

Babel

Version 3.32

2019/06/03

Original author

Johannes L. Braams

Current maintainer

Javier Bezos

The standard distribution of \LaTeX contains a number of document classes that are meant to be used, but also serve as examples for other users to create their own document classes. These document classes have become very popular among \LaTeX users. But it should be kept in mind that they were designed for American tastes and typography. At one time they even contained a number of hard-wired texts.

This manual describes babel, a package that makes use of the capabilities of \TeX , xetex and luatex to provide an environment in which documents can be typeset in a language other than US English, or in more than one language or script.

Current development is focused on Unicode engines (Xe \TeX and Lua \TeX) and the so-called *complex scripts*. New features related to font selection, bidi writing, line breaking and so on are being added incrementally.

Babel provides support (total or partial) for about 200 languages, either as a “classical” package option or as an ini file. Furthermore, new languages can be created from scratch easily.

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	5
1.3	Modifiers	6
1.4	xelatex and luatex	7
1.5	Troubleshooting	7
1.6	Plain	8
1.7	Basic language selectors	8
1.8	Auxiliary language selectors	9
1.9	More on selection	10
1.10	Shorthands	11
1.11	Package options	14
1.12	The base option	16
1.13	ini files	17
1.14	Selecting fonts	24
1.15	Modifying a language	26
1.16	Creating a language	26
1.17	Digits	29
1.18	Getting the current language name	29
1.19	Hyphenation and line breaking	30
1.20	Selecting scripts	31
1.21	Selecting directions	32
1.22	Language attributes	36
1.23	Hooks	36
1.24	Languages supported by babel with ldf files	37
1.25	Unicode character properties in luatex	39
1.26	Tips, workarounds, know issues and notes	39
1.27	Current and future work	40
1.28	Tentative and experimental code	41
2	Loading languages with language.dat	41
2.1	Format	41
3	The interface between the core of babel and the language definition files	42
3.1	Guidelines for contributed languages	43
3.2	Basic macros	44
3.3	Skeleton	45
3.4	Support for active characters	46
3.5	Support for saving macro definitions	47
3.6	Support for extending macros	47
3.7	Macros common to a number of languages	47
3.8	Encoding-dependent strings	47
4	Changes	51
4.1	Changes in babel version 3.9	51
II	Source code	52
5	Identification and loading of required files	52

6	locale directory	52
7	Tools	53
7.1	Multiple languages	57
8	The Package File (\LaTeX, babel.sty)	57
8.1	base	58
8.2	key=value options and other general option	60
8.3	Conditional loading of shorthands	61
8.4	Language options	63
9	The kernel of Babel (babel.def, common)	65
9.1	Tools	66
9.2	Hooks	68
9.3	Setting up language files	70
9.4	Shorthands	72
9.5	Language attributes	81
9.6	Support for saving macro definitions	83
9.7	Short tags	84
9.8	Hyphens	84
9.9	Multiencoding strings	86
9.10	Macros common to a number of languages	92
9.11	Making glyphs available	92
9.11.1	Quotation marks	92
9.11.2	Letters	93
9.11.3	Shorthands for quotation marks	94
9.11.4	Umlauts and tremas	95
9.12	Layout	96
9.13	Load engine specific macros	97
9.14	Creating languages	97
10	The kernel of Babel (babel.def, only \LaTeX)	107
10.1	The redefinition of the style commands	107
10.2	Cross referencing macros	108
10.3	Marks	111
10.4	Preventing clashes with other packages	112
10.4.1	ifthen	112
10.4.2	varioref	113
10.4.3	hhline	113
10.4.4	hyperref	114
10.4.5	fancyhdr	114
10.5	Encoding and fonts	114
10.6	Basic bidi support	116
10.7	Local Language Configuration	119
11	Multiple languages (switch.def)	120
11.1	Selecting the language	121
11.2	Errors	129
12	Loading hyphenation patterns	131
13	Font handling with fontspec	136

14	Hooks for XeTeX and LuaTeX	140
14.1	XeTeX	140
14.2	Layout	142
14.3	LuaTeX	143
14.4	Southeast Asian scripts	149
14.5	CJK line breaking	151
14.6	Layout	153
14.7	Auto bidi with basic and basic-r	155
15	Data for CJK	165
16	The ‘nil’ language	165
17	Support for Plain T_EX (plain.def)	166
17.1	Not renaming hyphen.tex	166
17.2	Emulating some L ^A T _E X features	167
17.3	General tools	168
17.4	Encoding related macros	171
18	Acknowledgements	174

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	4
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	5
You are loading directly a language style	7
Unknown language ‘LANG’	8
Argument of \language@active@arg” has an extra }	11
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	25

Part I

User guide

- This user guide focuses on \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**. The most recent features could be still unstable. Please, report any issues you find in <https://github.com/latex3/babel/issues>, 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 list on <http://tug.org/mailman/listinfo/kadingira>. You can follow the development of babel in <https://github.com/latex3/babel> (which provides some sample files, too).
- See section 3.1 for contributing a language.
- The first sections describe the traditional way of loading a language (with `ldf` files). The alternative way based on `ini` files, which complements the previous one (it will *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.

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):

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

TROUBLESHOOTING A common source of trouble is a wrong setting of the input encoding. Very often you will get the following somewhat cryptic error:

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

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, T_EXLive, etc.) for further info about how to configure it.

1.2 Multilingual documents

In multilingual documents, just use several options. The last one is considered the main language, activated by default. Sometimes, the main language changes the document layout (eg, spanish and french).

EXAMPLE In L^AT_EX, the preamble of the document:

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

would tell L^AT_EX 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:

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

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 L^AT_EX ≥ 2018-04-01 if the encoding is UTF-8.

```
\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}
```

1.3 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 accept 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 is a more general mechanism.

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

1.4 xelatex and luatex

Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents.

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 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.

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

EXAMPLE Here is a simple monolingual document in Russian (text from the Wikipedia). 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).

```
\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}
```

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}.
```

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.

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

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.

```
{\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` `{\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).

1.8 Auxiliary language selectors

`\begin{otherlanguage}` `{\language} ... \end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\selectlanguage`, except the 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*}` `{\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}` `{\language} ... \end{hyphenrules}`

The environment `hyphenrules` can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’, provided that in `language.dat` the ‘language’ nohyphenation is defined by loading `zerohyph.tex`. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, `hyphenrules` is discouraged and `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<language1>\{<text>\}`, and `\begin<tag1>\}` to be `\begin{otherlanguage*}\{<language1>\}`, and so on. Note `\<tag1>` is also allowed, but remember to set it locally inside a group.

EXAMPLE With

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

you can write

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

and

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

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

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

`\babelensure` $[include=\langle commands \rangle, exclude=\langle commands \rangle, fontenc=\langle encoding \rangle]\{\langle language \rangle\}$

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, as 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`).

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 `*{\<shorthands-list>}`

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

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

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

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

\useshorthands `*{\<char>}`

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*{\<char>}` 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 `[\<language>,\<language>,...]{\<shorthand>}{\<code>}`

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{\<lang>}` to the corresponding `\extras{\<lang>}`, as explained below). By default, user shorthands are (re)defined.

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

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

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

However, 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:

⁵With it encoded string may not work as expected.

```
\defineshorthand[*polish,*portugese]{"-"}{\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.

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

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character / over " in typing Polish texts, this can be achieved by entering `\aliasshorthand{/}{/}`.

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

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

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

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

\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 with, for example, `\usesshorthands`.)

Very often, this is a more convenient way to deactivate shorthands than `\shorthandoff`, as 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 you own user shorthands provided they do not overlap.)

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

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

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.

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.

⁷Thanks to Enrico Gregorio

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

activegrave Same for `.

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

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

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

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

safe= none | ref | bib

Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions – of course, in such a case you cannot use shorthands in these macros, but this is not a real problem (just use “allowed” characters).

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 any more.

config= $\langle file \rangle$

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

main= $\langle language \rangle$

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

headfoot= $\langle language \rangle$

By default, headlines and footlines are not touched (only marks), and if they contain language dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.

noconfigs Global and language default config files are not loaded, so you can make sure your document is not spoiled by an unexpected .cfg file. However, if the key config is set, this file is loaded.

showlanguages Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.

nocase	New 3.9l Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code>) are ignored. Use only if there are incompatibilities with other packages.
silent	New 3.9l No warnings and no <i>infos</i> are written to the log file. ⁹
strings=	generic unicode encoded $\langle label \rangle$ $\langle font\ encoding \rangle$ Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional \TeX , LICR and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in <code>\MakeUppercase</code> and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).
hyphenmap=	off main select other other* 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=	default basic basic-r bidi-l bidi-r New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.21.
layout=	New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.21.

1.12 The base option

With this package option `babel` just loads some basic macros (those in `switch.def`), defines `\AfterBabelLanguage` and exits. It also selects the hyphenations 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 hyphenations patterns of a single language, too.

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

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 paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

This command is currently the only provided by base. Executes `<code>` 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 `<option-name>` 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}
```

1.13 ini files

An alternative approach to define a language is by means of an `ini` file. Currently `babel` provides about 200 of these files containing the basic data required for a language. 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 development (in other words, `\babelprovide` is mainly intended for auxiliary tasks).

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

```
\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

NOTE The `ini` files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follows:

Arabic Monolingual documents mostly work in luatex, but it must be fine tuned, and a recent version of fontspec/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 seems better, but still problematic).

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

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

Other Indic scripts are still under development in luatex. On the other hand, xetex is better.

Southeast scripts Thai works in both luatex and xetex, but line breaking differs (rules can be modified in luatex; they are hardcoded in xetex). Lao seems to work, too, but there are no patterns for the latter in luatex. Some quick patterns could help, with something similar to:

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{ໂ ນ າ ັ ັ ັ ັ ັ ັ ັ ັ % Random
```

Khmer clusters are rendered wrongly.

East Asia scripts Internal inconsistencies in script and language names must be sorted out, so you may need to set them explicitly in \babel font, as well as CJKShape. luatex does basic line breaking, but currently xetex does not (you may load zhspacing). 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...), . Actually, this is what the ldf does in japanese with luatex, because the following piece of code loads luatexja:

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

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

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

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

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

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

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

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

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

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

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

1.14 Selecting fonts

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

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

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. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, `*devanagari`).

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.

```
\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:

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

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

`\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:

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

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

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

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

This makes sure the OpenType script for Devanagari is `deva` and not `dev2` (luatex does not detect automatically the correct script¹⁴). 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 (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 are 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 Do not use `\setxxxxfont` and `\babelfont` at the same time. `\babelfont` follows the standard \LaTeX conventions to set the basic families – define `\xxdefault`, and activate it with `\xxfamily`. On the other hand, `\setxxxxfont` in `fontspec` takes a different approach, because `\xxfamily` is redefined with the family name hardcoded (so that `\xxdefault` becomes no-op). Of course, both methods are incompatible, and if you use `\setxxxxfont`, font switching with `\babelfont` just does *not* work (nor the standard `\xxdefault`, for that matter).

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

¹⁴And even with the correct code some fonts could be rendered incorrectly by `fontspec`, so double check the results. `xetex` fares better, but some font are still problematic.

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 it.

- 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\extrarussian{\mymacro}
```

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

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

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

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

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

1.16 Creating a language

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

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

Defines the internal structure of the language with some defaults: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3, but captions and date are not defined. Conveniently, babel warns you about what to do. Very likely you will find alerts like that in the log file:

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

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

EXAMPLE If you need a language named arhinish:

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

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 (hindi, french, breton, and occitan).

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

captions= \langle language-tag \rangle

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. Only in newly defined languages.

script= \langle script-name \rangle

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

language= \langle language-name \rangle

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

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

mapfont= direction

Assigns the font for the writing direction of this language (only with `bidi=basic`).¹⁵ 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.¹⁶ So, there should be at most 3 directives of this kind.

¹⁵There will be another value, language, not yet implemented.

¹⁶In future releases an new value (script) will be added.

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. Requires `import`.

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). Requires `import`.

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

1.17 Digits

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 *ar, as, bn, bo, brx, ckb, dz, fa, gu, hi, km, kn, kok, ks, lo, lrc, ml, mr, my, mzn, ne, or, pa, ps, ta, te, th, ug, ur, uz, vai, yue, zh*.

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

1.18 Getting the current language name

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

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

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

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

WARNING The advice about `\language` also applies here – use `iflang` instead of `\iflanguage` if possible.

1.19 Hyphenation and line breaking

`\babelhyphen` $\ast\{\langle type \rangle\}$

`\babelhyphen` $\ast\{\langle text \rangle\}$

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, Portugese, 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 provide 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, Portugese and Spanish.
- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).
- `\babelhyphen{empty}` inserts a break oportunity without a hyphen at all.
- `\babelhyphen{\langle text \rangle}` is a hard “hyphen” using $\langle text \rangle$ instead. A typical case is `\babelhyphen{/}`.

With all of them hyphenation in the rest of the word is enabled. If you don’t want enabling 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 \TeX 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 `\babelexhyphen`; (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 pattern for the language, you can add at least some typical cases.

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

New 3.9m *In luatex only*,¹⁷ 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).

1.20 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either \fontencoding (low level) or a language name (high level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.¹⁸

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

¹⁷With 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.

¹⁸The 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.

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.¹⁹

`\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 cases. No assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

1.21 Selecting directions

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

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

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.)

¹⁹But still defined for backwards compatibility.

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.

New 3.32 There is some experimental support for `harftex`. Since it is based on `luatex`, the option `basic` mostly works. You may need to deactivate the `rtlm` or the `rtla` font features (besides loading `harfload` before `babel` and activating `mode=harf`; there is a sample in the GitHub repository).

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-r` is available in `luatex` only.

```
\documentclass{article}

\usepackage[bidi=basic-r]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[mapfont=direction]{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 `mapfont=direction`, 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` (as 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 `\texthe` must be defined to select the main language):

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

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

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

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

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection{<subsection>.<section>}`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it 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.²⁰

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 `\par shape` 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 options does exactly is also explained there).

²⁰Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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 `\LaTeX2e` **New 3.19** .

EXAMPLE Typically, in an Arabic document you would need:

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

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

Digits in `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 \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 `r l` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

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

Mainly for bidi text, but it 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-language \rangle}{\langle before \rangle}{\langle after \rangle}`

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.22 Language attributes

`\languageattribute`

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

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

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

1.23 Hooks

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

`\AddBabelHook`

```
{\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`).

Current events are the following; in some of them you can use one to three $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ parameters (`#1`, `#2`, `#3`), with the meaning given:

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the the above, if necessary.

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

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

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

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

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

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

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

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

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

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

loadkernel (file) By default loads `switch.def`. It can be used to load a different version of this files or to load nothing.

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

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

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

1.24 Languages supported by babel with ldf files

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

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

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

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

```

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

```

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

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

1.25 Unicode character properties in luatex

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

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

New 3.32 Here, $\{ \langle \text{char-code} \rangle \}$ 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. For example:

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

This command is allowed only in vertical mode (the preamble or between paragraphs).

1.26 Tips, workarounds, know 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.²¹ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of 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.

1.27 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.²² But that is the easy part, because they don't require modifying the L^AT_EX internals.

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

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

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

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

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).

1.28 Tentative and experimental code

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

Old stuff

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

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

So, for example:

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

2 Loading languages with language.dat

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

New 3.9q With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).²³ Until 3.9n, this task was delegated to the package `luatex-hyphen`, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).²⁴

2.1 Format

In that file the person who maintains a T_EX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored²⁵. When hyphenation

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

²⁴The loader for lua(e)tex is slightly different as it's not based on babel but on `etex.src`. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the babel way, i.e., with `language.dat`.

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

exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.²⁶ For example:

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

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

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

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\langle lang \rangle hyphenmins`, `\captions⟨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.²⁷
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

3.1 Guidelines for contributed languages

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

²⁷But not removed, for backward compatibility.

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

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

3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

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

`\bbl@declare@ttribute` This is a command to be used in the language definition files for declaring a language

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

3.3 Skeleton

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

```

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

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

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

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

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

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

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

```

```

% More strings

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

`\bbl@activate`

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

`\bbl@deactivate`

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

`\declare@shorthand`

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

`\bbl@add@special`

`\bbl@remove@special`

The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [2, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

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

`\babel@save` To save the current meaning of any control sequence, the macro `\babel@save` is provided. It takes one argument, $\langle csname \rangle$, 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 $\langle variable \rangle$.

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

3.7 Macros common to a number of languages

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

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

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

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

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

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

3.8 Encoding-dependent strings

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

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

²⁸This mechanism was introduced by Bernd Raichle.

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

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

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

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

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.²⁹ It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

²⁹In future releases further categories may be added.

```

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

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

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

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


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

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

\EndBabelCommands

```

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.³⁰

\EndBabelCommands Marks the end of the series of blocks.

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

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

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

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

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

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

\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
{\uccode`i=`I\relax
 \uccode`1=`I\relax}
{\lccode`I=`i\relax
 \lccode`I=`1\relax}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

There are three helper macros to be used inside `\SetHyphenMap`:

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

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

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

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

4 Changes

4.1 Changes in babel version 3.9

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

hyphen.cfg is the file to be used when generating the formats to load hyphenation patterns. By default it also loads `switch.def`.

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.32>>
2 <<date=2019/06/03>>
```

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

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

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

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

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

`\bbl@afterelse` `\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`-statement³¹. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

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

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

`\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.

```

28 \def\bbl@tempa#1{%
29   \long\def\bbl@trim##1##2{%
30     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
31   \def\bbl@trim@c{%
32     \ifx\bbl@trim@a\@sptoken
33       \expandafter\bbl@trim@b
34     \else
35       \expandafter\bbl@trim@b\expandafter#1%
36     \fi}%
37   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
38 \bbl@tempa{ }
39 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
40 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}

```

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

```

41 \def\bbl@ifunset#1{%
42   \expandafter\ifx\csname#1\endcsname\relax
43     \expandafter\@firstoftwo
44   \else
45     \expandafter\@secondoftwo
46   \fi}
47 \bbl@ifunset{ifcsname}%
48 {}%
49 {\def\bbl@ifunset#1{%
50   \ifcsname#1\endcsname
51     \expandafter\ifx\csname#1\endcsname\relax
52       \bbl@afterelse\expandafter\@firstoftwo
53     \else
54       \bbl@afterfi\expandafter\@secondoftwo
55     \fi
56   \else
57     \expandafter\@firstoftwo
58   \fi}}

```

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

```

59 \def\bbl@ifblank#1{%
60   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
61 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}

```

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

```

62 \def\bbl@forkv#1#2{%
63   \def\bbl@kvcmd##1##2##3{#2}%
64   \bbl@kvnext#1,\@nil,}
65 \def\bbl@kvnext#1,{%
66   \ifx\@nil#1\relax\else
67     \bbl@ifblank{#1}{ }\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
68   \expandafter\bbl@kvnext

```

```

69 \fi}
70 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
71 \bbl@trim@def\bbl@forkv@a{#1}%
72 \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}

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

73 \def\bbl@vforeach#1#2{%
74 \def\bbl@forcmd##1{#2}%
75 \bbl@fornext#1,\@nil,}
76 \def\bbl@fornext#1,{%
77 \ifx\@nil#1\relax\else
78 \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
79 \expandafter\bbl@fornext
80 \fi}
81 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

\bbl@replace

```

82 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
83 \toks@{}}%
84 \def\bbl@replace@aux##1#2##2#2{%
85 \ifx\bbl@nil##2%
86 \toks@\expandafter{\the\toks@##1}%
87 \else
88 \toks@\expandafter{\the\toks@##1#3}%
89 \bbl@afterfi
90 \bbl@replace@aux##2#2%
91 \fi}%
92 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
93 \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). It may change! (to add new features).

```

94 \expandafter\def\expandafter\bbl@parsedef\detokenize{macro:}#1->#2\relax{%
95 \def\bbl@tempa{#1}%
96 \def\bbl@tempb{#2}}
97 \def\bbl@sreplace#1#2#3{%
98 \begingroup
99 \expandafter\bbl@parsedef\meaning#1\relax
100 \def\bbl@tempc{#2}%
101 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
102 \def\bbl@tempd{#3}%
103 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
104 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@tempc}{\bbl@tempd}}%
105 \bbl@exp{%
106 \endgroup
107 \\\makeatletter % "internal" macros with @ are assumed
108 \\\scantokens{\def\#1\bbl@tempa{\bbl@tempb}}%
109 \catcode64=\the\catcode64\relax}} % Restore @

```

\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.

```

110 \def\bbl@exp#1{%
111 \begingroup

```



```

112 \let\\\noexpand
113 \def\<##1>\expandafter\noexpand\csname##1\endcsname}%
114 \edef\bbl@exp@aux{\endgroup#1}%
115 \bbl@exp@aux}

```

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 it in your language style if you want.

```

116 \def\bbl@ifsamestring#1#2{%
117 \begingroup
118 \protected@edef\bbl@tempb{#1}%
119 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
120 \protected@edef\bbl@tempc{#2}%
121 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
122 \ifx\bbl@tempb\bbl@tempc
123 \aftergroup\@firstoftwo
124 \else
125 \aftergroup\@secondoftwo
126 \fi
127 \endgroup}
128 \chardef\bbl@engine=%
129 \ifx\directlua\@undefined
130 \ifx\XeTeXinputencoding\@undefined
131 \z@
132 \else
133 \tw@
134 \fi
135 \else
136 \@ne
137 \fi
138 <</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 .

```

139 <<*Make sure ProvidesFile is defined>> ≡
140 \ifx\ProvidesFile\@undefined
141 \def\ProvidesFile#1[#2 #3 #4]{%
142 \wlog{File: #1 #4 #3 <#2>}%
143 \let\ProvidesFile\@undefined}
144 \fi
145 <</Make sure ProvidesFile is defined>>

```

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

```

146 <<*Load patterns in luatex>> ≡
147 \ifx\directlua\@undefined\else
148 \ifx\bbl@luapatterns\@undefined
149 \input luababel.def
150 \fi
151 \fi
152 <</Load patterns in luatex>>

```

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

```

153 <<*Load macros for plain if not LaTeX>> ≡
154 \ifx\AtBeginDocument\@undefined
155 \input plain.def\relax
156 \fi
157 <</Load macros for plain if not LaTeX>>

```

7.1 Multiple languages

<code>\language</code>	<p>Plain T_EX version 3.0 provides the primitive <code>\language</code> 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 <code>switch.def</code> and <code>hyphen.cfg</code>; the latter may seem redundant, but remember <code>babel</code> doesn't require loading <code>switch.def</code> in the format.</p> <pre> 158 <<*Define core switching macros>> ≡ 159 \ifx\language\@undefined 160 \csname newcount\endcsname\language 161 \fi 162 <</Define core switching macros>> </pre>
<code>\last@language</code>	<p>Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated.</p>
<code>\addlanguage</code>	<p>To add languages to T_EX's memory plain T_EX version 3.0 supplies <code>\newlanguage</code>, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original <code>\newlanguage</code> was defined to be <code>\outer</code>. For a format based on plain version 2.x, the definition of <code>\newlanguage</code> can not be copied because <code>\count 19</code> is used for other purposes in these formats. Therefore <code>\addlanguage</code> is defined using a definition based on the macros used to define <code>\newlanguage</code> in plain T_EX version 3.0.</p> <p>For formats based on plain version 3.0 the definition of <code>\newlanguage</code> can be simply copied, removing <code>\outer</code>. Plain T_EX version 3.0 uses <code>\count 19</code> for this purpose.</p> <pre> 163 <<*Define core switching macros>> ≡ 164 \ifx\newlanguage\@undefined 165 \csname newcount\endcsname\last@language 166 \def\addlanguage#1{% 167 \global\advance\last@language\@ne 168 \ifnum\last@language<\@ccclvi 169 \else 170 \errmessage{No room for a new \string\language!}% 171 \fi 172 \global\chardef#1\last@language 173 \wlog{\string#1 = \string\language\the\last@language}} 174 \else 175 \countdef\last@language=19 176 \def\addlanguage{\alloc@9\language\chardef\@ccclvi} 177 \fi 178 <</Define core switching macros>> </pre>

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 L^AT_EX 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

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

8 The Package File (L^AT_EX, `babel.sty`)

In order to make use of the features of L^AT_EX 2_ε, the `babel` system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language

options whose name is different from that of the .ldf file (like variant spellings). It also takes care of a number of compatibility issues with other packages and defines a few additional package options.

Apart from all the language options below we also have a few options that influence the behavior of language definition files.

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

8.1 base

The first option to be processed is base, which sets the hyphenation patterns then resets `ver@babel.sty` so that \LaTeX forgets about the first loading. After `switch.def` has been loaded (above) and `\AfterBabelLanguage` defined, exits.

```

179 (*package)
180 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
181 \ProvidesPackage{babel}[\langle date \rangle \langle version \rangle The Babel package]
182 \@ifpackagewith{babel}{debug}
183   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
184   \let\bbl@debug\@firstofone}
185   {\providecommand\bbl@trace[1]{}}%
186   \let\bbl@debug\gobble}
187 \ifx\bbl@switchflag@undefined % Prevent double input
188   \let\bbl@switchflag\relax
189   \input switch.def\relax
190 \fi
191 \langle Load patterns in luatex \rangle
192 \langle Basic macros \rangle
193 \def\AfterBabelLanguage#1{%
194   \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.

```

195 \ifx\bbl@languages\undefined\else
196   \begingroup
197     \catcode`\^^I=12
198     \@ifpackagewith{babel}{showlanguages}{%
199       \begingroup
200         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
201         \wlog{<*languages>}%
202         \bbl@languages
203         \wlog{</languages>}%
204       \endgroup}{%
205     \endgroup
206     \def\bbl@elt#1#2#3#4{%
207       \ifnum#2=\z@
208         \gdef\bbl@nulllanguage{#1}%
209         \def\bbl@elt##1##2##3##4{}%
210       \fi}%
211     \bbl@languages
212 \fi
213 \ifodd\bbl@engine
214   % Harftex is evolving, so the callback is not hardcoded, just in case
215   \def\bbl@harfpreamble{Harf pre_linebreak_filter callback}%
216   \def\bbl@activate@preotf{%
217     \let\bbl@activate@preotf\relax % only once
218     \directlua{

```

```

219     Babel = Babel or {}
220     %
221     function Babel.pre_otfload_v(head)
222         if Babel.numbers and Babel.digits_mapped then
223             head = Babel.numbers(head)
224         end
225         if Babel.bidi_enabled then
226             head = Babel.bidi(head, false, dir)
227         end
228         return head
229     end
230     %
231     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
232         if Babel.numbers and Babel.digits_mapped then
233             head = Babel.numbers(head)
234         end
235         if Babel.fixboxdirs then          % Temporary!
236             head = Babel.fixboxdirs(head)
237         end
238         if Babel.bidi_enabled then
239             head = Babel.bidi(head, false, dir)
240         end
241         return head
242     end
243     %
244     luatexbase.add_to_callback('pre_linebreak_filter',
245         Babel.pre_otfload_v,
246         'Babel.pre_otfload_v',
247         luatexbase.priority_in_callback('pre_linebreak_filter',
248             '\bbl@harfpreamble')
249         or luatexbase.priority_in_callback('pre_linebreak_filter',
250             'luaotfload.node_processor')
251         or nil)
252     %
253     luatexbase.add_to_callback('hpack_filter',
254         Babel.pre_otfload_h,
255         'Babel.pre_otfload_h',
256         luatexbase.priority_in_callback('hpack_filter',
257             '\bbl@harfpreamble')
258         or luatexbase.priority_in_callback('hpack_filter',
259             'luaotfload.node_processor')
260         or nil)
261     }}
262     \let\bbl@tempa\relax
263     \@ifpackagewith{babel}{bidi=basic}%
264     {\def\bbl@tempa{basic}}%
265     {\@ifpackagewith{babel}{bidi=basic-r}%
266     {\def\bbl@tempa{basic-r}}%
267     {}}
268     \ifx\bbl@tempa\relax\else
269         \let\bbl@beforeforeign\leavevmode
270         \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
271         \RequirePackage{luatexbase}%
272         \directlua{
273             require('babel-data-bidi.lua')
274             require('babel-bidi-\bbl@tempa.lua')
275         }
276         \bbl@activate@preotf
277     \fi

```

```
278 \fi
```

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

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

8.2 key=value options and other general option

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

```
294 \bbl@trace{key=value and another general options}
295 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
296 \def\bbl@tempb#1.#2{%
297   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
298 \def\bbl@tempd#1.#2\@nnil{%
299   \ifx\@empty#2%
300     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
301   \else
302     \in@{=}{#1}\ifin@
303     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
304   \else
305     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
306     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
307   \fi
308   \fi}
309 \let\bbl@tempc\@empty
310 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
311 \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.

