
6406     \ifx\reserved@a\reserved@b
6407       \noexpand\x@protect
6408       \noexpand#1%
6409     \fi
6410     \noexpand\protect
6411     \expandafter\noexpand\csname
6412       \expandafter\@gobble\string#1 \endcsname
6413   }%
6414   \expandafter\new@command\csname
6415     \expandafter\@gobble\string#1 \endcsname
6416 }
6417 \def\x@protect#1{%
6418   \ifx\protect\@typeset@protect\else
6419     \@x@protect#1%
6420   \fi

```

```

6421 }
6422 \catcode`\&=\z@ % Trick to hide conditionals
6423 \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`.

```

6424 \def\bbl@tempa{\csname newif\endcsname&fin@}
6425 \catcode`\&=4
6426 \ifx\in@\@undefined
6427 \def\in@#1#2{%
6428   \def\in@@##1#1##2##3\in@{%
6429     \ifx\in@@##2\in@false\else\in@true\fi}%
6430   \in@@#2#1\in@\in@@}
6431 \else
6432 \let\bbl@tempa\@empty
6433 \fi
6434 \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).

```

6435 \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.

```

6436 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\text{\LaTeX} 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

6437 \ifx\@tempcnta\@undefined
6438 \csname newcount\endcsname\@tempcnta\relax
6439 \fi
6440 \ifx\@tempcntb\@undefined
6441 \csname newcount\endcsname\@tempcntb\relax
6442 \fi

```

To prevent wasting two counters in $\text{\LaTeX} 2.09$ (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

6443 \ifx\bye\@undefined
6444 \advance\count10 by -2\relax
6445 \fi
6446 \ifx\@ifnextchar\@undefined
6447 \def\@ifnextchar#1#2#3{%
6448   \let\reserved@d=#1%
6449   \def\reserved@a{#2}\def\reserved@b{#3}%
6450   \futurelet\@let@token\@ifnch}
6451 \def\@ifnch{%
6452   \ifx\@let@token\@sptoken
6453     \let\reserved@c\@xifnch
6454   \else
6455     \ifx\@let@token\reserved@d
6456       \let\reserved@c\reserved@a

```

```

6457     \else
6458         \let\reserved@c\reserved@b
6459     \fi
6460 \fi
6461     \reserved@c}
6462 \def\:\let\sptoken= } \: % this makes \@sptoken a space token
6463 \def\:\@xifnch} \expandafter\def\:\{\futurelet\@let@token\@ifnch}
6464 \fi
6465 \def\@testopt#1#2{%
6466     \ifnextchar[{\#1}{\#1[\#2]}}
6467 \def\@protected@testopt#1{%
6468     \ifx\protect\@typeset@protect
6469         \expandafter\@testopt
6470     \else
6471         \@x@protect#1%
6472     \fi}
6473 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6474     #2\relax}\fi}
6475 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6476     \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

6477 \def\DeclareTextCommand{%
6478     \@dec@text@cmd\providecommand
6479 }
6480 \def\ProvideTextCommand{%
6481     \@dec@text@cmd\providecommand
6482 }
6483 \def\DeclareTextSymbol#1#2#3{%
6484     \@dec@text@cmd\chardef#1{#2}#3\relax
6485 }
6486 \def\@dec@text@cmd#1#2#3{%
6487     \expandafter\def\expandafter#2%
6488         \expandafter{%
6489             \csname#3-cmd\expandafter\endcsname
6490             \expandafter#2%
6491             \csname#3\string#2\endcsname
6492         }%
6493 %     \let\@ifdefinable\rc@ifdefinable
6494     \expandafter#1\csname#3\string#2\endcsname
6495 }
6496 \def\@current@cmd#1{%
6497     \ifx\protect\@typeset@protect\else
6498         \noexpand#1\expandafter\@gobble
6499     \fi
6500 }
6501 \def\@changed@cmd#1#2{%
6502     \ifx\protect\@typeset@protect
6503         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6504             \expandafter\ifx\csname ?\string#1\endcsname\relax
6505                 \expandafter\def\csname ?\string#1\endcsname{%
6506                     \@changed@x@err{#1}%
6507                 }%
6508             \fi
6509             \global\expandafter\let
6510             \csname\cf@encoding \string#1\expandafter\endcsname

```