```
312 \DeclareOption{KeepShorthandsActive}{}
313 \DeclareOption{activeacute}{}
314 \DeclareOption{activegrave}{}
315 \DeclareOption{debug}{}
316 \DeclareOption{noconfigs}{}
317 \DeclareOption{showlanguages}{}
318 \DeclareOption{silent}{}%
```

```

319 \DeclareOption{mono}{}
320 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
321 <<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.

```

322 \let\bbl@opt@shorthands\@nnil
323 \let\bbl@opt@config\@nnil
324 \let\bbl@opt@main\@nnil
325 \let\bbl@opt@headfoot\@nnil
326 \let\bbl@opt@layout\@nnil

```

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

```

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

```

337 \let\bbl@language@opts\@empty
338 \DeclareOption*{%
339   \bbl@xin@{\string=}{\CurrentOption}%
340   \ifin@
341     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
342   \else
343     \bbl@add@list\bbl@language@opts{\CurrentOption}%
344   \fi}

```

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

```

345 \ProcessOptions*

```

8.3 Conditional loading of shorthands

If there is no `shorthands=<chars>`, the original babel macros are left untouched, but if there is, these macros are wrapped (in `babel.def`) to define only those given. A bit of optimization: if there is no `shorthands=`, then `\bbl@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

346 \bbl@trace{Conditional loading of shorthands}
347 \def\bbl@sh@string#1{%
348   \ifx#1\@empty\else
349     \ifx#1t\string~%
350     \else\ifx#1c\string,%
351     \else\string#1%
352   \fi\fi

```

```

353 \expandafter\bb1@sh@string
354 \fi}
355 \ifx\bb1@opt@shorthands\@nnil
356 \def\bb1@ifshorthand#1#2#3{#2}%
357 \else\ifx\bb1@opt@shorthands\@empty
358 \def\bb1@ifshorthand#1#2#3{#3}%
359 \else

```

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

```

360 \def\bb1@ifshorthand#1{%
361 \bb1@xin@{\string#1}{\bb1@opt@shorthands}%
362 \ifin@
363 \expandafter\@firstoftwo
364 \else
365 \expandafter\@secondoftwo
366 \fi}

```

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

```

367 \edef\bb1@opt@shorthands{%
368 \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.

```

369 \bb1@ifshorthand{'}%
370 {\PassOptionsToPackage{activeacute}{babel}}{}
371 \bb1@ifshorthand{'}%
372 {\PassOptionsToPackage{activegrave}{babel}}{}
373 \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.

```

374 \ifx\bb1@opt@headfoot\@nnil\else
375 \g@addto@macro\@resetactivechars{%
376 \set@typeset@protect
377 \expandafter\select@language@x\expandafter{\bb1@opt@headfoot}%
378 \let\protect\noexpand}
379 \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.

```

380 \ifx\bb1@opt@safe\@undefined
381 \def\bb1@opt@safe{BR}
382 \fi
383 \ifx\bb1@opt@main\@nnil\else
384 \edef\bb1@language@opts{%
385 \ifx\bb1@language@opts\@empty\else\bb1@language@opts,\fi
386 \bb1@opt@main}
387 \fi

```

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

```

388 \bb1@trace{Defining IfBabelLayout}
389 \ifx\bb1@opt@layout\@nnil
390 \newcommand\IfBabelLayout[3]{#3}%
391 \else
392 \newcommand\IfBabelLayout[1]{%
393 \@expandtwoargs\in@{.#1.}{.\bb1@opt@layout.}%
394 \ifin@

```

```

395     \expandafter\@firstoftwo
396     \else
397     \expandafter\@secondoftwo
398     \fi}
399 \fi

```

8.4 Language options

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

```

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

```

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

```

418 \def\bbl@try@load@lang#1#2#3{%
419   \IfFileExists{\CurrentOption.ldf}%
420   {\bbl@load@language{\CurrentOption}}%
421   {#1\bbl@load@language{#2#3}}
422 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}}{}
423 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}}{}
424 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}}{}
425 \DeclareOption{hebrew}{%
426   \input{rlbabel.def}%
427   \bbl@load@language{hebrew}}
428 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}{}
429 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}{}
430 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}}{}
431 \DeclareOption{polutonikogreek}{%
432   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
433 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}}{}
434 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}{}
435 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}}{}
436 \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.


```

437 \ifx\bbl@opt@config\@nnil
438 \ifpackagewith{babel}{noconfigs}{}%
439 {\InputIfFileExists{bblopts.cfg}%
440  {\typeout{*****^J%
441           * Local config file bblopts.cfg used^^J%
442           *}}}%
443  {}}%
444 \else
445 \InputIfFileExists{\bbl@opt@config.cfg}%
446 {\typeout{*****^J%
447           * Local config file \bbl@opt@config.cfg used^^J%
448           *}}%
449 {\bbl@error{%
450   Local config file '\bbl@opt@config.cfg' not found}{%
451   Perhaps you misspelled it.}}%
452 \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 name of the option and the file are the same.

```

453 \bbl@for\bbl@tempa\bbl@language@opts{%
454   \bbl@ifunset{ds@\bbl@tempa}%
455   {\edef\bbl@tempb{%
456     \noexpand\DeclareOption
457     {\bbl@tempa}%
458     {\noexpand\bbl@load@language{\bbl@tempa}}}%
459    \bbl@tempb}%
460   \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.

```

461 \bbl@foreach\@classoptionslist{%
462   \bbl@ifunset{ds@#1}%
463   {\IfFileExists{#1.ldf}%
464    {\DeclareOption{#1}{\bbl@load@language{#1}}}%
465    {}}%
466   {}}

```

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

```

467 \ifx\bbl@opt@main\@nnil\else
468   \expandafter
469   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
470   \DeclareOption{\bbl@opt@main}{}
471 \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):

```

472 \def\AfterBabelLanguage#1{%
473   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
474 \DeclareOption*{}
475 \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`.

```

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

```

502 \ifx\bbl@main@language\undefined
503   \bbl@info{%
504     You haven't specified a language. I'll use 'nil'\%
505     as the main language. Reported}
506   \bbl@load@language{nil}
507 \fi
508 \</package>
509 \<core>

```

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

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `babel.def` contains most of the code, while `switch.def` defines the language switching commands; both can be read at run time. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns (by default, it also inputs `switch.def`, for “historical reasons”, but it is not necessary). When `babel.def` is loaded it checks if the current version of `switch.def` is in the format; if not, it is loaded. A further file, `babel.sty`, contains \LaTeX -specific stuff. Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

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

9.1 Tools

```

510 \ifx\ldf@quit\@undefined
511 \else
512   \expandafter\endinput
513 \fi
514 <<Make sure ProvidesFile is defined>>
515 \ProvidesFile{babel.def}[\<date>] \<version>] Babel common definitions]
516 <<Load macros for plain if not LaTeX>>

```

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 in no package options, and therefore and 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).

```

517 \ifx\bbl@ifshorthand\@undefined
518   \let\bbl@opt@shorthands\@nnil
519   \def\bbl@ifshorthand#1#2#3{#2}%
520   \let\bbl@language@opts\@empty
521   \ifx\babeloptionstrings\@undefined
522     \let\bbl@opt@strings\@nnil
523   \else
524     \let\bbl@opt@strings\babeloptionstrings
525   \fi
526   \def\BabelStringsDefault{generic}
527   \def\bbl@tempa{normal}
528   \ifx\babeloptionmath\bbl@tempa
529     \def\bbl@mathnormal{\noexpand\textormath}
530   \fi
531   \def\AfterBabelLanguage#1#2{}
532   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
533   \let\bbl@afterlang\relax
534   \def\bbl@opt@safe{BR}
535   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
536   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
537 \fi

```

And continue.

```

538 \ifx\bbl@switchflag\@undefined % Prevent double input
539   \let\bbl@switchflag\relax
540   \input switch.def\relax
541 \fi
542 \bbl@trace{Compatibility with language.def}
543 \ifx\bbl@languages\@undefined
544   \ifx\directlua\@undefined
545     \openin1 = language.def
546     \ifeof1
547       \closein1
548       \message{I couldn't find the file language.def}
549     \else
550       \closein1
551       \begingroup
552         \def\addlanguage#1#2#3#4#5{%
553           \expandafter\ifx\csname lang@#1\endcsname\relax\else

```

```

554      \global\expandafter\let\csname l@#1\expandafter\endcsname
555      \csname lang@#1\endcsname
556      \fi}%
557      \def\uselanguage#1{%
558      \input language.def
559      \endgroup
560      \fi
561      \fi
562      \chardef\l@english\z@
563      \fi
564      <<Load patterns in luatex>>
565      <<Basic macros>>

```

`\addto` For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro `\addto` is introduced. It takes two arguments, a *<control sequence>* and \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. Otherwise the replacement text for the *<control sequence>* is expanded and stored in a token register, together with the \TeX -code to be added. Finally the *<control sequence>* is redefined, using the contents of the token register.

```

566 \def\addto#1#2{%
567   \ifx#1\@undefined
568     \def#1{#2}%
569   \else
570     \ifx#1\relax
571       \def#1{#2}%
572     \else
573       {\toks@\expandafter{#1#2}%
574       \xdef#1{\the\toks@}}%
575     \fi
576   \fi}

```

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

```

577 \def\bbl@withactive#1#2{%
578   \begingroup
579   \lccode`~=#2\relax
580   \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).

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

```

581 \def\bbl@redefine#1{%
582   \edef\bbl@tempa{\bbl@stripslash#1}%
583   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
584   \expandafter\def\csname\bbl@tempa\endcsname}

```

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

```
585 \@onlypreamble\bbl@redefine
```

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

```

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

```

591 \def\bbl@redefineroobust#1{%
592   \edef\bbl@tempa{\bbl@stripslash#1}%
593   \bbl@ifunset{\bbl@tempa\space}%
594   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
595     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
596   {\bbl@exp{\let\<org@\bbl@tempa\>\<\bbl@tempa\space>}}}%
597   \@namedef{\bbl@tempa\space}}

```

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

```

598 \@onlypreamble\bbl@redefineroobust

```

9.2 Hooks

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

```

599 \bbl@trace{Hooks}
600 \def\AddBabelHook#1#2{%
601   \bbl@ifunset{\bbl@hk@#1}{\EnableBabelHook{#1}}}%
602   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}}%
603   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
604   \bbl@ifunset{\bbl@ev@#1@#2}%
605   {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}}%
606   \bbl@csarg\newcommand}%
607   {\bbl@csarg\let{ev@#1@#2}\relax
608   \bbl@csarg\newcommand}%
609   {ev@#1@#2}{\bbl@tempb}}
610 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
611 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
612 \def\bbl@usehooks#1#2{%
613   \def\bbl@elt##1{%
614     \@nameuse{\bbl@hk@##1}{\@nameuse{\bbl@ev@##1@#1}#2}}%
615   \@nameuse{\bbl@ev@#1}}

```

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).

```

616 \def\bbl@evargs{,% <- don't delete this comma
617   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
618   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
619   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
620   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0}

```

`\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.

```

621 \bbl@trace{Defining babelensure}
622 \newcommand\babelensure[2][{}]{% TODO - revise test files
623   \AddBabelHook{babel-ensure}{afterextras}{%
624     \ifcase\bbl@select@type
625       \@nameuse{\bbl@e@\language}\fi}%
626   \fi}%
627 \begingroup
628   \let\bbl@ens@include\@empty
629   \let\bbl@ens@exclude\@empty
630   \def\bbl@ens@fontenc{\relax}%
631   \def\bbl@tempb##1{%
632     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
633   \edef\bbl@tempa{\bbl@tempb#1\@empty}%
634   \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
635   \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
636   \def\bbl@tempc{\bbl@ensure}%
637   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
638     \expandafter{\bbl@ens@include}}%
639   \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
640     \expandafter{\bbl@ens@exclude}}%
641   \toks@\expandafter{\bbl@tempc}%
642   \bbl@exp{%
643     \endgroup
644     \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
645 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
646   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
647     \ifx##1\@undefined % 3.32 - Don't assume the macros exists
648       \edef##1{\noexpand\bbl@nocaption
649         {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
650       \fi
651       \ifx##1\@empty\else
652         \in@{##1}{#2}%
653         \ifin@ \else
654           \bbl@ifunset{\bbl@ensure@\language}%
655             {\bbl@exp{%
656               \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
657                 \\\foreignlanguage{\language}%
658                 {\ifx\relax#3\else
659                   \\\fontencoding{#3}\selectfont
660                   \fi
661                 #####1}}}%
662             }%
663           \toks@\expandafter{##1}%
664           \edef##1{%
665             \bbl@csarg\noexpand{ensure@\language}%
666             {\the\toks@}}%
667           \fi
668           \expandafter\bbl@tempb
669           \fi}%

```

```

670 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
671 \def\bb1@tempa##1{% elt for include list
672   \ifx##1\@empty\else
673     \bb1@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
674     \ifin@else
675       \bb1@tempb##1\@empty
676     \fi
677     \expandafter\bb1@tempa
678   \fi}%
679 \bb1@tempa#1\@empty}
680 \def\bb1@captionslist{%
681   \prefacename\refname\abstractname\bibname\chaptername\appendixname
682   \contentsname\listfigurename\listtablename\indexname\figurename
683   \tablename\partname\enc1name\ccname\headtoname\pagename\seename
684   \alsoname\proofname\glossaryname}

```

9.3 Setting up language files

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

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

Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions with the `\let` primitive. Therefore we store its current catcode and restore it later on. Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing #2 through string. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\undefined`.

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

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

Finally we check `\originalTeX`.

```

685 \bb1@trace{Macros for setting language files up}
686 \def\bb1@ldfinit{%
687   \let\bb1@screset\@empty
688   \let\BabelStrings\bb1@opt@string
689   \let\BabelOptions\@empty
690   \let\BabelLanguages\relax
691   \ifx\originalTeX\@undefined
692     \let\originalTeX\@empty
693   \else
694     \originalTeX
695   \fi}
696 \def\LdfInit#1#2{%
697   \chardef\atcatcode=\catcode`\@
698   \catcode`\@=11\relax
699   \chardef\eqcatcode=\catcode`\=
700   \catcode`\==12\relax
701   \expandafter\if\expandafter\@backslashchar
702     \expandafter\@car\string#2\@nil

```

```

703 \ifx#2\@undefined\else
704 \ldf@quit{#1}%
705 \fi
706 \else
707 \expandafter\ifx\csname#2\endcsname\relax\else
708 \ldf@quit{#1}%
709 \fi
710 \fi
711 \bbl@ldfinit}

```

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

```

712 \def\ldf@quit#1{%
713 \expandafter\main@language\expandafter{#1}%
714 \catcode\@=\atcatcode \let\atcatcode\relax
715 \catcode\==\eqcatcode \let\eqcatcode\relax
716 \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.

```

717 \def\bbl@afterldf#1{%
718 \bbl@afterlang
719 \let\bbl@afterlang\relax
720 \let\BabelModifiers\relax
721 \let\bbl@screset\relax}%
722 \def\ldf@finish#1{%
723 \loadlocalcfg{#1}%
724 \bbl@afterldf{#1}%
725 \expandafter\main@language\expandafter{#1}%
726 \catcode\@=\atcatcode \let\atcatcode\relax
727 \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 .

```

728 \@onlypreamble\LdfInit
729 \@onlypreamble\ldf@quit
730 \@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.

```

731 \def\main@language#1{%
732 \def\bbl@main@language{#1}%
733 \let\language\name\bbl@main@language
734 \bbl@id@assign
735 \chardef\localeid\@nameuse{\bbl@id@\language}%
736 \bbl@patterns{\language}}

```

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

```

737 \AtBeginDocument{%
738 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
739 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```


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

```

740 \def\select@language@x#1{%
741   \ifcase\bbbl@select@type
742     \bbbl@ifsamestring\languagename{#1}{\select@language{#1}}%
743   \else
744     \select@language{#1}%
745   \fi}

```

9.4 Shorthands

`\bbbl@add@special` The macro `\bbbl@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.

```

746 \bbbl@trace{Shorhands}
747 \def\bbbl@add@special#1{% 1:a macro like \", \?, etc.
748   \bbbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
749   \bbbl@ifunset{@sanitize}{\bbbl@add\@sanitize{\@makeother#1}}%
750   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
751     \begingroup
752       \catcode`#1\active
753       \nfss@catcodes
754       \ifnum\catcode`#1=\active
755         \endgroup
756         \bbbl@add\nfss@catcodes{\@makeother#1}%
757       \else
758         \endgroup
759     \fi
760   \fi}

```

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

```

761 \def\bbbl@remove@special#1{%
762   \begingroup
763     \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
764       \else\noexpand##1\noexpand##2\fi}%
765     \def\do{\x\do}%
766     \def\@makeother{\x\@makeother}%
767   \edef\x{\endgroup
768     \def\noexpand\dospecials{\dospecials}%
769     \expandafter\ifx\csname @sanitize\endcsname\relax\else
770       \def\noexpand\@sanitize{\@sanitize}%
771     \fi}%
772   \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 `\bbbl@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).

```

773 \def\bbl@active@def#1#2#3#4{%
774   \namedef{#3#1}{%
775     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
776       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
777     \else
778       \bbl@afterfi\csname#2@sh@#1\endcsname
779     \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.

```

780 \long\@namedef{#3@arg#1}##1{%
781   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
782     \bbl@afterelse\csname#4#1\endcsname##1%
783   \else
784     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
785   \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.

```

786 \def\@initiate@active@char#1#2#3{%
787   \bbl@ifunset{active@char\string#1}%
788   {\bbl@withactive
789     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
790   {}

```

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`).

```

791 \def\@initiate@active@char#1#2#3{%
792   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
793   \ifx#1\@undefined
794     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
795   \else
796     \bbl@csarg\let{oridef@#2}#1%
797     \bbl@csarg\edef{oridef@#2}{%
798       \let\noexpand#1%
799       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
800   \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*).

```

801 \ifx#1#3\relax
802   \expandafter\let\csname normal@char#2\endcsname#3%
803 \else
804   \bbl@info{Making #2 an active character}%
805   \ifnum\mathcode`#2="8000
806     \@namedef{normal@char#2}{%
807       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
808   \else
809     \@namedef{normal@char#2}{#3}%
810   \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).

```

811   \bbl@restoreactive{#2}%
812   \AtBeginDocument{%
813     \catcode`#2\active
814     \if@filesw
815       \immediate\write\@mainaux{\catcode`\string#2\active}%
816     \fi}%
817   \expandafter\bbl@add@special\csname#2\endcsname
818   \catcode`#2\active
819 \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}`).

```

820 \let\bbl@tempa\@firstoftwo
821 \if\string^#2%
822   \def\bbl@tempa{\noexpand\textormath}%
823 \else
824   \ifx\bbl@mathnormal\@undefined\else
825     \let\bbl@tempa\bbl@mathnormal
826   \fi
827 \fi
828 \expandafter\edef\csname active@char#2\endcsname{%
829   \bbl@tempa
830     {\noexpand\if@safe@actives
831       \noexpand\expandafter
832       \expandafter\noexpand\csname normal@char#2\endcsname
833     \noexpand\else
834       \noexpand\expandafter
835       \expandafter\noexpand\csname bbl@doactive#2\endcsname
836     \noexpand\fi}%
837   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
838 \bbl@csarg\edef{doactive#2}{%
839   \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!).

```
840 \bbl@csarg\edef{active@#2}{%
841   \noexpand\active@prefix\noexpand#1%
842   \expandafter\noexpand\csname active@char#2\endcsname}%
843 \bbl@csarg\edef{normal@#2}{%
844   \noexpand\active@prefix\noexpand#1%
845   \expandafter\noexpand\csname normal@char#2\endcsname}%
846 \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.

```
847 \bbl@active@def#2\user@group{user@active}{language@active}%
848 \bbl@active@def#2\language@group{language@active}{system@active}%
849 \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.

```
850 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
851   {\expandafter\noexpand\csname normal@char#2\endcsname}%
852 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
853   {\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.

```
854 \if\string'#2%
855   \let\prim@s\bbl@prim@s
856   \let\active@math@prime#1%
857 \fi
858 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
859 <<(*More package options)>> ≡
860 \DeclareOption{math=active}{}
861 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
862 <</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*.

```
863 \@ifpackagewith{babel}{KeepShorthandsActive}%
864   {\let\bbl@restoreactive\@gobble}%
865   {\def\bbl@restoreactive#1{%
866     \bbl@exp{%
867       \\\AfterBabelLanguage\\CurrentOption
868       {\catcode`#1=\the\catcode`#1\relax}%
869       \\\AtEndOfPackage
870       {\catcode`#1=\the\catcode`#1\relax}}}%
871   \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.

```
872 \def\bbl@sh@select#1#2{%
873   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
874     \bbl@afterelse\bbl@scndcs
875   \else
876     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
877   \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`.

```
878 \def\active@prefix#1{%
879   \ifx\protect\@typeset@protect
880     \else
```

When `\protect` is set to `\@unexpandable@protect` we make sure that the active character is also *not* expanded by inserting `\noexpand` in front of it. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with).

```
881   \ifx\protect\@unexpandable@protect
882     \noexpand#1%
883   \else
884     \protect#1%
885   \fi
886   \expandafter\@gobble
887 \fi}
```

`\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⟩`.

```
888 \newif\if@safe@actives
889 \@safe@activesfalse
```

`\bbl@restore@actives` When the output routine kicks in while the active characters were made “safe” this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them “unsafe” again.

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

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

```
891 \def\bbl@activate#1{%
892   \bbl@withactive{\expandafter\let\expandafter}#1%
893   \csname bbl@active@\string#1\endcsname}
894 \def\bbl@deactivate#1{%
895   \bbl@withactive{\expandafter\let\expandafter}#1%
896   \csname bbl@normal@\string#1\endcsname}
```

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

```
897 \def\bbl@firstcs#1#2{\csname#1\endcsname}
898 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

`\declare@shorthand` The command `\declare@shorthand` is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. ~ or "a;
3. the code to be executed when the shorthand is encountered.

```

899 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
900 \def\@decl@short#1#2#3\@nil#4{%
901   \def\bbl@tempa{#3}%
902   \ifx\bbl@tempa\@empty
903     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
904     \bbl@ifunset{#1@sh@\string#2@}\}%
905     {\def\bbl@tempa{#4}%
906       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
907       \else
908         \bbl@info
909         {Redefining #1 shorthand \string#2\\%
910          in language \CurrentOption}%
911       \fi}%
912     \@namedef{#1@sh@\string#2@}{#4}%
913   \else
914     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
915     \bbl@ifunset{#1@sh@\string#2@\string#3@}\}%
916     {\def\bbl@tempa{#4}%
917       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
918       \else
919         \bbl@info
920         {Redefining #1 shorthand \string#2\string#3\\%
921          in language \CurrentOption}%
922       \fi}%
923     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
924   \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.

```

925 \def\textormath{%
926   \ifmmode
927     \expandafter\@secondoftwo
928   \else
929     \expandafter\@firstoftwo
930   \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’.

```

931 \def\user@group{user}
932 \def\language@group{english}
933 \def\system@group{system}

```

`\useshorthands` This is the user level command to tell \TeX that user level shorthands will be used in the document. It takes one argument, the character that starts a shorthand. First note that this is user level, and then initialize and activate the character for use as a shorthand character (ie, it’s active in the preamble). Languages can deactivate shorthands, so a starred version is also provided which activates them always after the language has been switched.

```

934 \def\usesshorthands{%
935   \ifstar\bb1@usessh@s{\bb1@usessh@x{}}
936 \def\bb1@usessh@s#1{%
937   \bb1@usessh@x
938   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
939   {#1}}
940 \def\bb1@usessh@x#1#2{%
941   \bb1@ifshorthand{#2}%
942   {\def\user@group{user}%
943     \initiate@active@char{#2}%
944     #1%
945     \bb1@activate{#2}}%
946   {\bb1@error
947     {Cannot declare a shorthand turned off (\string#2)}
948     {Sorry, but you cannot use shorthands which have been\%
949       turned off in the package options}}}

```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user@<lang>` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bb1@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

950 \def\user@language@group{user@\language@group}
951 \def\bb1@set@user@generic#1#2{%
952   \bb1@ifunset{user@generic@active#1}%
953   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
954     \bb1@active@def#1\user@group{user@generic@active}{language@active}%
955     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
956       \expandafter\noexpand\csname normal@char#1\endcsname}%
957     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
958       \expandafter\noexpand\csname user@active#1\endcsname}}%
959   \@empty}
960 \newcommand\defineshorthand[3][user]{%
961   \edef\bb1@tempa{\zap@space#1 \@empty}%
962   \bb1@for\bb1@tempb\bb1@tempa{%
963     \if*\expandafter\@car\bb1@tempb\@nil
964       \edef\bb1@tempb{user@\expandafter\@gobble\bb1@tempb}%
965       \@expandtwoargs
966       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
967     \fi
968     \declare@shorthand{\bb1@tempb}{#2}{#3}}}

```

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

```

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

```

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

```

970 \def\aliasshorthand#1#2{%
971   \bb1@ifshorthand{#2}%
972   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
973     \ifx\document\@notprerr
974       \@notshorthand{#2}%
975     \else
976       \initiate@active@char{#2}%

```

Then, we define the new shorthand in terms of the original one, but note with `\aliasshorthands{"}{/}` is `\active@prefix / \active@char/`, so we still need to let the

littest to \active@char".

```

977      \expandafter\let\csname active@char\string#2\expandafter\endcsname
978      \csname active@char\string#1\endcsname
979      \expandafter\let\csname normal@char\string#2\expandafter\endcsname
980      \csname normal@char\string#1\endcsname
981      \bbl@activate{#2}%
982      \fi
983      \fi}%
984      {\bbl@error
985      {Cannot declare a shorthand turned off (\string#2)}
986      {Sorry, but you cannot use shorthands which have been\\%
987      turned off in the package options}}}
```

\@notshorthand

```

988 \def\@notshorthand#1{%
989   \bbl@error{%
990     The character '\string #1' should be made a shorthand character;\\%
991     add the command \string\useshorthands\string{#1\string} to
992     the preamble.\\%
993     I will ignore your instruction}%
994   {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.

```

995 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
996 \DeclareRobustCommand*\shorthandoff{%
997   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
998 \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.

```

999 \def\bbl@switch@sh#1#2{%
1000   \ifx#2\@nnil\else
1001     \bbl@ifunset{\bbl@active@\string#2}%
1002     {\bbl@error
1003       {I cannot switch '\string#2' on or off--not a shorthand}%
1004       {This character is not a shorthand. Maybe you made\\%
1005       a typing mistake? I will ignore your instruction}}}%
1006     {\ifcase#1%
1007       \catcode'#212\relax
1008       \or
1009       \catcode'#2\active
1010       \or
1011       \csname bbl@oricat@\string#2\endcsname
1012       \csname bbl@oridef@\string#2\endcsname
1013       \fi}%
1014     \bbl@afterfi\bbl@switch@sh#1%
1015   \fi}
```


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

```

1016 \def\babelshorthand{\active@prefix\babelshorthand\bb1@putsh}
1017 \def\bb1@putsh#1{%
1018   \bb1@ifunset{\bb1@active@\string#1}%
1019     {\bb1@putsh@i#1\@empty\@nnil}%
1020     {\csname bb1@active@\string#1\endcsname}}
1021 \def\bb1@putsh@i#1#2\@nnil{%
1022   \csname\language @sh@\string#1@%
1023     \ifx\@empty#2\else\string#2\fi\endcsname}
1024 \ifx\bb1@opt@shorthands\@nnil\else
1025   \let\bb1@s@initiate@active@char\initiate@active@char
1026   \def\initiate@active@char#1{%
1027     \bb1@ifshorthand{#1}{\bb1@s@initiate@active@char{#1}}{}}
1028   \let\bb1@s@switch@sh\bb1@switch@sh
1029   \def\bb1@switch@sh#1#2{%
1030     \ifx#2\@nnil\else
1031       \bb1@afterfi
1032       \bb1@ifshorthand{#2}{\bb1@s@switch@sh#1{#2}}{\bb1@switch@sh#1}%
1033       \fi}
1034   \let\bb1@s@activate\bb1@activate
1035   \def\bb1@activate#1{%
1036     \bb1@ifshorthand{#1}{\bb1@s@activate{#1}}{}}
1037   \let\bb1@s@deactivate\bb1@deactivate
1038   \def\bb1@deactivate#1{%
1039     \bb1@ifshorthand{#1}{\bb1@s@deactivate{#1}}{}}
1040 \fi

```

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

```

1041 \newcommand\ifbabelshorthand[3]{\bb1@ifunset{\bb1@active@\string#1}{#3}{#2}}

```

\bb1@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.

\bb1@pr@m@s

```

1042 \def\bb1@prim@s{%
1043   \prime\futurelet\@let@token\bb1@pr@m@s}
1044 \def\bb1@if@primes#1#2{%
1045   \ifx#1\@let@token
1046     \expandafter\@firstoftwo
1047   \else\ifx#2\@let@token
1048     \bb1@afterelse\expandafter\@firstoftwo
1049   \else
1050     \bb1@afterfi\expandafter\@secondoftwo
1051   \fi\fi}
1052 \begingroup
1053   \catcode`\^=7 \catcode`\*= \active \lccode`\^=\^
1054   \catcode`\'=12 \catcode`\`= \active \lccode`\`= \'
1055   \lowercase{%
1056     \gdef\bb1@pr@m@s{%
1057       \bb1@if@primes" "%
1058         \pr@@@s
1059         {\bb1@if@primes*\^ \pr@@@t\egroup}}
1060 \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).

```
1061 \initiate@active@char{~}
1062 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1063 \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.

```
1064 \expandafter\def\csname OT1dqpos\endcsname{127}
1065 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1066 \ifx\f@encoding\undefined
1067   \def\f@encoding{OT1}
1068 \fi
```

9.5 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```
1069 \bbl@trace{Language attributes}
1070 \newcommand\languageattribute[2]{%
1071   \def\bbl@tempc{#1}%
1072   \bbl@fixname\bbl@tempc
1073   \bbl@iflanguage\bbl@tempc{%
1074     \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.

```
1075     \ifx\bbl@known@attribs\undefined
1076       \in@false
1077     \else
```

Now we need to see if the attribute occurs in the list of already selected attributes.

```
1078       \bbl@xin@{\bbl@tempc-##1,}\bbl@known@attribs,%
1079     \fi
```

When the attribute was in the list we issue a warning; this might not be the users intention.

```
1080     \ifin@
1081       \bbl@warning{%
1082         You have more than once selected the attribute '##1'\%
1083         for language #1. Reported}%
1084     \else
```

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

```
1085       \bbl@exp{%
1086         \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
```

```

1087 \edef\bbl@tempa{\bbl@tempc-##1}%
1088 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1089 {\csname\bbl@tempc @attr##1\endcsname}%
1090 {\@attrerr{\bbl@tempc}{##1}}%
1091 \fi}}}

```

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

```

1092 \@onlypreamble\languageattribute

```

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

```

1093 \newcommand*{\@attrerr}[2]{%
1094 \bbl@error
1095 {The attribute #2 is unknown for language #1.}%
1096 {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}`.

```

1097 \def\bbl@declare@ttribute#1#2#3{%
1098 \bbl@xin@{, #2,}{, \BabelModifiers,}%
1099 \ifin@
1100 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1101 \fi
1102 \bbl@add@list\bbl@attributes{#1-#2}%
1103 \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.

```

1104 \def\bbl@ifattributeset#1#2#3#4{%
1105 \ifx\bbl@known@attribs\@undefined
1106 \in@false
1107 \else

```

The we need to check the list of known attributes.

```

1108 \bbl@xin@{, #1-#2,}{, \bbl@known@attribs,}%
1109 \fi

```

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

```

1110 \ifin@
1111 \bbl@afterelse#3%
1112 \else
1113 \bbl@afterfi#4%
1114 \fi
1115 }

```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise.

```

1116 \def\bbl@ifknown@ttrib#1#2{%

```

We first assume the attribute is unknown.

```
1117 \let\bbl@tempa\@secondoftwo
```

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

```
1118 \bbl@loopx\bbl@tempb{#2}{%
1119 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1120 \ifin@
```

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

```
1121 \let\bbl@tempa\@firstoftwo
1122 \else
1123 \fi}%
```

Finally we execute `\bbl@tempa`.

```
1124 \bbl@tempa
1125 }
```

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

```
1126 \def\bbl@clear@ttribs{%
1127 \ifx\bbl@attributes\undefined\else
1128 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1129 \expandafter\bbl@clear@ttrib\bbl@tempa.
1130 }%
1131 \let\bbl@attributes\undefined
1132 \fi}
1133 \def\bbl@clear@ttrib#1-#2.{%
1134 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1135 \AtBeginDocument{\bbl@clear@ttribs}
```

9.6 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.
`\babel@beginsave`

```
1136 \bbl@trace{Macros for saving definitions}
1137 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1138 \newcount\babel@savecnt
1139 \babel@beginsave
```

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX`³². To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented.

```
1140 \def\babel@save#1{%
1141 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1142 \toks@\expandafter{\originalTeX\let#1=}%
```

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

```

1143 \bbl@exp{%
1144   \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1145 \advance\babel@savecnt@one}

```

`\babel@savevariable` The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive.

```

1146 \def\babel@savevariable#1{%
1147   \toks@\expandafter{\originalTeX #1}%
1148   \bbl@exp{\def\originalTeX{\the\toks@the#1\relax}}}

```

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

```

1149 \def\bbl@frenchspacing{%
1150   \ifnum\the\sfcode\.\!=\@m
1151     \let\bbl@nonfrenchspacing\relax
1152   \else
1153     \frenchspacing
1154     \let\bbl@nonfrenchspacing\nonfrenchspacing
1155   \fi}
1156 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

9.7 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the macros `\text<tag>` and `\<tag>`. Definitions are first expanded so that they don't contain `\csname` but the actual macro.

```

1157 \bbl@trace{Short tags}
1158 \def\babeltags#1{%
1159   \edef\bbl@tempa{\zap@space#1 \@empty}%
1160   \def\bbl@tempb##1=##2\@{#}%
1161   \edef\bbl@tempc{%
1162     \noexpand\newcommand
1163     \expandafter\noexpand\csname ##1\endcsname{%
1164       \noexpand\protect
1165       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1166     \noexpand\newcommand
1167     \expandafter\noexpand\csname text##1\endcsname{%
1168       \noexpand\foreignlanguage{##2}}
1169   \bbl@tempc}%
1170   \bbl@for\bbl@tempa\bbl@tempa{%
1171     \expandafter\bbl@tempb\bbl@tempa\@{}}

```

9.8 Hyphens

`\babelhyphenation` This macro saves hyphenation exceptions. Two macros are used to store them: `\bbl@hyphenation@` for the global ones and `\bbl@hyphenation<lang>` for language ones. See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```

1172 \bbl@trace{Hyphens}
1173 \@onlypreamble\babelhyphenation
1174 \AtEndOfPackage{%
1175   \newcommand\babelhyphenation[2][\@empty]{%
1176     \ifx\bbl@hyphenation@\relax
1177       \let\bbl@hyphenation@\@empty
1178     \fi

```

```

1179 \ifx\bbl@hyphlist\@empty\else
1180 \bbl@warning{%
1181 You must not intermingle \string\selectlanguage\space and\\%
1182 \string\babelhyphenation\space or some exceptions will not\\%
1183 be taken into account. Reported}%
1184 \fi
1185 \ifx\@empty#1%
1186 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1187 \else
1188 \bbl@vforeach{#1}{%
1189 \def\bbl@tempa{##1}%
1190 \bbl@fixname\bbl@tempa
1191 \bbl@iflanguage\bbl@tempa{%
1192 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1193 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1194 \@empty
1195 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1196 #2}}}%
1197 \fi}}

```

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

```

1198 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1199 \def\bbl@t@one{T1}
1200 \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`.

```

1201 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1202 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1203 \def\bbl@hyphen{%
1204 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1205 \def\bbl@hyphen@i#1#2{%
1206 \bbl@ifunset{bbl@hy@#1#2\@empty}%
1207 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1208 {\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 hyphen 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 discretionaty 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.

`\nobreak` is always preceded by `\leavevmode`, in case the shorthand starts a paragraph.

```

1209 \def\bbl@usehyphen#1{%
1210 \leavevmode
1211 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1212 \nobreak\hskip\z@skip}
1213 \def\bbl@@usehyphen#1{%
1214 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1215 \def\bbl@hyphenchar{%
1216 \ifnum\hyphenchar\font=\m@ne
1217 \babellnullhyphen

```

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

```

1218 \else
1219 \char\hyphenchar\font
1220 \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.

```

1221 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}{}
1222 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}{}
1223 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1224 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1225 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}{}
1226 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}{}
1227 \def\bbl@hy@repeat{%
1228 \bbl@usehyphen{%
1229 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}{}
1230 \def\bbl@hy@@repeat{%
1231 \bbl@usehyphen{%
1232 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}{}
1233 \def\bbl@hy@empty{\hskip\z@skip}
1234 \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.

```

1235 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

```

9.9 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by `luatex` and `xetex`. The code is organized here with pseudo-guards, so we start with the basic commands.

Tools But first, a couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```

1236 \bbl@trace{Multiencoding strings}
1237 \def\bbl@tglobal#1{\global\let#1#1}
1238 \def\bbl@recatcode#1{%
1239 \@tempcnta="7F
1240 \def\bbl@tempa{%
1241 \ifnum\@tempcnta>"FF\else
1242 \catcode\@tempcnta=#1\relax
1243 \advance\@tempcnta\@ne
1244 \expandafter\bbl@tempa
1245 \fi}%
1246 \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).

```

1247 \@ifpackagewith{babel}{nocase}%
1248   {\let\bbl@patchucl\relax}%
1249   {\def\bbl@patchucl{%
1250     \global\let\bbl@patchucl\relax
1251     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1252     \gdef\bbl@uclc##1{%
1253       \let\bbl@encoded\bbl@encoded@uclc
1254       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1255       {##1}%
1256       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1257        \csname\language @bbl@uclc\endcsname}%
1258        {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1259     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1260     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1261 \langle\langle *More package options\rangle\rangle \equiv
1262 \DeclareOption{nocase}{}
1263 \rangle\rangle\langle\langle /More package options\rangle\rangle

```

The following package options control the behavior of \SetString.

```

1264 \langle\langle *More package options\rangle\rangle \equiv
1265 \let\bbl@opt@strings\@nnil % accept strings=value
1266 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1267 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1268 \def\BabelStringsDefault{generic}
1269 \rangle\rangle\langle\langle /More package options\rangle\rangle

```

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.

```

1270 \@onlypreamble\StartBabelCommands
1271 \def\StartBabelCommands{%
1272   \begingroup
1273   \bbl@recatcode{11}%
1274   \langle\langle Macros local to BabelCommands\rangle\rangle
1275   \def\bbl@provstring##1##2{%
1276     \providecommand##1{##2}%
1277     \bbl@tglobal##1}%
1278   \global\let\bbl@scafter\@empty
1279   \let\StartBabelCommands\bbl@startcmds
1280   \ifx\BabelLanguages\relax
1281     \let\BabelLanguages\CurrentOption
1282   \fi
1283   \begingroup
1284   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1285   \StartBabelCommands}
1286 \def\bbl@startcmds{%
1287   \ifx\bbl@screset\@nnil\else
1288     \bbl@usehooks{stopcommands}{}%
1289   \fi
1290   \endgroup
1291   \begingroup
1292   \@ifstar
1293     {\ifx\bbl@opt@strings\@nnil
1294       \let\bbl@opt@strings\BabelStringsDefault
1295     \fi
1296     \bbl@startcmds@i}%

```



```

1297 \bbl@startcmds@i}
1298 \def\bbl@startcmds@i#1#2{%
1299 \edef\bbl@L{\zap@space#1 \@empty}%
1300 \edef\bbl@G{\zap@space#2 \@empty}%
1301 \bbl@startcmds@ii}

```

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.

```

1302 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1303 \let\SetString@gobbletwo
1304 \let\bbl@stringdef@gobbletwo
1305 \let\AfterBabelCommands@gobble
1306 \ifx\@empty#1%
1307 \def\bbl@sc@label{generic}%
1308 \def\bbl@encstring##1##2{%
1309 \ProvideTextCommandDefault##1{##2}%
1310 \bbl@toglobal##1%
1311 \expandafter\bbl@toglobal\csname\string? \string##1\endcsname}%
1312 \let\bbl@sctest\in@true
1313 \else
1314 \let\bbl@sc@charset\space % <- zapped below
1315 \let\bbl@sc@fontenc\space % <- " "
1316 \def\bbl@tempa##1=##2\@nil{%
1317 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1318 \bbl@foreach{label=#1}{\bbl@tempa##1\@nil}%
1319 \def\bbl@tempa##1 ##2{% space -> comma
1320 ##1%
1321 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1322 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1323 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1324 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1325 \def\bbl@encstring##1##2{%
1326 \bbl@foreach\bbl@sc@fontenc{%
1327 \bbl@ifunset{T@####1}%
1328 }%
1329 {\ProvideTextCommand##1{####1}{##2}%
1330 \bbl@toglobal##1%
1331 \expandafter
1332 \bbl@toglobal\csname####1\string##1\endcsname}}}%
1333 \def\bbl@sctest{%
1334 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
1335 \fi
1336 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1337 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1338 \let\AfterBabelCommands\bbl@aftercmds
1339 \let\SetString\bbl@setstring
1340 \let\bbl@stringdef\bbl@encstring
1341 \else % ie, strings=value
1342 \bbl@sctest
1343 \fin@

```

```

1344 \let\AfterBabelCommands\bbl@aftercmds
1345 \let\SetString\bbl@setstring
1346 \let\bbl@stringdef\bbl@provstring
1347 \fi\fi\fi
1348 \bbl@scswitch
1349 \ifx\bbl@G\@empty
1350 \def\SetString##1##2{%
1351 \bbl@error{Missing group for string \string##1}%
1352 {You must assign strings to some category, typically\\%
1353 captions or extras, but you set none}}%
1354 \fi
1355 \ifx\@empty#1%
1356 \bbl@usehooks{defaultcommands}{}%
1357 \else
1358 \@expandtwoargs
1359 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1360 \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) .

```

1361 \def\bbl@forlang#1#2{%
1362 \bbl@for#1\bbl@L{%
1363 \bbl@xin@{, #1, }, {\BabelLanguages,}%
1364 \ifin@#2\relax\fi}}
1365 \def\bbl@scswitch{%
1366 \bbl@forlang\bbl@tempa{%
1367 \ifx\bbl@G\@empty\else
1368 \ifx\SetString\@gobbletwo\else
1369 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1370 \bbl@xin@{, \bbl@GL, }, {\bbl@screset,}%
1371 \ifin@\else
1372 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1373 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1374 \fi
1375 \fi
1376 \fi}}
1377 \AtEndOfPackage{%
1378 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1379 \let\bbl@scswitch\relax}
1380 \@onlypreamble\EndBabelCommands
1381 \def\EndBabelCommands{%
1382 \bbl@usehooks{stopcommands}{}%
1383 \endgroup
1384 \endgroup
1385 \bbl@scafter}

```

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.

```

1386 \def\bb@setstring#1#2{%
1387   \bb@forlang\bb@tempa{%
1388     \edef\bb@LC{\bb@tempa\bb@stripslash#1}%
1389     \bb@ifunset{\bb@LC}% eg, \germanchaptername
1390     {\global\expandafter % TODO - con \bb@exp ?
1391      \bb@add\csname\bb@G\bb@tempa\expandafter\endcsname\expandafter
1392       {\expandafter\bb@scset\expandafter#1\csname\bb@LC\endcsname}}}%
1393     }%
1394   \def\BabelString{#2}%
1395   \bb@usehooks{stringprocess}{}%
1396   \expandafter\bb@stringdef
1397     \csname\bb@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include `\bb@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`.

```

1398 \ifx\bb@opt@strings\relax
1399   \def\bb@scset#1#2{\def#1{\bb@encoded#2}}
1400   \bb@patchuclc
1401   \let\bb@encoded\relax
1402   \def\bb@encoded@uclc#1{%
1403     \@inmathwarn#1%
1404     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1405       \expandafter\ifx\csname ?\string#1\endcsname\relax
1406         \TextSymbolUnavailable#1%
1407       \else
1408         \csname ?\string#1\endcsname
1409       \fi
1410     \else
1411       \csname\cf@encoding\string#1\endcsname
1412     \fi}
1413 \else
1414   \def\bb@scset#1#2{\def#1{#2}}
1415 \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.

```

1416 <<{*Macros local to BabelCommands}>> ≡
1417 \def\SetStringLoop##1##2{%
1418   \def\bb@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1419   \count@\z@
1420   \bb@loop\bb@tempa{##2}{% empty items and spaces are ok
1421     \advance\count@\@ne
1422     \toks@\expandafter{\bb@tempa}%
1423     \bb@exp{%
1424       \\SetString\bb@templ{\romannumeral\count@}{\the\toks@}%
1425       \count@=\the\count@\relax}}}%
1426 <</Macros local to BabelCommands>>

```

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

```

1427 \def\bb@aftercmds#1{%
1428   \toks@\expandafter{\bb@scafter#1}%
1429   \xdef\bb@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.

```

1430 <<*Macros local to BabelCommands>> ≡
1431 \newcommand\SetCase[3][\%
1432   \bbl@patchuclc
1433   \bbl@forlang\bbl@tempa\%
1434   \expandafter\bbl@encstring
1435   \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1436   \expandafter\bbl@encstring
1437   \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1438   \expandafter\bbl@encstring
1439   \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1440 <</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.

```

1441 <<*Macros local to BabelCommands>> ≡
1442 \newcommand\SetHyphenMap[1]{\%
1443   \bbl@forlang\bbl@tempa\%
1444   \expandafter\bbl@stringdef
1445   \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1446 <</Macros local to BabelCommands>>

```

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

```

1447 \newcommand\BabelLower[2]{\% one to one.
1448   \ifnum\lccode#1=#2\else
1449     \babel@savevariable{\lccode#1}%
1450     \lccode#1=#2\relax
1451   \fi}
1452 \newcommand\BabelLowerMM[4]{\% many-to-many
1453   \@tempcnta=#1\relax
1454   \@tempcntb=#4\relax
1455   \def\bbl@tempa{\%
1456     \ifnum\@tempcnta>#2\else
1457       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1458       \advance\@tempcnta#3\relax
1459       \advance\@tempcntb#3\relax
1460       \expandafter\bbl@tempa
1461     \fi}%
1462   \bbl@tempa}
1463 \newcommand\BabelLowerMO[4]{\% many-to-one
1464   \@tempcnta=#1\relax
1465   \def\bbl@tempa{\%
1466     \ifnum\@tempcnta>#2\else
1467       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1468       \advance\@tempcnta#3
1469       \expandafter\bbl@tempa
1470     \fi}%
1471   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1472 <<*More package options>> ≡
1473 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1474 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1475 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1476 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}

```

```

1477 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1478 <</More package options>>

```

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

```

1479 \AtEndOfPackage{%
1480   \ifx\bb1@opt@hyphenmap\undefined
1481     \bb1@xin{,}{\bb1@language@opts}%
1482     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
1483   \fi}

```

9.10 Macros common to a number of languages

`\set@low@box` The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```

1484 \bb1@trace{Macros related to glyphs}
1485 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z\hbox{#1}%
1486   \dimen\z\ht\z@ \advance\dimen\z@ -\ht\tw@%
1487   \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.

```

1488 \def\save@sf@q#1{\leavevmode
1489   \begingroup
1490   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1491   \endgroup}

```

9.11 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through `T1enc.def`.

9.11.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```

1492 \ProvideTextCommand{\quotedblbase}{OT1}{%
1493   \save@sf@q{\set@low@box{\textquotedblright\}}%
1494   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

1495 \ProvideTextCommandDefault{\quotedblbase}{%
1496   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1497 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1498   \save@sf@q{\set@low@box{\textquoteright\}}%
1499   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

1500 \ProvideTextCommandDefault{\quotesinglbase}{%
1501   \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```
\guillemotright 1502 \ProvideTextCommand{\guillemotleft}{OT1}{%
1503   \ifmmode
1504     \ll
1505   \else
1506     \save@sf@q{\nobreak
1507       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1508   \fi}
1509 \ProvideTextCommand{\guillemotright}{OT1}{%
1510   \ifmmode
1511     \gg
1512   \else
1513     \save@sf@q{\nobreak
1514       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1515   \fi}
```

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

```
1516 \ProvideTextCommandDefault{\guillemotleft}{%
1517   \UseTextSymbol{OT1}{\guillemotleft}}
1518 \ProvideTextCommandDefault{\guillemotright}{%
1519   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1520 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1521   \ifmmode
1522     <%
1523   \else
1524     \save@sf@q{\nobreak
1525       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1526   \fi}
1527 \ProvideTextCommand{\guilsinglright}{OT1}{%
1528   \ifmmode
1529     >%
1530   \else
1531     \save@sf@q{\nobreak
1532       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1533   \fi}
```

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

```
1534 \ProvideTextCommandDefault{\guilsinglleft}{%
1535   \UseTextSymbol{OT1}{\guilsinglleft}}
1536 \ProvideTextCommandDefault{\guilsinglright}{%
1537   \UseTextSymbol{OT1}{\guilsinglright}}
```

9.11.2 Letters

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

```
\IJ 1538 \DeclareTextCommand{\ij}{OT1}{%
1539   i\kern-0.02em\bbl@allowhyphens j}
1540 \DeclareTextCommand{\IJ}{OT1}{%
1541   I\kern-0.02em\bbl@allowhyphens J}
1542 \DeclareTextCommand{\ij}{T1}{\char188}
1543 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
1544 \ProvideTextCommandDefault{\ij}{%
1545   \UseTextSymbol{OT1}{\ij}}
1546 \ProvideTextCommandDefault{\IJ}{%
1547   \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 Stipcevic Mario, (stipcevic@olimp.irb.hr).

```
1548 \def\crrtic@{\hrule height0.1ex width0.3em}
1549 \def\crttic@{\hrule height0.1ex width0.33em}
1550 \def\ddj@{%
1551   \setbox0\hbox{d}\dimen@=\ht0
1552   \advance\dimen@1ex
1553   \dimen@.45\dimen@
1554   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1555   \advance\dimen@ii.5ex
1556   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1557 \def\DDJ@{%
1558   \setbox0\hbox{D}\dimen@=.55\ht0
1559   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1560   \advance\dimen@ii.15ex % correction for the dash position
1561   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1562   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1563   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1564 %
1565 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1566 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
1567 \ProvideTextCommandDefault{\dj}{%
1568   \UseTextSymbol{OT1}{\dj}}
1569 \ProvideTextCommandDefault{\DJ}{%
1570   \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.

```
1571 \DeclareTextCommand{\SS}{OT1}{SS}
1572 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.11.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with `\ProvideTextCommandDefault`, but this is very likely not required because their definitions are based on encoding dependent macros.

`\glq` The ‘german’ single quotes.

```
\grq 1573 \ProvideTextCommandDefault{\glq}{%
1574   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```

1575 \ProvideTextCommand{\grq}{T1}{%
1576   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1577 \ProvideTextCommand{\grq}{TU}{%
1578   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1579 \ProvideTextCommand{\grq}{OT1}{%
1580   \save@sf@q{\kern-.0125em
1581     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1582     \kern.07em\relax}}
1583 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq 1584 \ProvideTextCommandDefault{\glqq}{%
1585   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1586 \ProvideTextCommand{\grqq}{T1}{%
1587   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1588 \ProvideTextCommand{\grqq}{TU}{%
1589   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1590 \ProvideTextCommand{\grqq}{OT1}{%
1591   \save@sf@q{\kern-.07em
1592     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1593     \kern.07em\relax}}
1594 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.

```

\frq 1595 \ProvideTextCommandDefault{\flq}{%
1596   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1597 \ProvideTextCommandDefault{\frq}{%
1598   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```

\frqq 1599 \ProvideTextCommandDefault{\flqq}{%
1600   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1601 \ProvideTextCommandDefault{\frqq}{%
1602   \textormath{\guillemotright}{\mbox{\guillemotright}}}

```

9.11.4 Umlauts and tremas

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

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

```

1603 \def\umlauthigh{%
1604   \def\bb1@umlauta##1{\leavevmode\bggroup%
1605     \expandafter\accent\csname\fontencoding dqpos\endcsname
1606     ##1\bb1@allowhyphens\egroup}%
1607   \let\bb1@umlaute\bb1@umlauta}
1608 \def\umlautlow{%
1609   \def\bb1@umlauta{\protect\lower@umlaut}}
1610 \def\umlautelow{%
1611   \def\bb1@umlaute{\protect\lower@umlaut}}
1612 \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.