```

6511         \csname ?\string#1\endcsname
6512     \fi
6513     \csname\cf@encoding\string#1%
6514         \expandafter\endcsname
6515 \else
6516     \noexpand#1%
6517 \fi
6518 }
6519 \def\@changed@x@err#1{%
6520     \errhelp{Your command will be ignored, type <return> to proceed}%
6521     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6522 \def\DeclareTextCommandDefault#1{%
6523     \DeclareTextCommand#1?%
6524 }
6525 \def\ProvideTextCommandDefault#1{%
6526     \ProvideTextCommand#1?%
6527 }
6528 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6529 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6530 \def\DeclareTextAccent#1#2#3{%
6531     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6532 }
6533 \def\DeclareTextCompositeCommand#1#2#3#4{%
6534     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6535     \edef\reserved@b{\string##1}%
6536     \edef\reserved@c{%
6537         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6538     \ifx\reserved@b\reserved@c
6539         \expandafter\expandafter\expandafter\ifx
6540             \expandafter\@car\reserved@a\relax\relax\@nil
6541             \@text@composite
6542     \else
6543         \edef\reserved@b##1{%
6544             \def\expandafter\noexpand
6545                 \csname#2\string#1\endcsname####1{%
6546                 \noexpand\@text@composite
6547                 \expandafter\noexpand\csname#2\string#1\endcsname
6548                 ####1\noexpand\@empty\noexpand\@text@composite
6549                 {##1}%
6550             }%
6551         }%
6552         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6553     \fi
6554     \expandafter\def\csname\expandafter\string\csname
6555         #2\endcsname\string#1-\string#3\endcsname{#4}
6556 \else
6557     \errhelp{Your command will be ignored, type <return> to proceed}%
6558     \errmessage{\string\DeclareTextCompositeCommand\space used on
6559         inappropriate command \protect#1}
6560 \fi
6561 }
6562 \def\@text@composite#1#2#3\@text@composite{%
6563     \expandafter\@text@composite@x
6564     \csname\string#1-\string#2\endcsname
6565 }
6566 \def\@text@composite@x#1#2{%
6567     \ifx#1\relax
6568         #2%
6569     \else

```

```

6570      #1%
6571      \fi
6572 }
6573 %
6574 \def\@strip@args#1:#2-#3\@strip@args{#2}
6575 \def\DeclareTextComposite#1#2#3#4{%
6576     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6577     \bgroup
6578         \lccode`\@=#4%
6579         \lowercase{%
6580     \egroup
6581     \reserved@a @%
6582 }%
6583 }
6584 %
6585 \def\UseTextSymbol#1#2{#2}
6586 \def\UseTextAccent#1#2#3{
6587 \def\@use@text@encoding#1{
6588 \def\DeclareTextSymbolDefault#1#2{%
6589     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6590 }
6591 \def\DeclareTextAccentDefault#1#2{%
6592     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6593 }
6594 \def\cf@encoding{OT1}

```

Currently we only use the $\LaTeX 2_{\epsilon}$ method for accents for those that are known to be made active in *some* language definition file.

```

6595 \DeclareTextAccent{"}{OT1}{127}
6596 \DeclareTextAccent{'}{OT1}{19}
6597 \DeclareTextAccent{^}{OT1}{94}
6598 \DeclareTextAccent{\`}{OT1}{18}
6599 \DeclareTextAccent{\~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```

6600 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6601 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
6602 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
6603 \DeclareTextSymbol{\textquoteright}{OT1}{''}
6604 \DeclareTextSymbol{\i}{OT1}{16}
6605 \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`.

```

6606 \ifx\scriptsize\@undefined
6607     \let\scriptsize\sevenrm
6608 \fi
6609 % End of code for plain
6610 <</Emulate LaTeX>>

```

A proxy file:

```

6611 <(*plain>
6612 \input babel.def
6613 </plain>

```


17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).