```
1613 \expandafter\ifx\csname U@D\endcsname\relax
1614 \csname newdimen\endcsname\U@D
1615 \fi
```

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

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

```
1616 \def\lower@umlaut#1{%
1617   \leavevmode\bgroup
1618     \U@D 1ex%
1619     {\setbox\z@\hbox{%
1620       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1621       \dimen@ -.45ex\advance\dimen@\ht\z@
1622       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1623     \expandafter\accent\csname\fontencoding dqpos\endcsname
1624     \fontdimen5\font\U@D #1%
1625   \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).

```
1626 \AtBeginDocument{%
1627   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
1628   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
1629   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
1630   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
1631   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
1632   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
1633   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
1634   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
1635   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
1636   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
1637   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%
1638 }
```

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

```
1639 \ifx\l@english\undefined
1640   \chardef\l@english\z@
1641 \fi
1642 \main@language{english}
```

9.12 Layout

Work in progress.

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

```

1643 \bbl@trace{Bidi layout}
1644 \providecommand\IfBabelLayout[3]{#3}%
1645 \newcommand\BabelPatchSection[1]{%
1646   \@ifundefined{#1}{}{%
1647     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1648     \@namedef{#1}{%
1649       \ifstar{\bbl@presec@s{#1}}%
1650       {\@dblarg{\bbl@presec@x{#1}}}}}%
1651 \def\bbl@presec@x#1[#2]#3{%
1652   \bbl@exp{%
1653     \\\select@language@x{\bbl@main@language}%
1654     \\\@nameuse{\bbl@sspre@#1}%
1655     \\\@nameuse{\bbl@ss@#1}%
1656     [\\foreignlanguage{\language}\unexpanded{#2}]}%
1657     {\\foreignlanguage{\language}\unexpanded{#3}}}%
1658     \\\select@language@x{\language}}}%
1659 \def\bbl@presec@s#1#2{%
1660   \bbl@exp{%
1661     \\\select@language@x{\bbl@main@language}%
1662     \\\@nameuse{\bbl@sspre@#1}%
1663     \\\@nameuse{\bbl@ss@#1}*%
1664     {\\foreignlanguage{\language}\unexpanded{#2}}}%
1665     \\\select@language@x{\language}}}%
1666 \IfBabelLayout{sectioning}%
1667   {\BabelPatchSection{part}%
1668    \BabelPatchSection{chapter}%
1669    \BabelPatchSection{section}%
1670    \BabelPatchSection{subsection}%
1671    \BabelPatchSection{subsubsection}%
1672    \BabelPatchSection{paragraph}%
1673    \BabelPatchSection{subparagraph}%
1674    \def\babel@toc#1{%
1675      \select@language@x{\bbl@main@language}}}%
1676 \IfBabelLayout{captions}%
1677   {\BabelPatchSection{caption}}}%

```

9.13 Load engine specific macros

```

1678 \bbl@trace{Input engine specific macros}
1679 \ifcase\bbl@engine
1680   \input txtbabel.def
1681 \or
1682   \input luababel.def
1683 \or
1684   \input xebabel.def
1685 \fi

```

9.14 Creating languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```

1686 \bbl@trace{Creating languages and reading ini files}
1687 \newcommand\babelprovide[2][{}]{%
1688   \let\bbl@savelangname\language
1689   \edef\bbl@savlocaleid{\the\localeid}%

```

```

1690 % Set name and locale id
1691 \def\languagename{#2}%
1692 \bbl@id@assign
1693 \chardef\localeid\@nameuse{\bbl@id@\languagename}%
1694 \let\bbl@KVP@captions\@nil
1695 \let\bbl@KVP@import\@nil
1696 \let\bbl@KVP@main\@nil
1697 \let\bbl@KVP@script\@nil
1698 \let\bbl@KVP@language\@nil
1699 \let\bbl@KVP@dir\@nil
1700 \let\bbl@KVP@hyphenrules\@nil
1701 \let\bbl@KVP@mapfont\@nil
1702 \let\bbl@KVP@maparabic\@nil
1703 \let\bbl@KVP@mapdigits\@nil
1704 \let\bbl@KVP@intraspace\@nil
1705 \let\bbl@KVP@intrapenalty\@nil
1706 \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}% TODO - error handling
1707 \ifx\bbl@KVP@import\@nil\else
1708   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1709   {\begingroup
1710     \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1711     \InputIfFileExists{babel-#2.tex}{}}%
1712   \endgroup}%
1713   {}%
1714 \fi
1715 \ifx\bbl@KVP@captions\@nil
1716   \let\bbl@KVP@captions\bbl@KVP@import
1717 \fi
1718 % Load ini
1719 \bbl@ifunset{date#2}%
1720   {\bbl@provide@new{#2}}%
1721   {\bbl@ifblank{#1}%
1722     {\bbl@error
1723       {If you want to modify `#2' you must tell how in\\
1724       the optional argument. See the manual for the\\
1725       available options.}%
1726       {Use this macro as documented}}%
1727     {\bbl@provide@renew{#2}}}%
1728 % Post tasks
1729 \bbl@exp{\bbl@babelensure[exclude=\today]{#2}}%
1730 \bbl@ifunset{\bbl@ensure@\languagename}%
1731   {\bbl@exp{%
1732     \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1733       \\\foreignlanguage{\languagename}%
1734       {###1}}}%
1735   }%
1736 % At this point all parameters are defined if 'import'. Now we
1737 % execute some code depending on them. But what about if nothing was
1738 % imported? We just load the very basic parameters: ids and a few
1739 % more.
1740 \bbl@ifunset{\bbl@lname#2}%
1741   {\def\BabelBeforeIni##1##2{%
1742     \begingroup
1743       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
1744       \let\bbl@ini@captions@aux\@gobbletwo
1745       \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6{%
1746         \bbl@read@ini{##1}%
1747         \bbl@exportkey{chrng}{characters.ranges}{}%
1748         \bbl@exportkey{dgnat}{numbers.digits.native}{}%

```

```

1749         \endgroup}%           boxed, to avoid extra spaces:
1750         {\setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
1751     }%
1752 % -
1753 % Override script and language names with script= and language=
1754 \ifx\bb1@KVP@script\@nil\else
1755     \bb1@csarg\edef\sname@#2{\bb1@KVP@script}%
1756 \fi
1757 \ifx\bb1@KVP@language\@nil\else
1758     \bb1@csarg\edef\lname@#2{\bb1@KVP@language}%
1759 \fi
1760 % For bidi texts, to switch the language based on direction
1761 \ifx\bb1@KVP@mapfont\@nil\else
1762     \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}{}%
1763     {\bb1@error{Option '\bb1@KVP@mapfont' unknown for\%
1764         mapfont. Use 'direction'.%
1765         {See the manual for details.}}}%
1766 \bb1@ifunset{\bb1@lsys@\language\name}{\bb1@provide@lsys{\language\name}}{}%
1767 \bb1@ifunset{\bb1@wdir@\language\name}{\bb1@provide@dirs{\language\name}}{}%
1768 \ifx\bb1@mapselect\@undefined
1769     \AtBeginDocument{%
1770         \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}%
1771         {\selectfont}}%
1772     \def\bb1@mapselect{%
1773         \let\bb1@mapselect\relax
1774         \edef\bb1@prefontid{\fontid\font}}%
1775     \def\bb1@mapdir##1{%
1776         {\def\language\name{##1}%
1777         \let\bb1@ifrestoring\@firstoftwo % avoid font warning
1778         \bb1@switchfont
1779         \directlua{Babel.fontmap
1780             [\the\csname bbl@wdir@##1\endcsname]%
1781             [\bb1@prefontid]=\fontid\font}}}%
1782     \fi
1783     \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language\name}}}%
1784 \fi
1785 % For East Asian, Southeast Asian, if interspace in ini - TODO: as hook?
1786 \ifx\bb1@KVP@intraspace\@nil\else % We may override the ini
1787     \bb1@csarg\edef\intsp@#2{\bb1@KVP@intraspace}%
1788 \fi
1789 \ifcase\bb1@engine\or
1790     \bb1@ifunset{\bb1@intsp@\language\name}{}%
1791     {\expandafter\ifx\csname bbl@intsp@\language\name\endcsname\@empty\else
1792         \bb1@xin@{\bb1@cs{sbcpr@\language\name}}{Hant,Hans,Jpan,Kore,Kana}%
1793         \ifin@
1794             \bb1@cjkintraspace
1795             \directlua{
1796                 Babel = Babel or {}
1797                 Babel.locale_props = Babel.locale_props or {}
1798                 Babel.locale_props[\the\localeid].linebreak = 'c'
1799             }%
1800             \bb1@exp{\bb1@intraspace\bb1@cs{intsp@\language\name}\@}%
1801             \ifx\bb1@KVP@intrapenalty\@nil
1802                 \bb1@intrapenalty0\@@
1803             \fi
1804         \else
1805             \bb1@seaintraspace
1806             \bb1@exp{\bb1@intraspace\bb1@cs{intsp@\language\name}\@}%
1807             \directlua{

```

```

1808         Babel = Babel or {}
1809         Babel.sea_ranges = Babel.sea_ranges or {}
1810         Babel.set_chranges('\bbl@cs{sbcpr@language}\language}',
1811                             '\bbl@cs{chrng@language}\language}')
1812     }%
1813     \ifx\bbl@KVP@intrapenalty\@nil
1814         \bbl@intrapenalty0\@@
1815     \fi
1816 \fi
1817 \fi
1818 \ifx\bbl@KVP@intrapenalty\@nil\else
1819     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
1820 \fi}%
1821 \or
1822     \bbl@xin@\bbl@cs{sbcpr@language}\{Thai,Lao,Khmr}%
1823 \ifin@
1824     \bbl@ifunset\bbl@intsp@language\{}%
1825     {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
1826         \ifx\bbl@KVP@intraspace\@nil
1827             \bbl@exp{%
1828                 \\bbl@intraspace\bbl@cs{intsp@language}\\@@}%
1829             \fi
1830             \ifx\bbl@KVP@intrapenalty\@nil
1831                 \bbl@intrapenalty0\@@
1832             \fi
1833             \fi
1834             \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
1835                 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
1836             \fi
1837             \ifx\bbl@KVP@intrapenalty\@nil\else
1838                 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
1839             \fi
1840             \ifx\bbl@ispace\@undefined
1841                 \AtBeginDocument{%
1842                     \expandafter\bbl@add
1843                     \csname selectfont \endcsname\bbl@ispace}%
1844                 \def\bbl@ispace{\bbl@cs{xeisp@\bbl@cs{sbcpr@language}}}%
1845                 \fi}%
1846 \fi
1847 \fi
1848 % Native digits, if provided in ini (TeX level, xe and lua)
1849 \ifcase\bbl@engine\else
1850     \bbl@ifunset\bbl@dgnat@language\{}%
1851     {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
1852         \expandafter\expandafter\expandafter
1853         \bbl@setdigits\csname bbl@dgnat@language\endcsname
1854         \ifx\bbl@KVP@maparabic\@nil\else
1855             \ifx\bbl@latinarabic\@undefined
1856                 \expandafter\let\expandafter\@arabic
1857                 \csname bbl@counter@language\endcsname
1858             \else % ie, if layout=counters, which redefines \@arabic
1859                 \expandafter\let\expandafter\bbl@latinarabic
1860                 \csname bbl@counter@language\endcsname
1861             \fi
1862         \fi
1863     \fi}%
1864 \fi
1865 % Native digits (lua level).
1866 \ifodd\bbl@engine

```

```

1867 \ifx\bbl@KVP@mapdigits\@nil\else
1868 \bbl@ifunset{\bbl@dgnat@language\name}{}%
1869 {\RequirePackage{luatexbase}}%
1870 \bbl@activate@preotf
1871 \directlua{
1872     Babel = Babel or {} %%% -> presets in luababel
1873     Babel.digits_mapped = true
1874     Babel.digits = Babel.digits or {}
1875     Babel.digits[\the\localeid] =
1876         table.pack(string.utfvalue('\bbl@cs{dgnat@language\name}'))
1877     if not Babel.numbers then
1878         function Babel.numbers(head)
1879             local LOCALE = luatexbase.registernumber'bbl@attr@locale'
1880             local GLYPH = node.id'glyph'
1881             local inmath = false
1882             for item in node.traverse(head) do
1883                 if not inmath and item.id == GLYPH then
1884                     local temp = node.get_attribute(item, LOCALE)
1885                     if Babel.digits[temp] then
1886                         local chr = item.char
1887                         if chr > 47 and chr < 58 then
1888                             item.char = Babel.digits[temp][chr-47]
1889                         end
1890                     end
1891                 elseif item.id == node.id'math' then
1892                     inmath = (item.subtype == 0)
1893                 end
1894             end
1895             return head
1896         end
1897     end
1898 }%
1899 \fi
1900 \fi
1901 % To load or reload the babel-*.tex, if require.babel in ini
1902 \bbl@ifunset{\bbl@rqtex@language\name}{}%
1903 {\expandafter\ifx\csname\bbl@rqtex@language\endcsname\@empty\else
1904     \let\BabelBeforeIni\@gobbletwo
1905     \chardef\atcatcode=\catcode\@
1906     \catcode\@=11\relax
1907     \InputIfFileExists{babel-\bbl@cs{rqtex@language\name}.tex}{}%
1908     \catcode\@=\atcatcode
1909     \let\atcatcode\relax
1910 \fi}%
1911 \let\language\name\bbl@savelangname
1912 \chardef\localeid\bbl@savelocaleid\relax}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in \TeX .

```

1913 \def\bbl@setdigits#1#2#3#4#5{%
1914     \bbl@exp{%
1915         \def<\language digits>####1{% ie, \langdigits
1916             \<\bbl@digits@language>####1\@nil}%
1917         \def<\language counter>####1{% ie, \langcounter
1918             \expandafter\<\bbl@counter@language>%
1919             \csname c@####1\endcsname}%
1920         \def<\bbl@counter@language>####1{% ie, \bbl@counter@lang
1921             \expandafter\<\bbl@digits@language>%
1922             \number####1\@nil}}%

```

```

1923 \def\bbl@tempa##1##2##3##4##5{%
1924   \bbl@exp{%    Wow, quite a lot of hashes! :-(
1925     \def\<bbl@digits@\language\>#####1{%
1926       \ifx#####1\\\@nil          % ie, \bbl@digits@lang
1927       \else
1928         \ifx0#####1#1%
1929         \else\ifx1#####1#2%
1930         \else\ifx2#####1#3%
1931         \else\ifx3#####1#4%
1932         \else\ifx4#####1#5%
1933         \else\ifx5#####1##1%
1934         \else\ifx6#####1##2%
1935         \else\ifx7#####1##3%
1936         \else\ifx8#####1##4%
1937         \else\ifx9#####1##5%
1938         \else#####1%
1939         \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
1940         \expandafter\<bbl@digits@\language\>%
1941       \fi}}}%
1942 \bbl@tempa}

```

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

-

```

1943 \def\bbl@provide@new#1{%
1944   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1945   \@namedef{extras#1}{}%
1946   \@namedef{noextras#1}{}%
1947   \StartBabelCommands*{#1}{captions}%
1948   \ifx\bbl@KVP@captions\@nil %    and also if import, implicit
1949     \def\bbl@tempb##1{%          elt for \bbl@captionslist
1950       \ifx##1\@empty\else
1951         \bbl@exp{%
1952           \SetString\##1{%
1953             \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
1954         \expandafter\bbl@tempb
1955       \fi}%
1956   \expandafter\bbl@tempb\bbl@captionslist\@empty
1957   \else
1958     \bbl@read@ini{\bbl@KVP@captions}% Here all letters cat = 11
1959     \bbl@after@ini
1960     \bbl@savestrings
1961   \fi
1962   \StartBabelCommands*{#1}{date}%
1963   \ifx\bbl@KVP@import\@nil
1964     \bbl@exp{%
1965       \SetString\today{\bbl@nocaption{today}{#1today}}}%
1966   \else
1967     \bbl@savetoday
1968     \bbl@savedate
1969   \fi
1970   \EndBabelCommands
1971   \bbl@exp{%
1972     \def\<#1hyphenmins>{%
1973       {\bbl@ifunset{\bbl@lfthm@#1}{2}{\@nameuse{\bbl@lfthm@#1}}}%
1974       {\bbl@ifunset{\bbl@rgthm@#1}{3}{\@nameuse{\bbl@rgthm@#1}}}}}%
1975   \bbl@provide@hyphens{#1}%
1976   \ifx\bbl@KVP@main\@nil\else
1977     \expandafter\main@language\expandafter{#1}%
1978   \fi}

```

```

1979 \def\bbl@provide@renew#1{%
1980   \ifx\bbl@KVP@captions\@nil\else
1981     \StartBabelCommands*{#1}{captions}%
1982     \bbl@read@ini{\bbl@KVP@captions}%   Here all letters cat = 11
1983     \bbl@after@ini
1984     \bbl@savestrings
1985     \EndBabelCommands
1986   \fi
1987   \ifx\bbl@KVP@import\@nil\else
1988     \StartBabelCommands*{#1}{date}%
1989     \bbl@savetoday
1990     \bbl@savedate
1991     \EndBabelCommands
1992   \fi
1993   \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

1994 \def\bbl@provide@hyphens#1{%
1995   \let\bbl@tempa\relax
1996   \ifx\bbl@KVP@hyphenrules\@nil\else
1997     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
1998     \bbl@foreach\bbl@KVP@hyphenrules{%
1999       \ifx\bbl@tempa\relax    % if not yet found
2000         \bbl@ifsamestring{##1}{+}%
2001         {\bbl@exp{\addlanguage<l@##1>}}}%
2002       {}%
2003       \bbl@ifunset{l@##1}%
2004       {}%
2005       {\bbl@exp{\let\bbl@tempa<l@##1>}}}%
2006     \fi}%
2007   \fi
2008   \ifx\bbl@tempa\relax %      if no opt or no language in opt found
2009     \ifx\bbl@KVP@import\@nil\else % if importing
2010       \bbl@exp{%
2011         \bbl@ifblank{\@nameuse{\bbl@hyphr@#1}}%
2012         {}%
2013         {\let\bbl@tempa<l@\@nameuse{\bbl@hyphr@\language}\>}}%
2014       \fi
2015     \fi
2016     \bbl@ifunset{\bbl@tempa}%      ie, relax or undefined
2017     {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
2018      {\bbl@exp{\adddialect<l@#1>\language}}%
2019      {}}%                        so, l@<lang> is ok - nothing to do
2020     {\bbl@exp{\adddialect<l@#1>\bbl@tempa}}% found in opt list or ini

```

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. *TODO - Work in progress.*

```

2021 \def\bbl@read@ini#1{%
2022   \openin1=babel-#1.ini          % FIXME - number must not be hardcoded
2023   \ifeof1
2024     \bbl@error
2025     {There is no ini file for the requested language\%
2026      (#1). Perhaps you misspelled it or your installation\%
2027      is not complete.}%
2028     {Fix the name or reinstall babel.}%
2029   \else
2030     \let\bbl@section\@empty
2031     \let\bbl@savestrings\@empty
2032     \let\bbl@savetoday\@empty

```



```

2033 \let\bbl@savestate\empty
2034 \let\bbl@inireader\bbl@iniskip
2035 \bbl@info{Importing data from babel-#1.ini for \language}%
2036 \loop
2037 \if T\ifeof1F\fi T\relax % Trick, because inside \loop
2038 \endlinechar\m@ne
2039 \read1 to \bbl@line
2040 \endlinechar`\^^M
2041 \ifx\bbl@line\empty\else
2042 \expandafter\bbl@inline\bbl@line\bbl@inline
2043 \fi
2044 \repeat
2045 \fi}
2046 \def\bbl@inline#1\bbl@inline{%
2047 \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inireader}#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.

```

2048 \def\bbl@iniskip#1\@@{%      if starts with ;
2049 \def\bbl@inisec[#1]#2\@@{%   if starts with opening bracket
2050 \@nameuse{\bbl@secpost\bbl@section}% ends previous section
2051 \def\bbl@section#1{%
2052 \@nameuse{\bbl@secpre\bbl@section}% starts current section
2053 \bbl@ifunset{\bbl@inikv@#1}%
2054 {\let\bbl@inireader\bbl@iniskip}%
2055 {\bbl@exp{\let\bbl@inireader\<bbl@inikv@#1>}}}

```

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

```

2056 \def\bbl@inikv#1=#2\@@{%      key=value
2057 \bbl@trim\def\bbl@tempa{#1}%
2058 \bbl@trim\toks@{#2}%
2059 \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.

```

2060 \def\bbl@exportkey#1#2#3{%
2061 \bbl@ifunset{\bbl@kv@#2}%
2062 {\bbl@csarg\gdef{#1@\language}{#3}}%
2063 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\empty
2064 \bbl@csarg\gdef{#1@\language}{#3}}%
2065 \else
2066 \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
2067 \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.

```

2068 \let\bbl@inikv@identification\bbl@inikv
2069 \def\bbl@secpost@identification{%
2070 \bbl@exportkey{lname}{identification.name.english}{}%
2071 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
2072 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2073 \bbl@exportkey{sname}{identification.script.name}{}%
2074 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2075 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
2076 \let\bbl@inikv@typography\bbl@inikv
2077 \let\bbl@inikv@characters\bbl@inikv
2078 \let\bbl@inikv@numbers\bbl@inikv
2079 \def\bbl@after@ini{%

```

```

2080 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2081 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2082 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2083 \bbl@exportkey{intsp}{typography.intraspace}{}%
2084 \bbl@exportkey{jstfy}{typography.justify}{w}%
2085 \bbl@exportkey{chrng}{characters.ranges}{}%
2086 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2087 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2088 \bbl@xin@{0.5}{\@nameuse{bbl@kv@identification.version}}%
2089 \ifin@
2090 \bbl@warning{%
2091     There are neither captions nor date in ``\language'`.\\%
2092     It may not be suitable for proper typesetting, and it\\%
2093     could change. Reported}%
2094 \fi
2095 \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
2096 \ifin@
2097 \bbl@warning{%
2098     The ``\language'` date format may not be suitable\\%
2099     for proper typesetting, and therefore it very likely will\\%
2100     change in a future release. Reported}%
2101 \fi
2102 \bbl@toglobal\bbl@savetoday
2103 \bbl@toglobal\bbl@savestate}

```

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

```

2104 \ifcase\bbl@engine
2105 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
2106     \bbl@ini@captions@aux{#1}{#2}}
2107 \else
2108 \def\bbl@inikv@captions#1=#2\@@{%
2109     \bbl@ini@captions@aux{#1}{#2}}
2110 \fi

```

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

```

2111 \def\bbl@ini@captions@aux#1#2{%
2112     \bbl@trim@def\bbl@tempa{#1}%
2113     \bbl@ifblank{#2}%
2114     {\bbl@exp{%
2115         \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2116     {\bbl@trim\toks@{#2}}}%
2117 \bbl@exp{%
2118     \bbl@add\bbl@savestrings{%
2119         \SetString\<\bbl@tempa name>{\the\toks@}}}

```

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

TODO. Remove cypypaste pattern.

```

2120 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%           for defaults
2121     \bbl@inidate#1...\relax{#2}}
2122 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
2123     \bbl@inidate#1...\relax{#2}{islamic}}
2124 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
2125     \bbl@inidate#1...\relax{#2}{hebrew}}
2126 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
2127     \bbl@inidate#1...\relax{#2}{persian}}

```

```

2128 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
2129 \bbl@inidate#1...\relax{#2}{indian}}
2130 \ifcase\bbl@engine
2131 \bbl@csarg\def{inikv@date.gregorian.ligr}#1=#2\@@{% override
2132 \bbl@inidate#1...\relax{#2}{}}
2133 \bbl@csarg\def{secpre@date.gregorian.ligr}{% discard uni
2134 \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
2135 \fi
2136 % eg: 1=months, 2=wide, 3=1, 4=dummy
2137 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2138 \bbl@trim@def\bbl@tempa{#1.#2}%
2139 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savestate
2140 {\bbl@trim@def\bbl@tempa{#3}%
2141 \bbl@trim\toks@{#5}%
2142 \bbl@exp{%
2143 \\\bbl@add\\bbl@savestate{%
2144 \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}%
2145 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
2146 {\bbl@trim@def\bbl@toreplace{#5}%
2147 \bbl@TG@date
2148 \global\bbl@csarg\let{date@\language name}\bbl@toreplace
2149 \bbl@exp{%
2150 \gdef\<\language name date>{\\\protect\<\language name date >}%
2151 \gdef\<\language name date >####1####2####3{%
2152 \\\bbl@usedategrouptrue
2153 \<bbl@ensure@\language name>{%
2154 \<bbl@date@\language name>{####1}{####2}{####3}}}%
2155 \\\bbl@add\\bbl@savetoday{%
2156 \\\SetString\\today{%
2157 \<\language name date>{\\\the\year}{\\the\month}{\\the\day}}}}}%
2158 {}

```

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.

```

2159 \let\bbl@calendar\empty
2160 \newcommand\BabelDateSpace{\nobreakspace}
2161 \newcommand\BabelDateDot{.\@}
2162 \newcommand\BabelDated[1]{\number#1}
2163 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
2164 \newcommand\BabelDateM[1]{\number#1}
2165 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2166 \newcommand\BabelDateMMM[1]{\ifnum#1<100 0\fi\number#1}
2167 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
2168 \newcommand\BabelDatey[1]{\number#1}%
2169 \newcommand\BabelDateyy[1]{\ifnum#1<10 0\number#1 %
2170 \else\ifnum#1<100 \number#1 %
2171 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2172 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2173 \else
2174 \bbl@error
2175 {Currently two-digit years are restricted to the\
2176 range 0-9999.}%
2177 {There is little you can do. Sorry.}%
2178 \fi\fi\fi\fi}
2179 \fi\fi\fi\fi}
2180 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2181 \def\bbl@replace@finish@iii#1{%
2182 \bbl@exp{\def\\#1####1####2####3{\the\toks@}}

```

```

2183 \def\bbl@TG@date{%
2184   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
2185   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot{}}%
2186   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
2187   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
2188   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
2189   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
2190   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{###2}}%
2191   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
2192   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
2193   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
2194 % Note after \bbl@replace \toks@ contains the resulting string.
2195 % TODO - Using this implicit behavior doesn't seem a good idea.
2196   \bbl@replace@finish@iii\bbl@toreplace}

```

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

```

2197 \def\bbl@provide@lsys#1{%
2198   \bbl@ifunset{bbl@lname@#1}%
2199     {\bbl@ini@ids{#1}}%
2200     {}%
2201   \bbl@csarg\let{lsys@#1}\@empty
2202   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
2203   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
2204   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2205   \bbl@ifunset{bbl@lname@#1}{}%
2206   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2207   \bbl@csarg\bbl@to@global{lsys@#1}}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language.

```

2208 \def\bbl@ini@ids#1{%
2209   \def\BabelBeforeIni###2{%
2210     \begingroup
2211       \bbl@add\bbl@secpost@identification{\closein1 }%
2212       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
2213       \bbl@read@ini{##1}%
2214     \endgroup}% boxed, to avoid extra spaces:
2215     {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}{}}}

```

10 The kernel of Babel (babel.def, only L^AT_EX)

10.1 The redefinition of the style commands

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

```

2216 {\def\format{plain}
2217 \ifx\fmtname\format
2218 \else
2219   \def\format{LaTeX2e}
2220   \ifx\fmtname\format

```

```

2221 \else
2222 \aftergroup\endinput
2223 \fi
2224 \fi}

```

10.2 Cross referencing macros

The \LaTeX book states:

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

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

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

The only way to accomplish this in most cases is to use the trick described in the \TeX book [2] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, ‘`\meaning\A`’ with `\A` defined as ‘`\def\A#1{\B}`’ expands to the characters ‘`macro:#1->\B`’ with all category codes set to ‘other’ or ‘space’.

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

```

2225 %\bbl@redefine\newlabel#1#2{%
2226 % \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activessfalse}

```

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

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

```

2227 <<(*More package options)>> ≡
2228 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2229 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2230 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2231 <</More package options>>

```

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

```

2232 \bbl@trace{Cross referencing macros}
2233 \ifx\bbl@opt@safe\@empty\else
2234 \def\@newl@bel#1#2#3{%
2235   {\@safe@activestrue
2236     \bbl@ifunset{#1#2}%
2237     \relax
2238     {\gdef\@multiplelabels{%
2239       \@latex@warning@no@line{There were multiply-defined labels}}%
2240     \@latex@warning@no@line{Label `#2' multiply defined}}%
2241     \global\@namedef{#1#2}{#3}}

```

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

```

2242 \CheckCommand*\@testdef[3]{%
2243 \def\reserved@a{#3}%
2244 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2245 \else
2246 \@tempswatrue
2247 \fi}

```

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

```

2248 \def\@testdef#1#2#3{%
2249 \@safe@activestrue

```

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

```

2250 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname

```

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

```

2251 \def\bbl@tempb{#3}%
2252 \@safe@activesfalse

```

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

```

2253 \ifx\bbl@tempa\relax
2254 \else
2255 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2256 \fi

```

We do the same for \bbl@tempb.

```

2257 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%

```

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

```

2258 \ifx\bbl@tempa\bbl@tempb
2259 \else
2260 \@tempswatrue
2261 \fi}
2262 \fi

```

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

```

2263 \bbl@xin@{R}\bbl@opt@safe
2264 \ifin@
2265 \bbl@redefineroobust\ref#1{%
2266 \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
2267 \bbl@redefineroobust\pageref#1{%
2268 \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
2269 \else
2270 \let\org@ref\ref
2271 \let\org@pageref\pageref
2272 \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.

```

2273 \bbl@xin@{B}\bbl@opt@safe
2274 \ifin@
2275 \bbl@redefine\@citex[#1]#2{%
2276 \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
2277 \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.

```
2278 \AtBeginDocument{%
2279   \@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.)

```
2280   \def\@citex[#1][#2]#3{%
2281     \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
2282     \org@citex[#1][#2]{\@tempa}}%
2283   }{}}
```

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

```
2284 \AtBeginDocument{%
2285   \@ifpackageloaded{cite}{%
2286     \def\@citex[#1]#2{%
2287       \@safe@activestrue\org@citex[#1]{#2}\@safe@activesfalse}%
2288     }{}}
```

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

```
2289 \bbl@redefine\nocite#1{%
2290   \@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 `\bbl@bibcite` in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\bbl@bibcite`. This new definition is then activated.

```
2291 \bbl@redefine\bibcite{%
2292   \bbl@cite@choice
2293   \bibcite}
```

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

```
2294 \def\bbl@bibcite#1#2{%
2295   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
2296 \def\bbl@cite@choice{%
2297   \global\let\bibcite\bbl@bibcite}
```

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

```
2298   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
2299   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
2300 \global\let\bbl@cite@choice\relax}
```

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

```
2301 \AtBeginDocument{\bbl@cite@choice}
```

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

```
2302 \bbl@redefine\@bibitem#1{%
2303 \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
2304 \else
2305 \let\org@nocite\nocite
2306 \let\org@@citex\@citex
2307 \let\org@bibcite\bibcite
2308 \let\org@@bibitem\@bibitem
2309 \fi
```

10.3 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of \markright and \markboth somewhat. We check whether the argument is empty; if it is, we just make sure the scratch token register is empty. Next, we store the argument to \markright in the scratch token register. This way these commands will not be expanded later, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while \@safe@activestrue is in effect.

```
2310 \bbl@trace{Marks}
2311 \IfBabelLayout{sectioning}
2312 {\ifx\bbl@opt@headfoot\@nnil
2313 \g@addto@macro\@resetactivechars{%
2314 \set@typeset@protect
2315 \expandafter\select@language@x\expandafter{\bbl@main@language}%
2316 \let\protect\noexpand
2317 \edef\thepage{%
2318 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2319 \fi}
2320 {\bbl@redefine\markright#1{%
2321 \bbl@ifblank{#1}%
2322 {\org@markright{}}%
2323 {\toks@{#1}%
2324 \bbl@exp{%
2325 \org@markright{\protect\foreignlanguage{\language}\thetoks@}%
2326 {\protect\bbl@restore@actives\thetoks@}}}%

```

`\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.

```
2327 \ifx\@mkboth\markboth
2328 \def\bbl@tempc{\let\@mkboth\markboth}
2329 \else
2330 \def\bbl@tempc{}
2331 \fi
```


Now we can start the new definition of `\markboth`

```

2332 \bbl@redefine\markboth#1#2{%
2333   \protected@edef\bbl@tempb##1{%
2334     \protect\foreignlanguage
2335       {\language\name}{\protect\bbl@restore@actives##1}}%
2336   \bbl@ifblank{#1}%
2337     {\toks@{}}%
2338     {\toks@\expandafter{\bbl@tempb{#1}}}%
2339   \bbl@ifblank{#2}%
2340     {\@temptokena{}}%
2341     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2342   \bbl@exp{\org@markboth{the\toks@}{the\@temptokena}}}
```

and copy it to `\mkboth` if necessary.

```

2343 \bbl@tempc} % end \IfBabelLayout
```

10.4 Preventing clashes with other packages

10.4.1 `ifthen`

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

```

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

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

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

```

2344 \bbl@trace{Preventing clashes with other packages}
2345 \bbl@xin@{R}\bbl@opt@safe
2346 \ifin@
2347   \AtBeginDocument{%
2348     \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```

2349   \bbl@redefine@long\ifthenelse#1#2#3{%
```

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

```

2350   \let\bbl@temp@pref\pageref
2351   \let\pageref\org@pageref
2352   \let\bbl@temp@ref\ref
2353   \let\ref\org@ref
```

Then we can set the `\@safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments. When the package wasn't loaded we do nothing.

```

2354   \@safe@activetrue
2355   \org@ifthenelse{#1}%
2356     {\let\pageref\bbl@temp@pref
2357      \let\ref\bbl@temp@ref
```

```

2358         \@safe@activesfalse
2359         #2}%
2360     {\let\pageref\bbl@temp@pref
2361      \let\ref\bbl@temp@ref
2362      \@safe@activesfalse
2363      #3}%
2364     }%
2365   }{}%
2366 }

```

10.4.2 varioref

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

```

\Ref 2367 \AtBeginDocument{%
2368     \@ifpackageloaded{varioref}{%
2369         \bbl@redefine\@vpageref#1[#2]#3{%
2370             \@safe@activestrue
2371             \org@@@vpageref{#1}[#2]#3}%
2372         \@safe@activesfalse}%

```

The same needs to happen for `\vrefpagemum`.

```

2373     \bbl@redefine\vrefpagemum#1#2{%
2374         \@safe@activestrue
2375         \org@vrefpagemum{#1}#2}%
2376     \@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.

```

2377     \expandafter\def\csname Ref \endcsname#1{%
2378         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2379     }{}%
2380 }
2381 \fi

```

10.4.3 hhline

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

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

```

2382 \AtEndOfPackage{%
2383     \AtBeginDocument{%
2384         \@ifpackageloaded{hhline}%

```

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

```

2385         {\expandafter\ifx\csname normal@char:string\endcsname\relax
2386             \else

```

In that case we simply reload the package. Note that this happens *after* the category code of the `@-sign` has been changed to other, so we need to temporarily change it to letter again.

```

2387         \makeatletter
2388         \def\@currname{hhline}\input{hhline.sty}\makeatother

```

```

2389     \fi}%
2390   {}}}

```

10.4.4 hyperref

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

```

2391 \AtBeginDocument{%
2392   \ifx\pdfstringdefDisableCommands\undefined\else
2393     \pdfstringdefDisableCommands{\languageshorthands{system}}%
2394   \fi}

```

10.4.5 fancyhdr

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

```

2395 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2396   \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.

```

2397 \def\substitutefontfamily#1#2#3{%
2398   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2399   \immediate\write15{%
2400     \string\ProvidesFile{#1#2.fd}%
2401     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2402     \space generated font description file]^^J
2403     \string\DeclareFontFamily{#1}{#2}{^^J
2404     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
2405     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
2406     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
2407     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
2408     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
2409     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
2410     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
2411     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
2412   }%
2413   \closeout15
2414 }

```

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

```

2415 \@onlypreamble\substitutefontfamily

```

10.5 Encoding and fonts

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

\ensureascii

```
2416 \bbl@trace{Encoding and fonts}
2417 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
2418 \newcommand\BabelNonText{TS1,T3,TS3}
2419 \let\org@TeX\TeX
2420 \let\org@LaTeX\LaTeX
2421 \let\ensureascii\@firstofone
2422 \AtBeginDocument{%
2423   \in@false
2424   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2425     \ifin@ \else
2426       \lowercase{\bbl@xin@{,#1enc.def,},{, \@filelist,}}%
2427     \fi}%
2428   \ifin@ % if a text non-ascii has been loaded
2429     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2430     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2431     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2432     \def\bbl@tempb#1@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
2433     \def\bbl@tempc#1ENC.DEF#2\@@{\%
2434       \ifx\@empty#2\else
2435         \bbl@ifunset{T@#1}%
2436         {}%
2437         {\bbl@xin@{,#1,},{, \BabelNonASCII, \BabelNonText,}%
2438         \ifin@
2439           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2440           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2441         \else
2442           \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2443         \fi}%
2444       \fi}%
2445     \bbl@foreach\@filelist{\bbl@tempb#1@@}% TODO - \@ de mas??
2446     \bbl@xin@{\cf@encoding,},{, \BabelNonASCII, \BabelNonText,}%
2447     \ifin@ \else
2448       \edef\ensureascii#1{%
2449         \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2450     \fi
2451   \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.

```
2452 \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.

```
2453 \AtBeginDocument{%
2454   \@ifpackageloaded{fontspec}%
2455   {\xdef\latinencoding{%
2456     \ifx\UTFencname\@undefined
2457       EU\ifcase\bbl@engine\or2\or1\fi
2458     \else
2459       \UTFencname
```

```

2460     \fi}}%
2461 {\gdef\latinencoding{OT1}%
2462 \ifx\cf@encoding\bbl@t@one
2463 \xdef\latinencoding{\bbl@t@one}%
2464 \else
2465 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
2466 \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.

```

2467 \DeclareRobustCommand{\latintext}{%
2468 \fontencoding{\latinencoding}\selectfont
2469 \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.

```

2470 \ifx\@undefined\DeclareTextFontCommand
2471 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2472 \else
2473 \DeclareTextFontCommand{\textlatin}{\latintext}
2474 \fi

```

10.6 Basic bidi support

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

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

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

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua \TeX -ja` shows, vertical typesetting is possible, too. Its main drawback is font handling is often considered to be less mature than `xetex`, mainly in Indic scripts (but there are steps to make `HarfBuzz`, the `xetex` font engine, available in `luatex`; see <<https://github.com/tatzetwerk/luatex-harfbuzz>>).

```

2475 \bbl@trace{Basic (internal) bidi support}
2476 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2477 \def\bbl@rscripts{%
2478 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2479 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
2480 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
2481 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%

```

```

2482 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2483 Old South Arabian,}%
2484 \def\bbl@provide@dirs#1{%
2485   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2486   \ifin@
2487     \global\bbl@csarg\chardef{wdir@#1}\@ne
2488     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2489     \ifin@
2490       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2491       \fi
2492     \else
2493       \global\bbl@csarg\chardef{wdir@#1}\z@
2494       \fi
2495     \ifodd\bbl@engine
2496       \bbl@csarg\ifcase{wdir@#1}%
2497         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
2498         \or
2499         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
2500         \or
2501         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
2502         \fi
2503       \fi}
2504 \def\bbl@switchdir{%
2505   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
2506   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
2507   \bbl@exp{\bbl@setdirs\bbl@cs{wdir@\languagename}}%
2508 \def\bbl@setdirs#1{% TODO - math
2509   \ifcase\bbl@select@type % TODO - strictly, not the right test
2510     \bbl@bodydir{#1}%
2511     \bbl@pardir{#1}%
2512   \fi
2513   \bbl@texmdir{#1}}
2514 \ifodd\bbl@engine % luatex=1
2515   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2516   \DisableBabelHook{babel-bidi}
2517   \chardef\bbl@thetexmdir\z@
2518   \chardef\bbl@thepardir\z@
2519   \def\bbl@getluadir#1{%
2520     \directlua{
2521       if tex.#1dir == 'TLT' then
2522         tex.sprint('0')
2523       elseif tex.#1dir == 'TRT' then
2524         tex.sprint('1')
2525       end}}
2526 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
2527   \ifcase#3\relax
2528     \ifcase\bbl@getluadir{#1}\relax\else
2529       #2 TLT\relax
2530     \fi
2531   \else
2532     \ifcase\bbl@getluadir{#1}\relax
2533       #2 TRT\relax
2534     \fi
2535   \fi}
2536 \def\bbl@texmdir#1{%
2537   \bbl@setluadir{text}\texmdir{#1}%
2538   \chardef\bbl@thetexmdir#1\relax
2539   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2540 \def\bbl@pardir#1{%

```

```

2541 \bbl@setluadir{par}\pardir{#1}%
2542 \chardef\bbl@thepardir#1\relax}
2543 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2544 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2545 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
2546 % Sadly, we have to deal with boxes in math with basic.
2547 % Activated every math with the package option bidi=:
2548 \def\bbl@mathboxdir{%
2549 \ifcase\bbl@thetextdir\relax
2550 \everyhbox{\textdir TLT\relax}%
2551 \else
2552 \everyhbox{\textdir TRT\relax}%
2553 \fi}
2554 \else % pdftex=0, xetex=2
2555 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2556 \DisableBabelHook{babel-bidi}
2557 \newcount\bbl@dirlevel
2558 \chardef\bbl@thetextdir\z@
2559 \chardef\bbl@thepardir\z@
2560 \def\bbl@textdir#1{%
2561 \ifcase#1\relax
2562 \chardef\bbl@thetextdir\z@
2563 \bbl@textdir@i\beginL\endL
2564 \else
2565 \chardef\bbl@thetextdir\@ne
2566 \bbl@textdir@i\beginR\endR
2567 \fi}
2568 \def\bbl@textdir@i#1#2{%
2569 \ifhmode
2570 \ifnum\currentgrouplevel>\z@
2571 \ifnum\currentgrouplevel=\bbl@dirlevel
2572 \bbl@error{Multiple bidi settings inside a group}%
2573 {I'll insert a new group, but expect wrong results.}%
2574 \bgroup\aftergroup#2\aftergroup\egroup
2575 \else
2576 \ifcase\currentgrouptype\or % 0 bottom
2577 \aftergroup#2% 1 simple {}
2578 \or
2579 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2580 \or
2581 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2582 \or\or\or % vbox vtop align
2583 \or
2584 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2585 \or\or\or\or\or\or % output math disc insert vcent mathchoice
2586 \or
2587 \aftergroup#2% 14 \begingroup
2588 \else
2589 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2590 \fi
2591 \fi
2592 \bbl@dirlevel\currentgrouplevel
2593 \fi
2594 #1%
2595 \fi}
2596 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2597 \let\bbl@bodydir\@gobble
2598 \let\bbl@pagedir\@gobble
2599 \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).

```

2600 \def\bbl@xebidipar{%
2601   \let\bbl@xebidipar\relax
2602   \TeXeTstate\@ne
2603   \def\bbl@xeverypar{%
2604     \ifcase\bbl@thepardir
2605       \ifcase\bbl@thetextdir\else\beginR\fi
2606     \else
2607       {\setbox\z@\lastbox\beginR\box\z@}%
2608     \fi}%
2609   \let\bbl@severypar\everypar
2610   \newtoks\everypar
2611   \everypar=\bbl@severypar
2612   \bbl@severypar{\bbl@xeverypar\the\everypar}}
2613 \@ifpackagewith{babel}{bidi=bidi}%
2614 {\let\bbl@textdir@i@gobbletwo
2615   \let\bbl@xebidipar\@empty
2616   \AddBabelHook{bidi}{foreign}{%
2617     \def\bbl@tempa{\def\BabelText###1}%
2618     \ifcase\bbl@thetextdir
2619       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
2620     \else
2621       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
2622     \fi}
2623   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}}
2624 {}%
2625 \fi

```

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

```

2626 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
2627 \AtBeginDocument{%
2628   \ifx\pdfstringdefDisableCommands\@undefined\else
2629     \ifx\pdfstringdefDisableCommands\relax\else
2630       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2631     \fi
2632   \fi}

```

10.7 Local Language Configuration

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

```

2633 \bbl@trace{Local Language Configuration}
2634 \ifx\loadlocalcfg\@undefined
2635   \@ifpackagewith{babel}{noconfigs}%
2636   {\let\loadlocalcfg@gobble}%
2637   {\def\loadlocalcfg#1{%
2638     \InputIfFileExists{#1.cfg}%
2639     {\typeout{*****^J%
2640               * Local config file #1.cfg used^^J%
2641               *}}}%
2642   \@empty}}

```



```
2643 \fi
```

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

```
2644 \ifx\@unexpandable@protect\@undefined
2645   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2646   \long\def\protected@write#1#2#3{%
2647     \begingroup
2648       \let\thepage\relax
2649       #2%
2650       \let\protect\@unexpandable@protect
2651       \edef\reserved@a{\write#1{#3}}%
2652       \reserved@a
2653     \endgroup
2654   \if@nobreak\ifvmode\nobreak\fi\fi}
2655 \fi
2656 </core>
2657 <*kernel>
```

11 Multiple languages (switch.def)

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

```
2658 <<Make sure ProvidesFile is defined>>
2659 \ProvidesFile{switch.def}[\<date>] <<version>> Babel switching mechanism]
2660 <<Load macros for plain if not LaTeX>>
2661 <<Define core switching macros>>
```

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

```
2662 \def\bbl@version{\<version>}
2663 \def\bbl@date{\<date>}
2664 \def\adddialect#1#2{%
2665   \global\chardef#1#2\relax
2666   \bbl@usehooks{adddialect}{\#1}{\#2}}%
2667   \log{\string#1 = a dialect from \string\language#2}}
```

`\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.

```
2668 \def\bbl@fixname#1{%
2669   \begingroup
2670     \def\bbl@tempe{l@}%
2671     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2672     \bbl@tempd
2673     {\lowercase\expandafter{\bbl@tempd}%
2674      {\uppercase\expandafter{\bbl@tempd}%
2675       \@empty
2676       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2677        \uppercase\expandafter{\bbl@tempd}}}%
2678     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2679      \lowercase\expandafter{\bbl@tempd}}}%

```

```

2680      \@empty
2681      \edef\bb1@tempd{\endgroup\def\noexpand#1{#1}}%
2682      \bb1@tempd}
2683 \def\bb1@iflanguage#1{%
2684   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

```

2685 \def\iflanguage#1{%
2686   \bb1@iflanguage{#1}{%
2687     \ifnum\csname l@#1\endcsname=\language
2688       \expandafter\@firstoftwo
2689     \else
2690       \expandafter\@secondoftwo
2691     \fi}}

```

11.1 Selecting the language

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

To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character. To convert a control sequence to a string, we use the `\string` primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer `\escapechar` to a character number, we have to compare this number with the character of the string. To do this we have to use \TeX 's backquote notation to specify the character as a number. If the first character of the `\string`'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or `\escapechar` is set to a value outside of the character range 0–255. If the user gives an empty argument, we provide a default argument for `\string`. This argument should expand to nothing.

```

2692 \let\bb1@select@type\z@
2693 \edef\selectlanguage{%
2694   \noexpand\protect
2695   \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`.

```

2696 \ifx\@undefined\protect\let\protect\relax\fi

```

As \LaTeX 2.09 writes to files *expanded* whereas \LaTeX 2_ε takes care *not* to expand the arguments of `\write` statements we need to be a bit clever about the way we add information to .aux files. Therefore we introduce the macro `\xstring` which should expand to the right amount of `\string`'s.

```

2697 \ifx\documentclass\@undefined
2698   \def\xstring{\string\string\string}
2699 \else
2700   \let\xstring\string
2701 \fi

```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

`\bbl@pop@language` But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need T_EX'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.

```
2702 \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`

```
2703 \def\bbl@push@language{%
```

```
2704 \xdef\bbl@language@stack{\language+\bbl@language@stack}}
```

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

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

```
2705 \def\bbl@pop@lang#1+#2-#3{%
```

```
2706 \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 T_EX 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.

```
2707 \let\bbl@ifrestoring\@secondoftwo
```

```
2708 \def\bbl@pop@language{%
```

```
2709 \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
```

```
2710 \let\bbl@ifrestoring\@firstoftwo
```

```
2711 \expandafter\bbl@set@language\expandafter{\language}%
```

```
2712 \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.

```
2713 \chardef\localeid\z@
```

```
2714 \def\bbl@id@last{0} % No real need for a new counter
```

```
2715 \def\bbl@id@assign{%
```

```
2716 \bbl@ifunset{bbl@id@@\language}%
```

```

2717 {\count@bbl@id@last\relax
2718 \advance\count@\@ne
2719 \bbl@csarg\chardef{id@\language}\count@
2720 \edef\bbl@id@last{\the\count@}%
2721 \ifcase\bbl@engine\or
2722 \directlua{
2723     Babel = Babel or {}
2724     Babel.locale_props = Babel.locale_props or {}
2725     Babel.locale_props[\bbl@id@last] = {}
2726 }%
2727 \fi}%
2728 {}

```

The unprotected part of `\selectlanguage`.

```

2729 \expandafter\def\csname selectlanguage \endcsname#1{%
2730 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
2731 \bbl@push@language
2732 \aftergroup\bbl@pop@language
2733 \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.

```

2734 \def\BabelContentsFiles{toc,lof,lot}
2735 \def\bbl@set@language#1{% from selectlanguage, pop@
2736 \edef\language{%
2737 \ifnum\escapechar=\expandafter`\string#1\@empty
2738 \else\string#1\@empty\fi}%
2739 \select@language{\language}%
2740 % write to aux
2741 \expandafter\ifx\csname date\language\endcsname\relax\else
2742 \if@filesw
2743 \protected@write\@auxout{}\string\babel@aux{\language}\fi}%
2744 \bbl@usehooks{write}\fi}%
2745 \fi
2746 \fi}
2747 \def\select@language#1{% from set@, babel@aux
2748 % set hmap
2749 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2750 % set name
2751 \edef\language{#1}%
2752 \bbl@fixname\language
2753 \bbl@iflanguage\language{%
2754 \expandafter\ifx\csname date\language\endcsname\relax
2755 \bbl@error
2756 {Unknown language `#1'. Either you have\\%
2757 misspelled its name, it has not been installed,\\%
2758 or you requested it in a previous run. Fix its name,\\%
2759 install it or just rerun the file, respectively. In\\%
2760 some cases, you may need to remove the aux file}%
2761 {You may proceed, but expect wrong results}%
2762 \else
2763 % set type

```

```

2764 \let\bbl@select@type\z@
2765 \expandafter\bbl@switch\expandafter{\language}%
2766 \fi}}
2767 \def\babel@aux#1#2{%
2768 \expandafter\ifx\csname date#1\endcsname\relax
2769 \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
2770 \@namedef{bbl@auxwarn@#1}{}%
2771 \bbl@warning
2772 {Unknown language `#1'. Very likely you\\%
2773 requested it in a previous run. Expect some\\%
2774 wrong results in this run, which should vanish\\%
2775 in the next one. Reported}%
2776 \fi
2777 \else
2778 \select@language{#1}%
2779 \bbl@foreach\BabelContentsFiles{%
2780 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
2781 \fi}
2782 \def\babel@toc#1#2{%
2783 \select@language{#1}}

```

A bit of optimization. Select in heads/foots the language only if necessary. The real thing is in `babel.def`.

```
2784 \let\select@language@x\select@language
```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring \TeX in a certain pre-defined state.

The name of the language is stored in the control sequence `\language`.

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\<lang>hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\<lang>hyphenmins` will be used.

```

2785 \newif\ifbbl@usedategroup
2786 \def\bbl@switch#1{% from select@, foreign@
2787 % restore
2788 \originalTeX
2789 \expandafter\def\expandafter\originalTeX\expandafter{%
2790 \csname noextras#1\endcsname
2791 \let\originalTeX\@empty
2792 \babel@beginsave}%
2793 \bbl@usehooks{afterreset}{}%
2794 \languageshortands{none}%
2795 % set the locale id
2796 \bbl@id@assign
2797 \chardef\localeid\@nameuse{bbl@id@\@language}%
2798 % switch captions, date
2799 \ifcase\bbl@select@type
2800 \ifhmode
2801 \hskip\z@skip % trick to ignore spaces
2802 \csname captions#1\endcsname\relax

```

```

2803     \csname date#1\endcsname\relax
2804     \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2805     \else
2806         \csname captions#1\endcsname\relax
2807         \csname date#1\endcsname\relax
2808     \fi
2809 \else
2810     \ifbbl@usedategroup % if \foreign... within \<lang>date
2811         \bbl@usedategroupfalse
2812         \ifhmode
2813             \hskip\z@skip % trick to ignore spaces
2814             \csname date#1\endcsname\relax
2815             \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2816         \else
2817             \csname date#1\endcsname\relax
2818         \fi
2819     \fi
2820 \fi
2821 % switch extras
2822 \bbl@usehooks{beforeextras}{}%
2823 \csname extras#1\endcsname\relax
2824 \bbl@usehooks{afterextras}{}%
2825 % > babel-ensure
2826 % > babel-sh-<short>
2827 % > babel-bidi
2828 % > babel-fontspec
2829 % hyphenation - case mapping
2830 \ifcase\bbl@opt@hyphenmap\or
2831     \def\BabelLower##1##2{\lccode##1=##2\relax}%
2832     \ifnum\bbl@hymapsel>4\else
2833         \csname\language @bbl@hyphenmap\endcsname
2834     \fi
2835     \chardef\bbl@opt@hyphenmap\z@
2836 \else
2837     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2838         \csname\language @bbl@hyphenmap\endcsname
2839     \fi
2840 \fi
2841 \global\let\bbl@hymapsel@cclv
2842 % hyphenation - patterns
2843 \bbl@patterns{#1}%
2844 % hyphenation - mins
2845 \babel@savevariable\lefthyphenmin
2846 \babel@savevariable\righthyphenmin
2847 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2848     \set@hyphenmins\tw@\thr@@\relax
2849 \else
2850     \expandafter\expandafter\expandafter\set@hyphenmins
2851     \csname #1hyphenmins\endcsname\relax
2852 \fi}

```

otherlanguage The other language environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to. The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

2853 \long\def\otherlanguage#1{%

```

```

2854 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
2855 \csname selectlanguage \endcsname{#1}%
2856 \ignorespaces}

```

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

```

2857 \long\def\endotherlanguage{%
2858   \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`.

```

2859 \expandafter\def\csname otherlanguage*\endcsname#1{%
2860   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2861   \foreign@language{#1}}

```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```

2862 \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`.

```

2863 \providecommand\bbl@beforeforeign{}
2864 \edef\foreignlanguage{%
2865   \noexpand\protect
2866   \expandafter\noexpand\csname foreignlanguage \endcsname}
2867 \expandafter\def\csname foreignlanguage \endcsname{%
2868   \@ifstar\bbl@foreign@s\bbl@foreign@x}
2869 \def\bbl@foreign@x#1#2{%
2870   \begingroup
2871     \let\BabelText\@firstofone
2872     \bbl@beforeforeign
2873     \foreign@language{#1}%
2874     \bbl@usehooks{foreign}{}%
2875     \BabelText{#2}% Now in horizontal mode!

```

```

2876 \endgroup}
2877 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
2878 \begingroup
2879 {\par}%
2880 \let\BabelText\@firstofone
2881 \foreign@language{#1}%
2882 \bbl@usehooks{foreign*}{}%
2883 \bbl@dirparastext
2884 \BabelText{#2}% Still in vertical mode!
2885 {\par}%
2886 \endgroup}

```

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

```

2887 \def\foreign@language#1{%
2888 % set name
2889 \edef\language{#1}%
2890 \bbl@fixname\language
2891 \bbl@iflanguage\language{\language\endcsname\relax
2892 \expandafter\ifx\csname date\language\endcsname\relax
2893 \bbl@warning % TODO - why a warning, not an error?
2894 {Unknown language `#1'. Either you have\\%
2895 misspelled its name, it has not been installed,\\%
2896 or you requested it in a previous run. Fix its name,\\%
2897 install it or just rerun the file, respectively. In\\%
2898 some cases, you may need to remove the aux file.\\%
2899 I'll proceed, but expect wrong results.\\%
2900 Reported}%
2901 \fi
2902 % set type
2903 \let\bbl@select@type\@ne
2904 \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.

```

2905 \let\bbl@hyphlist\@empty
2906 \let\bbl@hyphenation@\relax
2907 \let\bbl@pttnlist\@empty
2908 \let\bbl@patterns@\relax
2909 \let\bbl@hymapsel=\@cclv
2910 \def\bbl@patterns#1{%
2911 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2912 \csname l@#1\endcsname
2913 \edef\bbl@tempa{#1}%
2914 \else
2915 \csname l@#1:\f@encoding\endcsname
2916 \edef\bbl@tempa{#1:\f@encoding}%
2917 \fi
2918 \@expandtwoargs\bbl@usehooks{patterns}{\bbl@tempa}}

```



```

2919 % > luatex
2920 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
2921   \begingroup
2922     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
2923     \ifin@else
2924       \@expandtwoargs\bbl@usehooks{hyphenation}{\#1}{\bbl@tempa}}%
2925     \hyphenation{%
2926       \bbl@hyphenation@
2927       \@ifundefined{bbl@hyphenation@#1}%
2928       \@empty
2929       {\space\csname bbl@hyphenation@#1\endcsname}}%
2930     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2931   \fi
2932 \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*`.

```

2933 \def\hyphenrules#1{%
2934   \edef\bbl@tempf{#1}%
2935   \bbl@fixname\bbl@tempf
2936   \bbl@iflanguage\bbl@tempf{%
2937     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2938     \languageshortands{none}%
2939     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2940       \set@hyphenmins\tw@\thr@@\relax
2941     \else
2942       \expandafter\expandafter\expandafter\set@hyphenmins
2943       \csname\bbl@tempf hyphenmins\endcsname\relax
2944     \fi}}
2945 \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.

```

2946 \def\providehyphenmins#1#2{%
2947   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2948     \@namedef{#1hyphenmins}{#2}%
2949   \fi}

```

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

```

2950 \def\set@hyphenmins#1#2{%
2951   \lefthyphenmin#1\relax
2952   \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.

```

2953 \ifx\ProvidesFile\@undefined
2954   \def\ProvidesLanguage#1[#2 #3 #4]{%
2955     \wlog{Language: #1 #4 #3 <#2>}%
2956   }
2957 \else
2958   \def\ProvidesLanguage#1{%

```

```

2959 \begingroup
2960 \catcode`\ 10 %
2961 \@makeother\/%
2962 \@ifnextchar[%]
2963 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2964 \def\@provideslanguage#1[#2]{%
2965 \wlog{Language: #1 #2}%
2966 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2967 \endgroup}
2968 \fi

```

`\LdfInit` This macro is defined in two versions. The first version is to be part of the ‘kernel’ of babel, ie. the part that is loaded in the format; the second version is defined in `babel.def`. The version in the format just checks the category code of the ampersand and then loads `babel.def`.

The category code of the ampersand is restored and the macro calls itself again with the new definition from `babel.def`

```

2969 \def\LdfInit{%
2970 \chardef\atcatcode=\catcode`\@
2971 \catcode`\@=11\relax
2972 \input babel.def\relax
2973 \catcode`\@=\atcatcode \let\atcatcode\relax
2974 \LdfInit}

```

`\originalTeX` The macro `\originalTeX` should be known to \TeX at this moment. As it has to be expandable we `\let` it to `\@empty` instead of `\relax`.

```

2975 \ifx\originalTeX\undefined\let\originalTeX\@empty\fi

```

Because this part of the code can be included in a format, we make sure that the macro which initialises the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

2976 \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’:

```

2977 \providecommand\setlocale{%
2978 \bbl@error
2979 {Not yet available}%
2980 {Find an armchair, sit down and wait}}
2981 \let\uselocale\setlocale
2982 \let\locale\setlocale
2983 \let\selectlocale\setlocale
2984 \let\textlocale\setlocale
2985 \let\textlanguage\setlocale
2986 \let\languagegetext\setlocale

```

11.2 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'.

```

2987 \edef\bbl@nulllanguage{\string\language=0}
2988 \ifx\PackageError\@undefined
2989   \def\bbl@error#1#2{%
2990     \begingroup
2991       \newlinechar=`^^J
2992       \def\{^^J(babel) }%
2993       \errhelp{#2}\errmessage{\#1}%
2994     \endgroup}
2995   \def\bbl@warning#1{%
2996     \begingroup
2997       \newlinechar=`^^J
2998       \def\{^^J(babel) }%
2999       \message{\#1}%
3000     \endgroup}
3001   \def\bbl@info#1{%
3002     \begingroup
3003       \newlinechar=`^^J
3004       \def\{^^J}%
3005       \wlog{#1}%
3006     \endgroup}
3007 \else
3008   \def\bbl@error#1#2{%
3009     \begingroup
3010       \def\{\MessageBreak}%
3011       \PackageError{babel}{#1}{#2}%
3012     \endgroup}
3013   \def\bbl@warning#1{%
3014     \begingroup
3015       \def\{\MessageBreak}%
3016       \PackageWarning{babel}{#1}%
3017     \endgroup}
3018   \def\bbl@info#1{%
3019     \begingroup
3020       \def\{\MessageBreak}%
3021       \PackageInfo{babel}{#1}%
3022     \endgroup}
3023 \fi
3024 \@ifpackagewith{babel}{silent}
3025   {\let\bbl@info@gobble
3026    \let\bbl@warning@gobble}
3027   {}
3028 \def\bbl@nocaption{\protect\bbl@nocaption@i}
3029 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
3030   \global\@namedef{#2}{\textbf{?#1?}}%
3031   \@nameuse{#2}%
3032   \bbl@warning{%
3033     \@backslashchar#2 not set. Please, define\\%
3034     it in the preamble with something like:\\%
3035     \string\renewcommand\@backslashchar#2{..}\\%
3036     Reported}}
3037 \def\bbl@tentative{\protect\bbl@tentative@i}
3038 \def\bbl@tentative@i#1{%
3039   \bbl@warning{%
3040     Some functions for '#1' are tentative.\\%
3041     They might not work as expected and their behavior\\%
3042     could change in the future.\\%

```

```

3043   Reported}}
3044 \def\@nolanerr#1{%
3045   \bbl@error
3046   {You haven't defined the language #1\space yet}%
3047   {Your command will be ignored, type <return> to proceed}}
3048 \def\@nopatterns#1{%
3049   \bbl@warning
3050   {No hyphenation patterns were preloaded for\\%
3051    the language `#1' into the format.\\%
3052    Please, configure your TeX system to add them and\\%
3053    rebuild the format. Now I will use the patterns\\%
3054    preloaded for \bbl@nulllanguage\space instead}}
3055 \let\bbl@usehooks\@gobbletwo
3056 </kernel>
3057 <*patterns>

```

12 Loading hyphenation patterns

The following code is meant to be read by $\text{\texttt{iniTeX}}$ because it should instruct $\text{\texttt{TeX}}$ to read hyphenation patterns. To this end the `docstrip` option `patterns` can be used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

We want to add a message to the message $\text{\texttt{L\TeX}}$ 2.09 puts in the `\everyjob` register. This could be done by the following code:

```

\let\orgeveryjob\everyjob
\def\everyjob#1{%
  \orgeveryjob{#1}%
  \orgeveryjob\expandafter{\the\orgeveryjob\immediate\write16{%
    hyphenation patterns for \the\loaded@patterns loaded.}}%
  \let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}

```

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before $\text{\texttt{L\TeX}}$ fills the register.

There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with $\text{\texttt{SL\TeX}}$ the above scheme won't work. The reason is that $\text{\texttt{SL\TeX}}$ overwrites the contents of the `\everyjob` register with its own message.
- Plain $\text{\texttt{TeX}}$ does not use the `\everyjob` register so the message would not be displayed.

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, `\dump`. Therefore the original `\dump` is saved in `\org@dump` and a new definition is supplied.

To make sure that $\text{\texttt{L\TeX}}$ 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.

```

3058 <<Make sure ProvidesFile is defined>>
3059 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
3060 \xdef\bbl@format{\jobname}

```

```

3061 \ifx\AtBeginDocument\@undefined
3062   \def\@empty{}
3063   \let\orig@dump\dump
3064   \def\dump{%
3065     \ifx\@ztryfc\@undefined
3066       \else
3067         \toks0=\expandafter{\@preamblecmds}%
3068         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3069         \def\@begindocumenthook{}%
3070       \fi
3071       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3072 \fi
3073 <<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.

```

3074 \def\process@line#1#2 #3 #4 {%
3075   \ifx=#1%
3076     \process@synonym{#2}%
3077   \else
3078     \process@language{#1#2}{#3}{#4}%
3079   \fi
3080   \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.

```

3081 \toks@{}
3082 \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.

```

3083 \def\process@synonym#1{%
3084   \ifnum\last@language=\m@ne
3085     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3086   \else
3087     \expandafter\chardef\csname l@#1\endcsname\last@language
3088     \wlog{\string\l@#1=\string\language\the\last@language}%
3089     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3090       \csname\language\hyphenmins\endcsname
3091     \let\bbl@elt\relax
3092     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
3093   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read. For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the

name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form `\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

```

3094 \def\process@language#1#2#3{%
3095   \expandafter\addlanguage\csname l@#1\endcsname
3096   \expandafter\language\csname l@#1\endcsname
3097   \edef\language#1#2#3%
3098   \bbl@hook@everylanguage{#1}%
3099   % > luatex
3100   \bbl@get@enc#1::\@@@
3101   \begingroup
3102     \lefthyphenmin\m@ne
3103     \bbl@hook@loadpatterns{#2}%
3104     % > luatex
3105     \ifnum\lefthyphenmin=\m@ne
3106     \else
3107       \expandafter\xdef\csname #1hyphenmins\endcsname{%
3108         \the\lefthyphenmin\the\righthyphenmin}%
3109     \fi
3110   \endgroup
3111   \def\bbl@tempa{#3}%
3112   \ifx\bbl@tempa\@empty\else
3113     \bbl@hook@loadexceptions{#3}%
3114     % > luatex
3115   \fi
3116   \let\bbl@elt\relax
3117   \edef\bbl@languages{%
3118     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3119   \ifnum\the\language=\z@
3120     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3121       \set@hyphenmins\tw@\thr@@\relax
3122     \else
3123       \expandafter\expandafter\expandafter\set@hyphenmins
3124       \csname #1hyphenmins\endcsname
3125     \fi
3126     \the\toks@
3127     \toks@{}%
```

3128 \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.

3129 \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.

```
3130 \def\bbl@hook@everylanguage#1{}
3131 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3132 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3133 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
3134 \begingroup
3135   \def\AddBabelHook#1#2{%
3136     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3137       \def\next{\toks1}%
3138     \else
3139       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3140     \fi
3141     \next}
3142 \ifx\directlua\@undefined
3143   \ifx\XeTeXinputencoding\@undefined\else
3144     \input xebabel.def
3145   \fi
3146 \else
3147   \input luababel.def
3148 \fi
3149 \openin1 = babel-\bbl@format.cfg
3150 \ifeof1
3151 \else
3152   \input babel-\bbl@format.cfg\relax
3153 \fi
3154 \closein1
3155 \endgroup
3156 \bbl@hook@loadkernel{switch.def}
```

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

3157 \openin1 = language.dat

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

```
3158 \def\languagename{english}%
3159 \ifeof1
3160   \message{I couldn't find the file language.dat,\space
3161           I will try the file hyphen.tex}
3162   \input hyphen.tex\relax
3163   \chardef\l@english\z@
3164 \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.

3165 \last@language\m@ne

We now read lines from the file until the end is found

3166 \loop

While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
3167 \endlinechar\m@ne
3168 \read1 to \bbl@line
3169 \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.

```
3170 \if T\ifeof1F\fi T\relax
3171 \ifx\bbl@line\@empty\else
3172 \edef\bbl@line{\bbl@line\space\space\space}%
3173 \expandafter\process@line\bbl@line\relax
3174 \fi
3175 \repeat
```

Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns.

```
3176 \begingroup
3177 \def\bbl@elt#1#2#3#4{%
3178 \global\language=#2\relax
3179 \gdef\language#1}%
3180 \def\bbl@elt##1##2##3##4{}}%
3181 \bbl@languages
3182 \endgroup
3183 \fi
```

and close the configuration file.

```
3184 \closein1
```

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

```
3185 \if/\the\toks@\else
3186 \errhelp{language.dat loads no language, only synonyms}
3187 \errmessage{Orphan language synonym}
3188 \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.

```
3189 \let\bbl@line\@undefined
3190 \let\process@line\@undefined
3191 \let\process@synonym\@undefined
3192 \let\process@language\@undefined
3193 \let\bbl@get@enc\@undefined
3194 \let\bbl@hyph@enc\@undefined
3195 \let\bbl@tempa\@undefined
3196 \let\bbl@hook@loadkernel\@undefined
3197 \let\bbl@hook@everylanguage\@undefined
3198 \let\bbl@hook@loadpatterns\@undefined
3199 \let\bbl@hook@loadexceptions\@undefined
3200 \patterns)
```

Here the code for `initTeX` ends.

13 Font handling with fontspec

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

```

3201 <<{*More package options}>> ≡
3202 \ifodd\bb1@engine
3203   \DeclareOption{bidi=basic-r}%
3204     {\ExecuteOptions{bidi=basic}}
3205   \DeclareOption{bidi=basic}%
3206     {\let\bb1@beforeforeign\leavevmode
3207       % TODO - to locale_props, not as separate attribute
3208       \newattribute\bb1@attr@dir
3209       % I don't like it, hackish:
3210       \frozen@everymath\expandafter{%
3211         \expandafter\bb1@mathboxdir\the\frozen@everymath}%
3212       \frozen@everydisplay\expandafter{%
3213         \expandafter\bb1@mathboxdir\the\frozen@everydisplay}%
3214       \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
3215     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
3216 \else
3217   \DeclareOption{bidi=basic-r}%
3218     {\ExecuteOptions{bidi=basic}}
3219   \DeclareOption{bidi=basic}%
3220     {\bb1@error
3221       {The bidi method `basic' is available only in\\%
3222         luatex. I'll continue with `bidi=default', so\\%
3223         expect wrong results}%
3224       {See the manual for further details.}%
3225     \let\bb1@beforeforeign\leavevmode
3226     \AtEndOfPackage{%
3227       \EnableBabelHook{babel-bidi}%
3228       \bb1@xebidipar}}
3229   \def\bb1@loadxebidi#1{%
3230     \ifx\RTLfootnotetext\@undefined
3231       \AtEndOfPackage{%
3232         \EnableBabelHook{babel-bidi}%
3233         \ifx\fontspec\@undefined
3234           \usepackage{fontspec}% bidi needs fontspec
3235         \fi
3236         \usepackage#1{bidi}}}%
3237     \fi}
3238   \DeclareOption{bidi=bidi}%
3239     {\bb1@tentative{bidi=bidi}%
3240     \bb1@loadxebidi{}}
3241   \DeclareOption{bidi=bidi-r}%
3242     {\bb1@tentative{bidi=bidi-r}%
3243     \bb1@loadxebidi{[rldocument]}}
3244   \DeclareOption{bidi=bidi-l}%
3245     {\bb1@tentative{bidi=bidi-l}%
3246     \bb1@loadxebidi{}}
3247 \fi
3248 \DeclareOption{bidi=default}%
3249   {\let\bb1@beforeforeign\leavevmode
3250   \ifodd\bb1@engine
3251     \newattribute\bb1@attr@dir
3252     \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
3253   \fi

```

```

3254 \AtEndOfPackage{%
3255   \EnableBabelHook{babel-bidi}%
3256   \ifodd\bbbl@engine\else
3257     \bbbl@xebidipar
3258   \fi}}
3259 <</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.

```

3260 <<{*Font selection}>> ≡
3261 \bbbl@trace{Font handling with fontspec}
3262 \@onlypreamble\babelfont
3263 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
3264   \edef\bbbl@tempa{#1}%
3265   \def\bbbl@tempb{#2}%
3266   \ifx\fontspec\undefined
3267     \usepackage{fontspec}%
3268   \fi
3269   \EnableBabelHook{babel-fontspec}% Just calls \bbbl@switchfont
3270   \bbbl@babelfont}
3271 \newcommand\bbbl@babelfont[2][]{% 1=features 2=fontname
3272   \bbbl@ifunset{\bbbl@tempb family}{\bbbl@providefam{\bbbl@tempb}}{}}%
3273 % For the default font, just in case:
3274 \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\@languagename}}{}}%
3275 \expandafter\bbbl@ifblank\expandafter{\bbbl@tempa}%
3276   {\bbbl@csarg\edef{\bbbl@tempb dflt@}{<#{1}{#2}}% save \bbbl@rmdflt@
3277   \bbbl@exp{%
3278     \let<\bbbl@bbbl@tempb dflt@\@languagename>\<\bbbl@bbbl@tempb dflt@>%
3279     \\\bbbl@font@set<\bbbl@bbbl@tempb dflt@\@languagename>%
3280       \<\bbbl@tempb default>\<\bbbl@tempb family>}}%
3281   {\bbbl@foreach\bbbl@tempa{% ie \bbbl@rmdflt@lang / *scrt
3282     \bbbl@csarg\def{\bbbl@tempb dflt@##1}{<#{1}{#2}}}}}%

```

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

```

3283 \def\bbbl@providefam#1{%
3284   \bbbl@exp{%
3285     \\\newcommand\<#1default>{}% Just define it
3286     \\\bbbl@add@list\\bbbl@font@fams{#1}%
3287     \\\DeclareRobustCommand\<#1family>%
3288       \\\not@math@alphabet\<#1family>\relax
3289     \\\fontfamily\<#1default>\selectfont}%
3290     \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}

```

The following macro is activated when the hook babel-fontspec is enabled.

```

3291 \def\bbbl@switchfont{%
3292   \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\@languagename}}{}}%
3293   \bbbl@exp{% eg Arabic -> arabic
3294     \lowercase{\edef\\bbbl@tempa{\bbbl@cs{sname@\@languagename}}}}%
3295   \bbbl@foreach\bbbl@font@fams{%
3296     \bbbl@ifunset{\bbbl@##1dflt@\@languagename}% (1) language?
3297     {\bbbl@ifunset{\bbbl@##1dflt@*\bbbl@tempa}% (2) from script?
3298       {\bbbl@ifunset{\bbbl@##1dflt@}% 2=F - (3) from generic?
3299         {}}% 123=F - nothing!
3300       {\bbbl@exp{% 3=T - from generic
3301         \global\let<\bbbl@##1dflt@\@languagename>%
3302         \<\bbbl@##1dflt@>}}}%
3303     {\bbbl@exp{% 2=T - from script
3304       \global\let<\bbbl@##1dflt@\@languagename>%
3305       \<\bbbl@##1dflt@*\bbbl@tempa>}}}%

```

```

3306     {}}%                                1=T - language, already defined
3307 \def\bb1@tempa{%
3308   \bb1@warning{The current font is not a standard family:\%
3309     \fontname\font\%
3310     Script and Language are not applied. Consider\%
3311     defining a new family with \string\babelfont.\%
3312     Reported}}%
3313 \bb1@foreach\bb1@font@fams{%      don't gather with prev for
3314   \bb1@ifunset{\bb1@##1dflt@\language}%
3315     {\bb1@cs{famrst@##1}%
3316       \global\bb1@csarg\let{famrst@##1}\relax}%
3317     {\bb1@exp{% order is relevant
3318       \\\bb1@add\\\originalTeX{%
3319         \\\bb1@font@rst{\bb1@cs{##1dflt@\language}}}%
3320         \<##1default>\<##1family>{##1}}}%
3321       \\\bb1@font@set{\bb1@##1dflt@\language}% the main part!
3322       \<##1default>\<##1family>}}}%
3323 \bb1@ifrestoring{}{\bb1@tempa}}%

```

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 \bb1@mapselect because \selectfont is called internally when a font is defined.

```

3324 \def\bb1@font@set#1#2#3{% eg \bb1@rmdflt@lang \rmdefault \rmfamily
3325   \bb1@xin@{<>}{#1}%
3326   \ifin@
3327     \bb1@exp{\\\bb1@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
3328   \fi
3329   \bb1@exp{%
3330     \def\\#2{#1}%          eg, \rmdefault{\bb1@rmdflt@lang}
3331     \\\bb1@ifsamestring{#2}{\f@family}{\\#3\let\\bb1@tempa\relax}{}}%
3332 %      TODO - next should be global?, but even local does its job. I'm
3333 %      still not sure -- must investigate:
3334 \def\bb1@fontspec@set#1#2#3#4{% eg \bb1@rmdflt@lang fnt-opt fnt-nme \xxfamily
3335   \let\bb1@tempe\bb1@mapselect
3336   \let\bb1@mapselect\relax
3337   \let\bb1@temp@fam#4%      eg, '\rmfamily', to be restored below
3338   \let#4\relax              % So that can be used with \newfontfamily
3339   \bb1@exp{%
3340     \let\\bb1@temp@pfam\<\bb1@stripslash#4\space>% eg, '\rmfamily '
3341     \<keys_if_exist:nnF>{fontspec-opentype}%
3342       {Script/\bb1@cs{sname@\language}}}%
3343     {\\\newfontscript{\bb1@cs{sname@\language}}}%
3344     {\bb1@cs{sotf@\language}}}%
3345     \<keys_if_exist:nnF>{fontspec-opentype}%
3346       {Language/\bb1@cs{lname@\language}}}%
3347     {\\\newfontlanguage{\bb1@cs{lname@\language}}}%
3348     {\bb1@cs{lotf@\language}}}%
3349     \\\newfontfamily\\#4%
3350     [\bb1@cs{lsys@\language},#2]{#3}% ie \bb1@exp{.}{#3}
3351   \begingroup
3352     #4%
3353     \xdef#1{\f@family}%      eg, \bb1@rmdflt@lang{FreeSerif(0)}
3354   \endgroup
3355   \let#4\bb1@temp@fam
3356   \bb1@exp{\let\<\bb1@stripslash#4\space>\bb1@temp@pfam
3357   \let\bb1@mapselect\bb1@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.

```
3358 \def\bbbl@font@rst#1#2#3#4{%
3359   \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.

```
3360 \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 :-).

```
3361 \newcommand\babelFSstore[2][{%
3362   \bbbl@ifblank{#1}%
3363   {\bbbl@csarg\def{sname@#2}{Latin}}%
3364   {\bbbl@csarg\def{sname@#2}{#1}}%
3365   \bbbl@provide@dirs{#2}%
3366   \bbbl@csarg\ifnum{wdir@#2}>\z@
3367     \let\bbbl@beforeforeign\leavevmode
3368     \EnableBabelHook{babel-bidi}%
3369   \fi
3370   \bbbl@foreach{#2}{%
3371     \bbbl@FSstore{##1}{rm}\rmdefault\bbbl@save@rmdefault
3372     \bbbl@FSstore{##1}{sf}\sfdefault\bbbl@save@sfdefault
3373     \bbbl@FSstore{##1}{tt}\ttdefault\bbbl@save@ttdefault}}
3374 \def\bbbl@FSstore#1#2#3#4{%
3375   \bbbl@csarg\edef{#2default#1}{#3}%
3376   \expandafter\addto\csname extras#1\endcsname{%
3377     \let#4#3%
3378     \ifx#3\f@family
3379       \edef#3{\csname bbl@#2default#1\endcsname}%
3380       \fontfamily{#3}\selectfont
3381     \else
3382       \edef#3{\csname bbl@#2default#1\endcsname}%
3383     \fi}%
3384   \expandafter\addto\csname noextras#1\endcsname{%
3385     \ifx#3\f@family
3386       \fontfamily{#4}\selectfont
3387     \fi
3388     \let#3#4}}
3389 \let\bbbl@langfeatures\@empty
3390 \def\babelFSfeatures{% make sure \fontspec is redefined once
3391   \let\bbbl@ori@fontspec\fontspec
3392   \renewcommand\fontspec[1][{%
3393     \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
3394   \let\babelFSfeatures\bbbl@FSfeatures
3395   \babelFSfeatures}
3396 \def\bbbl@FSfeatures#1#2{%
3397   \expandafter\addto\csname extras#1\endcsname{%
3398     \babel@save\bbbl@langfeatures
3399     \edef\bbbl@langfeatures{#2,}}
3400 <</Font selection>>
```

14 Hooks for XeTeX and LuaTeX

14.1 XeTeX

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

`ℒTeX` sets many “codes” just before loading `hyphen.cfg`. That is not a problem in `luaTeX`, but in `xetTeX` they must be reset to the proper value. Most of the work is done in `xe(la)tex.ini`, so here we just “undo” some of the changes done by `ℒTeX`. Anyway, for consistency `LuaTeX` also resets the catcodes.

```
3401 <<*Restore Unicode catcodes before loading patterns>> ≡
3402   \begingroup
3403     % Reset chars "80-"C0 to category "other", no case mapping:
3404     \catcode\@=11 \count@=128
3405     \loop\ifnum\count@<192
3406       \global\uccode\count@=0 \global\lccode\count@=0
3407       \global\catcode\count@=12 \global\sfcode\count@=1000
3408       \advance\count@ by 1 \repeat
3409     % Other:
3410     \def\O ##1 {%
3411       \global\uccode"##1=0 \global\lccode"##1=0
3412       \global\catcode"##1=12 \global\sfcode"##1=1000 }%
3413     % Letter:
3414     \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3415       \global\uccode"##1="##2
3416       \global\lccode"##1="##3
3417       % Uppercase letters have sfcode=999:
3418       \ifnum"##1="##3 \else \global\sfcode"##1=999 \fi }%
3419     % Letter without case mappings:
3420     \def\l ##1 {\L ##1 ##1 ##1 }%
3421     \l 00AA
3422     \L 00B5 039C 00B5
3423     \l 00BA
3424     \O 00D7
3425     \l 00DF
3426     \O 00F7
3427     \L 00FF 0178 00FF
3428   \endgroup
3429   \input #1\relax
3430 <</Restore Unicode catcodes before loading patterns>>
```

Some more common code.

```
3431 <<*Footnote changes>> ≡
3432 \bbl@trace{Bidi footnotes}
3433 \ifx\bbl@beforeforeign\leavevmode
3434   \def\bbl@footnote#1#2#3{%
3435     \@ifnextchar[%
3436       {\bbl@footnote@o{#1}{#2}{#3}}%
3437       {\bbl@footnote@x{#1}{#2}{#3}}}
3438   \def\bbl@footnote@x#1#2#3#4{%
3439     \bgroup
3440     \select@language@x{\bbl@main@language}%
3441     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3442     \egroup}
3443   \def\bbl@footnote@o#1#2#3[#4]#5{%
3444     \bgroup
3445     \select@language@x{\bbl@main@language}%
3446     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3447     \egroup}
```

```

3447 \egroup}
3448 \def\bb1@footnotetext#1#2#3{%
3449 \@ifnextchar[%
3450 {\bb1@footnotetext@o{#1}{#2}{#3}}%
3451 {\bb1@footnotetext@x{#1}{#2}{#3}}}
3452 \def\bb1@footnotetext@x#1#2#3#4{%
3453 \bgroup
3454 \select@language@x{\bb1@main@language}%
3455 \bb1@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3456 \egroup}
3457 \def\bb1@footnotetext@o#1#2#3[#4]#5{%
3458 \bgroup
3459 \select@language@x{\bb1@main@language}%
3460 \bb1@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3461 \egroup}
3462 \def\BabelFootnote#1#2#3#4{%
3463 \ifx\bb1@fn@footnote\@undefined
3464 \let\bb1@fn@footnote\footnote
3465 \fi
3466 \ifx\bb1@fn@footnotetext\@undefined
3467 \let\bb1@fn@footnotetext\footnotetext
3468 \fi
3469 \bb1@ifblank{#2}%
3470 {\def#1{\bb1@footnote{\@firstofone}{#3}{#4}}
3471 \@namedef{\bb1@stripslash#1text}%
3472 {\bb1@footnotetext{\@firstofone}{#3}{#4}}}%
3473 {\def#1{\bb1@exp{\bb1@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3474 \@namedef{\bb1@stripslash#1text}%
3475 {\bb1@exp{\bb1@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3476 \fi
3477 <</Footnote changes>>

```

Now, the code.

```

3478 (*xetex)
3479 \def\BabelStringsDefault{unicode}
3480 \let\xebbl@stop\relax
3481 \AddBabelHook{xetex}{encodedcommands}{%
3482 \def\bb1@tempa{#1}%
3483 \ifx\bb1@tempa\@empty
3484 \XeTeXinputencoding"bytes"%
3485 \else
3486 \XeTeXinputencoding"#1"%
3487 \fi
3488 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3489 \AddBabelHook{xetex}{stopcommands}{%
3490 \xebbl@stop
3491 \let\xebbl@stop\relax}
3492 \def\bb1@intraspace#1 #2 #3\@@{%
3493 \bb1@csarg\gdef{\xeisp@bb1@cs{\sbcp@languagename}}%
3494 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3495 \def\bb1@intrapenalty#1\@@{%
3496 \bb1@csarg\gdef{\xeipn@bb1@cs{\sbcp@languagename}}%
3497 {\XeTeXlinebreakpenalty #1\relax}}
3498 \AddBabelHook{xetex}{loadkernel}{%
3499 <<Restore Unicode catcodes before loading patterns>>}
3500 \ifx\DisableBabelHook\@undefined\endinput\fi
3501 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
3502 \DisableBabelHook{babel-fontspec}
3503 <<Font selection>>

```

```

3504 \input txtbabel.def
3505 \xetex

```

14.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, 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.

```

3506 (*texxet)
3507 \bbl@trace{Redefinitions for bidi layout}
3508 \def\bbl@sspre@caption{%
3509   \bbl@exp{\everybox{\bbl@textdir\bbl@cs{wdir\bbl@main@language}}}}
3510 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
3511 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3512 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3513 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3514   \def\@hangfrom#1{%
3515     \setbox\@tempboxa\hbox{#1}%
3516     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3517     \noindent\box\@tempboxa}
3518 \def\raggedright{%
3519   \let\@centercr
3520   \bbl@startskip\z@skip
3521   \@rightskip\@flushglue
3522   \bbl@endskip\@rightskip
3523   \parindent\z@
3524   \parfillskip\bbl@startskip}
3525 \def\raggedleft{%
3526   \let\@centercr
3527   \bbl@startskip\@flushglue
3528   \bbl@endskip\z@skip
3529   \parindent\z@
3530   \parfillskip\bbl@endskip}
3531 \fi
3532 \IfBabelLayout{lists}
3533   {\bbl@sreplace\list
3534     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
3535     \def\bbl@listleftmargin{%
3536       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
3537     \ifcase\bbl@engine
3538       \def\labelenumii{}\theenumii{}% pdfTeX doesn't reverse ()
3539       \def\p@enumiii{\p@enumii}\theenumii{}%
3540     \fi
3541     \bbl@sreplace\@verbatim
3542       {\leftskip\@totalleftmargin}%
3543       {\bbl@startskip\textwidth
3544         \advance\bbl@startskip-\linewidth}%
3545     \bbl@sreplace\@verbatim
3546       {\rightskip\z@skip}%
3547       {\bbl@endskip\z@skip}}%
3548   {}
3549 \IfBabelLayout{contents}

```

```

3550 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
3551 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
3552 {}
3553 \IfBabelLayout{columns}
3554 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
3555 \def\bbl@outputbox#1{%
3556 \hb@xt@\textwidth{%
3557 \hskip\columnwidth
3558 \hfil
3559 {\normalcolor\vrule \@width\columnseprule}%
3560 \hfil
3561 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3562 \hskip-\textwidth
3563 \hb@xt@\columnwidth{\box\@outputbox \hss}%
3564 \hskip\columnsep
3565 \hskip\columnwidth}}}%
3566 {}
3567 <<Footnote changes>>
3568 \IfBabelLayout{footnotes}%
3569 {\BabelFootnote\footnote\language\language{}{}}%
3570 \BabelFootnote\localfootnote\language\language{}{}}%
3571 \BabelFootnote\mainfootnote{}{}{}}
3572 {}

```

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.

```

3573 \IfBabelLayout{counters}%
3574 {\let\bbl@latinarabic=\@arabic
3575 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3576 \let\bbl@asciroman=\@roman
3577 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3578 \let\bbl@asciiRoman=\@Roman
3579 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
3580 /texet

```

14.3 LuaTeX

The new loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with `luatex` patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). For the moment, a dangerous approach is used – just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

```

3581 (*luatex)
3582 \ifx\AddBabelHook\@undefined
3583 \bbl@trace{Read language.dat}
3584 \begingroup
3585 \toks@{}
3586 \count@ \z@ % 0=start, 1=0th, 2=normal
3587 \def\bbl@process@line#1#2 #3 #4 {%
3588   \ifx=#1%
3589     \bbl@process@synonym{#2}%
3590   \else
3591     \bbl@process@language{#1#2}{#3}{#4}%
3592   \fi
3593   \ignorespaces}
3594 \def\bbl@manylang{%
3595   \ifnum\bbl@last>\@ne
3596     \bbl@info{Non-standard hyphenation setup}%
3597   \fi
3598   \let\bbl@manylang\relax}
3599 \def\bbl@process@language#1#2#3{%
3600   \ifcase\count@
3601     \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
3602   \or
3603     \count@\tw@
3604   \fi
3605   \ifnum\count@=\tw@
3606     \expandafter\addlanguage\csname l@#1\endcsname
3607     \language\allocationnumber
3608     \chardef\bbl@last\allocationnumber
3609     \bbl@manylang
3610     \let\bbl@elt\relax
3611     \xdef\bbl@languages{%
3612       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3613   \fi
3614   \the\toks@
3615   \toks@{}}
3616 \def\bbl@process@synonym@aux#1#2{%
3617   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3618   \let\bbl@elt\relax
3619   \xdef\bbl@languages{%
3620     \bbl@languages\bbl@elt{#1}{#2}{}}}%
3621 \def\bbl@process@synonym#1{%
3622   \ifcase\count@
3623     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3624   \or
3625     \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
3626   \else

```

```

3627     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3628     \fi}
3629 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
3630     \chardef\l@english\z@
3631     \chardef\l@USenglish\z@
3632     \chardef\bbl@last\z@
3633     \global\@namedef{bbl@hyphendata@0}{\hyphen.tex{}}
3634     \gdef\bbl@languages{%
3635         \bbl@elt{english}{0}{\hyphen.tex}{}}%
3636         \bbl@elt{USenglish}{0}{}}{}
3637 \else
3638     \global\let\bbl@languages@format\bbl@languages
3639     \def\bbl@elt#1#2#3#4{% Remove all except language 0
3640         \ifnum#2>\z@\else
3641             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3642             \fi}%
3643     \xdef\bbl@languages{\bbl@languages}%
3644 \fi
3645 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3646 \bbl@languages
3647 \openin1=language.dat
3648 \ifeof1
3649     \bbl@warning{I couldn't find language.dat. No additional\\%
3650                 patterns loaded. Reported}%
3651 \else
3652     \loop
3653         \endlinechar\m@ne
3654         \read1 to \bbl@line
3655         \endlinechar\^^M
3656         \if T\ifeof1F\fi T\relax
3657         \ifx\bbl@line\@empty\else
3658             \edef\bbl@line{\bbl@line\space\space\space}%
3659             \expandafter\bbl@process@line\bbl@line\relax
3660         \fi
3661     \repeat
3662 \fi
3663 \endgroup
3664 \bbl@trace{Macros for reading patterns files}
3665 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
3666 \ifx\babelcatcodetablenum\@undefined
3667     \def\babelcatcodetablenum{5211}
3668 \fi
3669 \def\bbl@luapatterns#1#2{%
3670     \bbl@get@enc#1::\@@@
3671     \setbox\z@\hbox\bgroup
3672     \begin{group}
3673         \ifx\catcodetable\@undefined
3674             \let\savecatcodetable\luatexsavecatcodetable
3675             \let\initcatcodetable\luatexinitcatcodetable
3676             \let\catcodetable\luatexcatcodetable
3677         \fi
3678         \savecatcodetable\babelcatcodetablenum\relax
3679         \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3680         \catcodetable\numexpr\babelcatcodetablenum+1\relax
3681         \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3682         \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\-=13
3683         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3684         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3685         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12

```

```

3686 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
3687 \input #1\relax
3688 \catcodetable\babelcatcodetablenum\relax
3689 \endgroup
3690 \def\bbl@tempa{#2}%
3691 \ifx\bbl@tempa\@empty\else
3692 \input #2\relax
3693 \fi
3694 \egroup}%
3695 \def\bbl@patterns@lua#1{%
3696 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3697 \csname l@#1\endcsname
3698 \edef\bbl@tempa{#1}%
3699 \else
3700 \csname l@#1:\f@encoding\endcsname
3701 \edef\bbl@tempa{#1:\f@encoding}%
3702 \fi\relax
3703 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3704 \@ifundefined{bbl@hyphendata@the\language}%
3705 {\def\bbl@elt##1##2##3##4{%
3706 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3707 \def\bbl@tempb{##3}%
3708 \ifx\bbl@tempb\@empty\else % if not a synonymous
3709 \def\bbl@tempc{##3}##4}%
3710 \fi
3711 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3712 \fi}%
3713 \bbl@languages
3714 \@ifundefined{bbl@hyphendata@the\language}%
3715 {\bbl@info{No hyphenation patterns were set for\%
3716 language '\bbl@tempa'. Reported}}%
3717 {\expandafter\expandafter\expandafter\bbl@luapatterns
3718 \csname bbl@hyphendata@the\language\endcsname}}}%
3719 \endinput\fi
3720 \begingroup
3721 \catcode`\%=12
3722 \catcode`\'=12
3723 \catcode`\`=12
3724 \catcode`\:=12
3725 \directlua{
3726 Babel = Babel or {}
3727 function Babel.bytes(line)
3728 return line:gsub(".",
3729 function (chr) return unicode.utf8.char(string.byte(chr)) end)
3730 end
3731 function Babel.begin_process_input()
3732 if luatexbase and luatexbase.add_to_callback then
3733 luatexbase.add_to_callback('process_input_buffer',
3734 Babel.bytes,'Babel.bytes')
3735 else
3736 Babel.callback = callback.find('process_input_buffer')
3737 callback.register('process_input_buffer',Babel.bytes)
3738 end
3739 end
3740 function Babel.end_process_input ()
3741 if luatexbase and luatexbase.remove_from_callback then
3742 luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
3743 else
3744 callback.register('process_input_buffer',Babel.callback)

```

```

3745     end
3746 end
3747 function Babel.addpatterns(pp, lg)
3748     local lg = lang.new(lg)
3749     local pats = lang.patterns(lg) or ''
3750     lang.clear_patterns(lg)
3751     for p in pp:gmatch('[^%s]+') do
3752         ss = ''
3753         for i in string.utfcharacters(p:gsub('%d', '')) do
3754             ss = ss .. '%d?' .. i
3755         end
3756         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
3757         ss = ss:gsub('%.%d%?$', '%%.')
3758         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3759         if n == 0 then
3760             tex.sprint(
3761                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
3762                 .. p .. [[]])
3763             pats = pats .. ' ' .. p
3764         else
3765             tex.sprint(
3766                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
3767                 .. p .. [[]])
3768         end
3769     end
3770     lang.patterns(lg, pats)
3771 end
3772 }
3773 \endgroup
3774 \ifx\newattribute\@undefined\else
3775     \newattribute\bbl@attr@locale
3776     \AddBabelHook{luatex}{beforeextras}{%
3777         \setattribute\bbl@attr@locale\localeid}
3778 \fi
3779 \def\BabelStringsDefault{unicode}
3780 \let\luabbl@stop\relax
3781 \AddBabelHook{luatex}{encodedcommands}{%
3782     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3783     \ifx\bbl@tempa\bbl@tempb\else
3784         \directlua{Babel.begin_process_input()}%
3785         \def\luabbl@stop{%
3786             \directlua{Babel.end_process_input()}}%
3787     \fi}%
3788 \AddBabelHook{luatex}{stopcommands}{%
3789     \luabbl@stop
3790     \let\luabbl@stop\relax}
3791 \AddBabelHook{luatex}{patterns}{%
3792     \@ifundefined{bbl@hyphendata@the\language}%
3793     {\def\bbl@elt##1##2##3##4{%
3794         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3795         \def\bbl@tempb{##3}%
3796         \ifx\bbl@tempb\empty\else % if not a synonymous
3797             \def\bbl@tempc{##3}{##4}}%
3798         \fi
3799         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3800     \fi}%
3801     \bbl@languages
3802     \@ifundefined{bbl@hyphendata@the\language}%
3803     {\bbl@info{No hyphenation patterns were set for\%

```

```

3804         language '#2'. Reported}}%
3805     {\expandafter\expandafter\expandafter\bbbl@luapatterns
3806       \csname bbl@hyphendata@the\language\endcsname}}}%
3807 \ifundefined{bbl@patterns@}{}%
3808   \begingroup
3809     \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
3810     \ifin@else
3811       \ifx\bbbl@patterns@\@empty\else
3812         \directlua{ Babel.addpatterns(
3813           [[\bbbl@patterns@]], \number\language) }%
3814         \fi
3815       \ifundefined{bbl@patterns@#1}%
3816         \@empty
3817         {\directlua{ Babel.addpatterns(
3818           [[\space\csname bbl@patterns@#1\endcsname]],
3819           \number\language) }}%
3820       \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
3821     \fi
3822   \endgroup}}
3823 \AddBabelHook{luatex}{everylanguage}{%
3824   \def\process@language##1##2##3{%
3825     \def\process@line####1####2 ####3 ####4 {}}
3826 \AddBabelHook{luatex}{loadpatterns}{%
3827   \input #1\relax
3828   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3829     {#{1}}}}
3830 \AddBabelHook{luatex}{loadexceptions}{%
3831   \input #1\relax
3832   \def\bbbl@tempb##1##2{#{1}}{#{1}}%
3833   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3834     {\expandafter\expandafter\expandafter\bbbl@tempb
3835       \csname bbl@hyphendata@the\language\endcsname}}

```

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

```

3836 \@onlypreamble\babelpatterns
3837 \AtEndOfPackage{%
3838   \newcommand\babelpatterns[2][\@empty]{%
3839     \ifx\bbbl@patterns@\relax
3840       \let\bbbl@patterns@\@empty
3841     \fi
3842     \ifx\bbbl@pttnlist@\empty\else
3843       \bbbl@warning{%
3844         You must not intermingle \string\selectlanguage\space and\%
3845         \string\babelpatterns\space or some patterns will not\%
3846         be taken into account. Reported}%
3847       \fi
3848       \ifx@\empty#1%
3849         \protected@edef\bbbl@patterns@{\bbbl@patterns@\space#2}%
3850       \else
3851         \edef\bbbl@tempb{\zap@space#1 \@empty}%
3852         \bbbl@for\bbbl@tempa\bbbl@tempb{%
3853           \bbbl@fixname\bbbl@tempa
3854           \bbbl@iflanguage\bbbl@tempa{%
3855             \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
3856               \ifundefined{bbl@patterns@\bbbl@tempa}%
3857                 \@empty

```

```

3858         {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3859         #2}}}%
3860     \fi}}

```

14.4 Southeast Asian scripts

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).

```

3861 \def\bbl@intraspace#1 #2 #3\@@{%
3862   \directlua{
3863     Babel = Babel or {}
3864     Babel.intraspaces = Babel.intraspaces or {}
3865     Babel.intraspaces['\csname bbl@sbcpr@\language\endcsname'] = %
3866       {b = #1, p = #2, m = #3}
3867     Babel.locale_props[\the\localeid].intraspace = %
3868       {b = #1, p = #2, m = #3}
3869   }}
3870 \def\bbl@intrapenalty#1\@@{%
3871   \directlua{
3872     Babel = Babel or {}
3873     Babel.intrapenalties = Babel.intrapenalties or {}
3874     Babel.intrapenalties['\csname bbl@sbcpr@\language\endcsname'] = #1
3875     Babel.locale_props[\the\localeid].intrapenalty = #1
3876   }}
3877 \begingroup
3878 \catcode`\%=12
3879 \catcode`\^=14
3880 \catcode`\'=12
3881 \catcode`\~=12
3882 \gdef\bbl@seaintraspace{^
3883   \let\bbl@seaintraspace\relax
3884   \directlua{
3885     Babel = Babel or {}
3886     Babel.sea_enabled = true
3887     Babel.sea_ranges = Babel.sea_ranges or {}
3888     function Babel.set_chranges (script, chrng)
3889       local c = 0
3890       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
3891         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
3892         c = c + 1
3893       end
3894     end
3895     function Babel.sea_disc_to_space (head)
3896       local sea_ranges = Babel.sea_ranges
3897       local last_char = nil
3898       local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
3899       for item in node.traverse(head) do
3900         local i = item.id
3901         if i == node.id'glyph' then
3902           last_char = item
3903         elseif i == 7 and item.subtype == 3 and last_char
3904           and last_char.char > 0x0C99 then
3905           quad = font.getfont(last_char.font).size
3906           for lg, rg in pairs(sea_ranges) do
3907             if last_char.char > rg[1] and last_char.char < rg[2] then
3908               lg = lg:sub(1, 4)

```

```

3909         local intraspace = Babel.intraspaces[lg]
3910         local intrapenalty = Babel.intrapanalties[lg]
3911         local n
3912         if intrapenalty ~= 0 then
3913             n = node.new(14, 0)    ^^ penalty
3914             n.penalty = intrapenalty
3915             node.insert_before(head, item, n)
3916         end
3917         n = node.new(12, 13)    ^^ (glue, spaceskip)
3918         node.setglue(n, intraspace.b * quad,
3919                     intraspace.p * quad,
3920                     intraspace.m * quad)
3921         node.insert_before(head, item, n)
3922         node.remove(head, item)
3923     end
3924 end
3925 end
3926 end
3927 end
3928 }^^
3929 \bbl@luahyphenate}
3930 \catcode`\%=14
3931 \gdef\bbl@cjkintraspaces{%
3932   \let\bbl@cjkintraspaces\relax
3933   \directlua{
3934     Babel = Babel or {}
3935     require'babel-data-cjk.lua'
3936     Babel.cjk_enabled = true
3937     function Babel.cjk_linebreak(head)
3938       local GLYPH = node.id'glyph'
3939       local last_char = nil
3940       local quad = 655360      % 10 pt = 655360 = 10 * 65536
3941       local last_class = nil
3942       local last_lang = nil
3943
3944       for item in node.traverse(head) do
3945         if item.id == GLYPH then
3946
3947           local lang = item.lang
3948
3949           local LOCALE = node.get_attribute(item,
3950             luatexbase.registernumber'bbl@attr@locale')
3951           local props = Babel.locale_props[LOCALE]
3952
3953           class = Babel.cjk_class[item.char].c
3954
3955           if class == 'cp' then class = 'cl' end % )] as CL
3956           if class == 'id' then class = 'I' end
3957
3958           if class and last_class and Babel.cjk_breaks[last_class][class] then
3959             br = Babel.cjk_breaks[last_class][class]
3960           else
3961             br = 0
3962           end
3963
3964           if br == 1 and props.linebreak == 'c' and
3965             lang ~= \the\l@nohyphenation\space and
3966             last_lang ~= \the\l@nohyphenation then
3967             local intrapenalty = props.intrapenalty

```

```

3968         if intrapenalty ~= 0 then
3969             local n = node.new(14, 0)      % penalty
3970             n.penalty = intrapenalty
3971             node.insert_before(head, item, n)
3972         end
3973         local intraspace = props.intraspace
3974         local n = node.new(12, 13)        % (glue, spaceskip)
3975         node.setglue(n, intraspace.b * quad,
3976                       intraspace.p * quad,
3977                       intraspace.m * quad)
3978         node.insert_before(head, item, n)
3979     end
3980
3981     quad = font.getfont(item.font).size
3982     last_class = class
3983     last_lang = lang
3984     else % if penalty, glue or anything else
3985         last_class = nil
3986     end
3987 end
3988 lang.hyphenate(head)
3989 end
3990 }%
3991 \bbl@luahyphenate}
3992 \gdef\bbl@luahyphenate{%
3993 \let\bbl@luahyphenate\relax
3994 \directlua{
3995     luatexbase.add_to_callback('hyphenate',
3996     function (head, tail)
3997         if Babel.cjk_enabled then
3998             Babel.cjk_linebreak(head)
3999         end
4000         lang.hyphenate(head)
4001         if Babel.sea_enabled then
4002             Babel.sea_disc_to_space(head)
4003         end
4004     end,
4005     'Babel.hyphenate')
4006 }
4007 }
4008 \endgroup

```

14.5 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.

```

4009 \AddBabelHook{luatex}{loadkernel}{%
4010 <<Restore Unicode catcodes before loading patterns>>}
4011 \ifx\DisableBabelHook\undefined\endinput\fi
4012 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}

```



```

4013 \DisableBabelHook{babel-fontspec}
4014 <<Font selection>>

```

Temporary fix for luatex <1.10, which sometimes inserted a spurious closing dir node with a \textdir within \hboxes. This will be eventually removed.

```

4015 \def\bbl@luafixboxdir{%
4016   \setbox\z@\hbox{\textdir TLT}%
4017   \directlua{
4018     function Babel.first_dir(head)
4019       for item in node.traverse_id(node.id'dir', head) do
4020         return item
4021       end
4022       return nil
4023     end
4024     if Babel.first_dir(tex.box[0].head) then
4025       function Babel.fixboxdirs(head)
4026         local fd = Babel.first_dir(head)
4027         if fd and fd.dir:sub(1,1) == '-' then
4028           head = node.remove(head, fd)
4029         end
4030         return head
4031       end
4032     end
4033   }}
4034 \AtBeginDocument{\bbl@luafixboxdir}

```

The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.

```

4035 \newcommand\babelcharproperty[1]{%
4036   \count@=#1\relax
4037   \ifvmode
4038     \expandafter\bbl@chprop
4039   \else
4040     \bbl@error{\string\babelcharproperty\space can be used only in\\%
4041               vertical mode (preamble or between paragraphs)}%
4042     {See the manual for futher info}%
4043   \fi}
4044 \newcommand\bbl@chprop[3][\the\count@]{%
4045   \@tempcnta=#1\relax
4046   \bbl@ifunset\bbl@chprop@#2}%
4047   {\bbl@error{No property named '#2'. Allowed values are\\%
4048             direction (bc), mirror (bmg), and linebreak (lb)}%
4049     {See the manual for futher info}}%
4050   }%
4051   \loop
4052     \@nameuse\bbl@chprop@#2}{#3}%
4053   \ifnum\count@<\@tempcnta
4054     \advance\count@\@ne
4055   \repeat}
4056 \def\bbl@chprop@direction#1{%
4057   \directlua{
4058     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4059     Babel.characters[\the\count@]['d'] = '#1'
4060   }}
4061 \let\bbl@chprop@bc\bbl@chprop@direction
4062 \def\bbl@chprop@mirror#1{%
4063   \directlua{
4064     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4065     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

4066 }}
4067 \let\bbl@chprop@bmg\bbl@chprop@mirror
4068 \def\bbl@chprop@linebreak#1{%
4069   \directlua{
4070     Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4071     Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4072   }}
4073 \let\bbl@chprop@lb\bbl@chprop@linebreak

```

14.6 Layout

Work in progress.

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

4074 \bbl@trace{Redefinitions for bidi layout}
4075 \ifx\@eqnnum\@undefined\else
4076   \ifx\bbl@attr@dir\@undefined\else
4077     \edef\@eqnnum{%
4078       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
4079       \unexpanded\expandafter{\@eqnnum}}}%
4080   \fi
4081 \fi
4082 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
4083 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4084   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
4085     \bbl@exp{%
4086       \mathdir\the\bodydir
4087       #1%           Once entered in math, set boxes to restore values
4088       \<ifmmode>%
4089       \everyvbox{%
4090         \the\everyvbox
4091         \bodydir\the\bodydir
4092         \mathdir\the\mathdir
4093         \everyhbox{\the\everyhbox}%
4094         \everyvbox{\the\everyvbox}}%
4095       \everyhbox{%
4096         \the\everyhbox
4097         \bodydir\the\bodydir
4098         \mathdir\the\mathdir
4099         \everyhbox{\the\everyhbox}%
4100         \everyvbox{\the\everyvbox}}%
4101       \<fi>}}%
4102   \def\@hangfrom#1{%
4103     \setbox\@tempboxa\hbox{{#1}}%
4104     \hangindent\wd\@tempboxa
4105     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4106       \shapemode\@ne

```

```

4107 \fi
4108 \noindent\box\@tempboxa}
4109 \fi
4110 \IfBabelLayout{tabular}
4111 {\bbl@replace\@tabular{$$}{\bbl@nextfake$}%
4112 \let\bbl@tabular\@tabular
4113 \AtBeginDocument{%
4114 \ifx\bbl@tabular\@tabular\else
4115 \bbl@replace\@tabular{$$}{\bbl@nextfake$}%
4116 \fi}}
4117 {}
4118 \IfBabelLayout{lists}
4119 {\bbl@sreplace\list{\parshape}{\bbl@listparshape}%
4120 \def\bbl@listparshape#1#2#3{%
4121 \parshape #1 #2 #3 %
4122 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4123 \shapemode\tw@
4124 \fi}}
4125 {}
4126 \IfBabelLayout{graphics}
4127 {\let\bbl@pictresetdir\relax
4128 \def\bbl@pictsetdir{%
4129 \ifcase\bbl@thetextdir
4130 \let\bbl@pictresetdir\relax
4131 \else
4132 \textdir TLT\relax
4133 \def\bbl@pictresetdir{\textdir TRT\relax}%
4134 \fi}%
4135 \bbl@sreplace\@picture{\hskip-}{\bbl@pictsetdir\hskip-}%
4136 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
4137 \@killglue
4138 \raise#2\unitlength
4139 \hb@xt@\z@{\kern#1\unitlength{\bbl@pictresetdir#3}\hss}}%
4140 \AtBeginDocument
4141 {\ifx\tikz@atbegin@node\undefined\else
4142 \bbl@sreplace\pgfpicture{\pgfpicturetrue}{\bbl@pictsetdir\pgfpicturetrue}%
4143 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir}%
4144 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
4145 \fi}}
4146 {}

```

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.

```

4147 \IfBabelLayout{counters}%
4148 {\bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
4149 \let\bbl@latinarabic=\@arabic
4150 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4151 \@ifpackagewith{babel}{bidi=default}%
4152 {\let\bbl@asciroman=\@roman
4153 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4154 \let\bbl@asciiRoman=\@Roman
4155 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
4156 \def\labelenumii{}\theenumii{}%
4157 \def\p@enumiii{\p@enumii}\theenumii{}\}}%
4158 <<Footnote changes>>
4159 \IfBabelLayout{footnotes}%
4160 {\BabelFootnote\footnote\languagename{}\}%
4161 \BabelFootnote\localfootnote\languagename{}\}%

```

```

4162 \BabelFootnote\mainfootnote{}{}{}
4163 {}

```

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.

```

4164 \IfBabelLayout{extras}%
4165 {\bbl@sreplace\underline{$\@@underline}\bbl@nextfake$\@@underline}%
4166 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
4167   \if b\expandafter\@car\@series\@nil\boldmath\fi
4168   \babelsublr}%
4169   \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}}}}}
4170 {}
4171 \</luatex>

```

14.7 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's not shown here. See the generated file.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (`<l>`, `<r>` or `<al>`).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```

4172 (*basic-r)
4173 Babel = Babel or {}
4174
4175 Babel.bidi_enabled = true
4176
4177 require('babel-data-bidi.lua')
4178
4179 local characters = Babel.characters
4180 local ranges = Babel.ranges
4181
4182 local DIR = node.id("dir")

```

```

4183
4184 local function dir_mark(head, from, to, outer)
4185   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
4186   local d = node.new(DIR)
4187   d.dir = '+' .. dir
4188   node.insert_before(head, from, d)
4189   d = node.new(DIR)
4190   d.dir = '-' .. dir
4191   node.insert_after(head, to, d)
4192 end
4193
4194 function Babel.bidi(head, ispar)
4195   local first_n, last_n          -- first and last char with nums
4196   local last_es                  -- an auxiliary 'last' used with nums
4197   local first_d, last_d          -- first and last char in L/R block
4198   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):

```

4199   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
4200   local strong_lr = (strong == 'l') and 'l' or 'r'
4201   local outer = strong
4202
4203   local new_dir = false
4204   local first_dir = false
4205   local inmath = false
4206
4207   local last_lr
4208
4209   local type_n = ''
4210
4211   for item in node.traverse(head) do
4212
4213     -- three cases: glyph, dir, otherwise
4214     if item.id == node.id'glyph'
4215       or (item.id == 7 and item.subtype == 2) then
4216
4217       local itemchar
4218       if item.id == 7 and item.subtype == 2 then
4219         itemchar = item.replace.char
4220       else
4221         itemchar = item.char
4222       end
4223       local chardata = characters[itemchar]
4224       dir = chardata and chardata.d or nil
4225       if not dir then
4226         for nn, et in ipairs(ranges) do
4227           if itemchar < et[1] then
4228             break
4229           elseif itemchar <= et[2] then
4230             dir = et[3]
4231             break
4232           end
4233         end
4234       end
4235       dir = dir or 'l'
4236       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).

```

4237     if new_dir then
4238         attr_dir = 0
4239         for at in node.traverse(item.attr) do
4240             if at.number == luatexbase.registernumber'bbl@attr@dir' then
4241                 attr_dir = at.value % 3
4242             end
4243         end
4244         if attr_dir == 1 then
4245             strong = 'r'
4246         elseif attr_dir == 2 then
4247             strong = 'al'
4248         else
4249             strong = 'l'
4250         end
4251         strong_lr = (strong == 'l') and 'l' or 'r'
4252         outer = strong_lr
4253         new_dir = false
4254     end
4255
4256     if dir == 'nsm' then dir = strong end          -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

4257     dir_real = dir          -- We need dir_real to set strong below
4258     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:

```

4259     if strong == 'al' then
4260         if dir == 'en' then dir = 'an' end          -- W2
4261         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
4262         strong_lr = 'r'          -- W3
4263     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

4264     elseif item.id == node.id'dir' and not inmath then
4265         new_dir = true
4266         dir = nil
4267     elseif item.id == node.id'math' then
4268         inmath = (item.subtype == 0)
4269     else
4270         dir = nil          -- Not a char
4271     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>.

```

4272     if dir == 'en' or dir == 'an' or dir == 'et' then
4273         if dir ~= 'et' then
4274             type_n = dir

```

```

4275     end
4276     first_n = first_n or item
4277     last_n = last_es or item
4278     last_es = nil
4279     elseif dir == 'es' and last_n then -- W3+W6
4280         last_es = item
4281     elseif dir == 'cs' then             -- it's right - do nothing
4282     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
4283         if strong_lr == 'r' and type_n ~= '' then
4284             dir_mark(head, first_n, last_n, 'r')
4285         elseif strong_lr == 'l' and first_d and type_n == 'an' then
4286             dir_mark(head, first_n, last_n, 'r')
4287             dir_mark(head, first_d, last_d, outer)
4288             first_d, last_d = nil, nil
4289         elseif strong_lr == 'l' and type_n ~= '' then
4290             last_d = last_n
4291         end
4292         type_n = ''
4293         first_n, last_n = nil, nil
4294     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:

```

4295     if dir == 'l' or dir == 'r' then
4296         if dir ~= outer then
4297             first_d = first_d or item
4298             last_d = item
4299         elseif first_d and dir ~= strong_lr then
4300             dir_mark(head, first_d, last_d, outer)
4301             first_d, last_d = nil, nil
4302         end
4303     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>, resptly, 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.

```

4304     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
4305         item.char = characters[item.char] and
4306             characters[item.char].m or item.char
4307     elseif (dir or new_dir) and last_lr ~= item then
4308         local mir = outer .. strong_lr .. (dir or outer)
4309         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
4310             for ch in node.traverse(node.next(last_lr)) do
4311                 if ch == item then break end
4312                 if ch.id == node.id'glyph' then
4313                     ch.char = characters[ch.char].m or ch.char
4314                 end
4315             end
4316         end
4317     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).

```

4318     if dir == 'l' or dir == 'r' then

```

```

4319     last_lr = item
4320     strong = dir_real          -- Don't search back - best save now
4321     strong_lr = (strong == 'l') and 'l' or 'r'
4322     elseif new_dir then
4323         last_lr = nil
4324     end
4325 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

4326 if last_lr and outer == 'r' then
4327     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
4328         ch.char = characters[ch.char].m or ch.char
4329     end
4330 end
4331 if first_n then
4332     dir_mark(head, first_n, last_n, outer)
4333 end
4334 if first_d then
4335     dir_mark(head, first_d, last_d, outer)
4336 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

4337 return node.prev(head) or head
4338 end
4339 </basic-r>

```

And here the Lua code for bidi=basic:

```

4340 (*basic)
4341 Babel = Babel or {}
4342
4343 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
4344
4345 Babel.fontmap = Babel.fontmap or {}
4346 Babel.fontmap[0] = {}      -- l
4347 Babel.fontmap[1] = {}      -- r
4348 Babel.fontmap[2] = {}      -- al/an
4349
4350 Babel.bidi_enabled = true
4351
4352 require('babel-data-bidi.lua')
4353
4354 local characters = Babel.characters
4355 local ranges = Babel.ranges
4356
4357 local DIR = node.id('dir')
4358 local GLYPH = node.id('glyph')
4359
4360 local function insert_implicit(head, state, outer)
4361     local new_state = state
4362     if state.sim and state.eim and state.sim ~= state.eim then
4363         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
4364         local d = node.new(DIR)
4365         d.dir = '+' .. dir
4366         node.insert_before(head, state.sim, d)
4367         local d = node.new(DIR)
4368         d.dir = '-' .. dir
4369         node.insert_after(head, state.eim, d)
4370     end

```



```

4371 new_state.sim, new_state.eim = nil, nil
4372 return head, new_state
4373 end
4374
4375 local function insert_numeric(head, state)
4376   local new
4377   local new_state = state
4378   if state.san and state.ean and state.san ~= state.ean then
4379     local d = node.new(DIR)
4380     d.dir = '+TLT'
4381     _, new = node.insert_before(head, state.san, d)
4382     if state.san == state.sim then state.sim = new end
4383     local d = node.new(DIR)
4384     d.dir = '-TLT'
4385     _, new = node.insert_after(head, state.ean, d)
4386     if state.ean == state.eim then state.eim = new end
4387   end
4388   new_state.san, new_state.ean = nil, nil
4389   return head, new_state
4390 end
4391
4392 -- TODO - \hbox with an explicit dir can lead to wrong results
4393 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
4394 -- was made to improve the situation, but the problem is the 3-dir
4395 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
4396 -- well.
4397
4398 function Babel.bidi(head, ispar, hdir)
4399   local d -- d is used mainly for computations in a loop
4400   local prev_d = ''
4401   local new_d = false
4402
4403   local nodes = {}
4404   local outer_first = nil
4405   local inmath = false
4406
4407   local glue_d = nil
4408   local glue_i = nil
4409
4410   local has_en = false
4411   local first_et = nil
4412
4413   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
4414
4415   local save_outer
4416   local temp = node.get_attribute(head, ATDIR)
4417   if temp then
4418     temp = temp % 3
4419     save_outer = (temp == 0 and 'l') or
4420                  (temp == 1 and 'r') or
4421                  (temp == 2 and 'al')
4422   elseif ispar then -- Or error? Shouldn't happen
4423     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
4424   else -- Or error? Shouldn't happen
4425     save_outer = ('TRT' == hdir) and 'r' or 'l'
4426   end
4427   -- when the callback is called, we are just _after_ the box,
4428   -- and the textdir is that of the surrounding text
4429   -- if not ispar and hdir ~= tex.textdir then

```

```

4430 -- save_outer = ('TRT' == hdir) and 'r' or 'l'
4431 -- end
4432 local outer = save_outer
4433 local last = outer
4434 -- 'al' is only taken into account in the first, current loop
4435 if save_outer == 'al' then save_outer = 'r' end
4436
4437 local fontmap = Babel.fontmap
4438
4439 for item in node.traverse(head) do
4440
4441     -- In what follows, #node is the last (previous) node, because the
4442     -- current one is not added until we start processing the neutrals.
4443
4444     -- three cases: glyph, dir, otherwise
4445     if item.id == GLYPH
4446         or (item.id == 7 and item.subtype == 2) then
4447
4448         local d_font = nil
4449         local item_r
4450         if item.id == 7 and item.subtype == 2 then
4451             item_r = item.replace -- automatic discs have just 1 glyph
4452         else
4453             item_r = item
4454         end
4455         local chardata = characters[item_r.char]
4456         d = chardata and chardata.d or nil
4457         if not d or d == 'nsm' then
4458             for nn, et in ipairs(ranges) do
4459                 if item_r.char < et[1] then
4460                     break
4461                 elseif item_r.char <= et[2] then
4462                     if not d then d = et[3]
4463                     elseif d == 'nsm' then d_font = et[3]
4464                     end
4465                     break
4466                 end
4467             end
4468         end
4469         d = d or 'l'
4470
4471         -- A short 'pause' in bidi for mapfont
4472         d_font = d_font or d
4473         d_font = (d_font == 'l' and 0) or
4474             (d_font == 'nsm' and 0) or
4475             (d_font == 'r' and 1) or
4476             (d_font == 'al' and 2) or
4477             (d_font == 'an' and 2) or nil
4478         if d_font and fontmap and fontmap[d_font][item_r.font] then
4479             item_r.font = fontmap[d_font][item_r.font]
4480         end
4481
4482         if new_d then
4483             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4484             if inmath then
4485                 attr_d = 0
4486             else
4487                 attr_d = node.get_attribute(item, ATDIR)
4488                 attr_d = attr_d % 3

```

```

4489         end
4490         if attr_d == 1 then
4491             outer_first = 'r'
4492             last = 'r'
4493         elseif attr_d == 2 then
4494             outer_first = 'r'
4495             last = 'al'
4496         else
4497             outer_first = 'l'
4498             last = 'l'
4499         end
4500         outer = last
4501         has_en = false
4502         first_et = nil
4503         new_d = false
4504     end
4505
4506     if glue_d then
4507         if (d == 'l' and 'l' or 'r') ~= glue_d then
4508             table.insert(nodes, {glue_i, 'on', nil})
4509         end
4510         glue_d = nil
4511         glue_i = nil
4512     end
4513
4514     elseif item.id == DIR then
4515         d = nil
4516         new_d = true
4517
4518     elseif item.id == node.id'glue' and item.subtype == 13 then
4519         glue_d = d
4520         glue_i = item
4521         d = nil
4522
4523     elseif item.id == node.id'math' then
4524         inmath = (item.subtype == 0)
4525
4526     else
4527         d = nil
4528     end
4529
4530     -- AL <= EN/ET/ES      -- W2 + W3 + W6
4531     if last == 'al' and d == 'en' then
4532         d = 'an'          -- W3
4533     elseif last == 'al' and (d == 'et' or d == 'es') then
4534         d = 'on'          -- W6
4535     end
4536
4537     -- EN + CS/ES + EN      -- W4
4538     if d == 'en' and #nodes >= 2 then
4539         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4540             and nodes[#nodes-1][2] == 'en' then
4541             nodes[#nodes][2] = 'en'
4542         end
4543     end
4544
4545     -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
4546     if d == 'an' and #nodes >= 2 then
4547         if (nodes[#nodes][2] == 'cs')

```

```

4548         and nodes[#nodes-1][2] == 'an' then
4549             nodes[#nodes][2] = 'an'
4550         end
4551     end
4552
4553     -- ET/EN          -- W5 + W7->l / W6->on
4554     if d == 'et' then
4555         first_et = first_et or (#nodes + 1)
4556     elseif d == 'en' then
4557         has_en = true
4558         first_et = first_et or (#nodes + 1)
4559     elseif first_et then      -- d may be nil here !
4560         if has_en then
4561             if last == 'l' then
4562                 temp = 'l'      -- W7
4563             else
4564                 temp = 'en'     -- W5
4565             end
4566         else
4567             temp = 'on'        -- W6
4568         end
4569         for e = first_et, #nodes do
4570             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4571         end
4572         first_et = nil
4573         has_en = false
4574     end
4575
4576     if d then
4577         if d == 'al' then
4578             d = 'r'
4579             last = 'al'
4580         elseif d == 'l' or d == 'r' then
4581             last = d
4582         end
4583         prev_d = d
4584         table.insert(nodes, {item, d, outer_first})
4585     end
4586
4587     outer_first = nil
4588
4589 end
4590
4591 -- TODO -- repeated here in case EN/ET is the last node. Find a
4592 -- better way of doing things:
4593 if first_et then      -- dir may be nil here !
4594     if has_en then
4595         if last == 'l' then
4596             temp = 'l'      -- W7
4597         else
4598             temp = 'en'     -- W5
4599         end
4600     else
4601         temp = 'on'        -- W6
4602     end
4603     for e = first_et, #nodes do
4604         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4605     end
4606 end

```

```

4607
4608 -- dummy node, to close things
4609 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4610
4611 ----- NEUTRAL -----
4612
4613 outer = save_outer
4614 last = outer
4615
4616 local first_on = nil
4617
4618 for q = 1, #nodes do
4619     local item
4620
4621     local outer_first = nodes[q][3]
4622     outer = outer_first or outer
4623     last = outer_first or last
4624
4625     local d = nodes[q][2]
4626     if d == 'an' or d == 'en' then d = 'r' end
4627     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4628
4629     if d == 'on' then
4630         first_on = first_on or q
4631     elseif first_on then
4632         if last == d then
4633             temp = d
4634         else
4635             temp = outer
4636         end
4637         for r = first_on, q - 1 do
4638             nodes[r][2] = temp
4639             item = nodes[r][1] -- MIRRORING
4640             if item.id == GLYPH and temp == 'r' then
4641                 item.char = characters[item.char].m or item.char
4642             end
4643         end
4644         first_on = nil
4645     end
4646
4647     if d == 'r' or d == 'l' then last = d end
4648 end
4649
4650 ----- IMPLICIT, REORDER -----
4651
4652 outer = save_outer
4653 last = outer
4654
4655 local state = {}
4656 state.has_r = false
4657
4658 for q = 1, #nodes do
4659
4660     local item = nodes[q][1]
4661
4662     outer = nodes[q][3] or outer
4663
4664     local d = nodes[q][2]
4665

```

```

4666   if d == 'nsm' then d = last end           -- W1
4667   if d == 'en' then d = 'an' end
4668   local isdir = (d == 'r' or d == 'l')
4669
4670   if outer == 'l' and d == 'an' then
4671     state.san = state.san or item
4672     state.ean = item
4673   elseif state.san then
4674     head, state = insert_numeric(head, state)
4675   end
4676
4677   if outer == 'l' then
4678     if d == 'an' or d == 'r' then           -- im -> implicit
4679       if d == 'r' then state.has_r = true end
4680       state.sim = state.sim or item
4681       state.eim = item
4682     elseif d == 'l' and state.sim and state.has_r then
4683       head, state = insert_implicit(head, state, outer)
4684     elseif d == 'l' then
4685       state.sim, state.eim, state.has_r = nil, nil, false
4686     end
4687   else
4688     if d == 'an' or d == 'l' then
4689       if nodes[q][3] then -- nil except after an explicit dir
4690         state.sim = item -- so we move sim 'inside' the group
4691       else
4692         state.sim = state.sim or item
4693       end
4694       state.eim = item
4695     elseif d == 'r' and state.sim then
4696       head, state = insert_implicit(head, state, outer)
4697     elseif d == 'r' then
4698       state.sim, state.eim = nil, nil
4699     end
4700   end
4701
4702   if isdir then
4703     last = d           -- Don't search back - best save now
4704   elseif d == 'on' and state.san then
4705     state.san = state.san or item
4706     state.ean = item
4707   end
4708
4709 end
4710
4711 return node.prev(head) or head
4712 end
4713 </basic>

```

15 Data for CJK

It is a boring file and it's not shown here. See the generated file.

16 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.

```
4714 ⟨*nil⟩
4715 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
4716 \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.

```
4717 \ifx\l@nil\undefined
4718   \newlanguage\l@nil
4719   \@namedef{bbl@hyphendata@the\l@nil}{{}}{}% Remove warning
4720 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
4721 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil
4722 \let\captionnil\@empty
4723 \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.

```
4724 \ldf@finish{nil}
4725 ⟨/nil⟩
```

17 Support for Plain \TeX (plain.def)

17.1 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 `locallyphen.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 `ini \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 `ini \TeX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
4726 ⟨*bplain | blplain⟩
4727 \catcode`\{=1 % left brace is begin-group character
4728 \catcode`\}=2 % right brace is end-group character
4729 \catcode`\#=6 % hash mark is macro parameter character
```

Now let's see if a file called `hyphen.cfg` can be found somewhere on \TeX 's input path by trying to open it for reading...

```
4730 \openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
4731 \ifeof0
```

```
4732 \else
```

When `hyphen.cfg` could be opened we make sure that *it* will be read instead of the file `hyphen.tex` which should (according to Don Knuth's ruling) contain the american English hyphenation patterns and nothing else.

We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
4733 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
4734 \def\input #1 {%
```

```
4735 \let\input\input
```

```
4736 \input hyphen.cfg
```

Once that's done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
4737 \let\input\undefined
```

```
4738 }
```

```
4739 \fi
```

```
4740 </bplain | bplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
4741 <bplain>\input plain.tex
```

```
4742 <bplain>\input 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.

```
4743 <bplain>\def\fmtname{babel-plain}
```

```
4744 <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.

17.2 Emulating some \TeX features

The following code duplicates or emulates parts of $\TeX 2_{\epsilon}$ that are needed for `babel`.

```
4745 <*plain>
```

```
4746 \def\@empty{}
```

```
4747 \def\loadlocalcfg#1{%
```

```
4748 \openin0#1.cfg
```

```
4749 \ifeof0
```

```
4750 \closein0
```

```
4751 \else
```

```
4752 \closein0
```

```
4753 {\immediate\write16{*****}%
```

```
4754 \immediate\write16{* Local config file #1.cfg used}%
```

```
4755 \immediate\write16{*}%
```

```
4756 }
```

```
4757 \input #1.cfg\relax
```

```
4758 \fi
```

```
4759 \@endofldf}
```


17.3 General tools

A number of \LaTeX macro's that are needed later on.

```
4760 \long\def\@firstofone#1{#1}
4761 \long\def\@firstoftwo#1#2{#1}
4762 \long\def\@secondoftwo#1#2{#2}
4763 \def\@nnil{\@nil}
4764 \def\@gobbletwo#1#2{}
4765 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4766 \def\@star@or@long#1{%
4767   \@ifstar
4768   {\let\l@ngrel@x\relax#1}%
4769   {\let\l@ngrel@x\long#1}}
4770 \let\l@ngrel@x\relax
4771 \def\@car#1#2\@nil{#1}
4772 \def\@cdr#1#2\@nil{#2}
4773 \let\@typeset@protect\relax
4774 \let\protected@edef\edef
4775 \long\def\@gobble#1{}
4776 \edef\@backslashchar{\expandafter\@gobble\string\}
4777 \def\strip@prefix#1>{}
4778 \def\g@addto@macro#1#2{%
4779   \toks@\expandafter{#1#2}%
4780   \xdef#1{\the\toks@}}
4781 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4782 \def\@nameuse#1{\csname #1\endcsname}
4783 \def\@ifundefined#1{%
4784   \expandafter\ifx\csname#1\endcsname\relax
4785     \expandafter\@firstoftwo
4786     \else
4787       \expandafter\@secondoftwo
4788     \fi}
4789 \def\@expandtwoargs#1#2#3{%
4790   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4791 \def\zap@space#1 #2{%
4792   #1%
4793   \ifx#2\@empty\else\expandafter\zap@space\fi
4794   #2}
```

$\LaTeX 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
4795 \ifx\@preamblecmds\@undefined
4796   \def\@preamblecmds{}
4797 \fi
4798 \def\@onlypreamble#1{%
4799   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4800     \@preamblecmds\do#1}}
4801 \@onlypreamble\@onlypreamble
```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```
4802 \def\begindocument{%
4803   \@begindocumenthook
4804   \global\let\@begindocumenthook\@undefined
4805   \def\do##1{\global\let##1\@undefined}%
4806   \@preamblecmds
4807   \global\let\do\noexpand}
4808 \ifx\@begindocumenthook\@undefined
```

```

4809 \def\@begindocumenthook{}
4810 \fi
4811 \@onlypreamble\@begindocumenthook
4812 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimic L^AT_EX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```

4813 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
4814 \@onlypreamble\AtEndOfPackage
4815 \def\@endofldf{}
4816 \@onlypreamble\@endofldf
4817 \let\bbl@afterlang\@empty
4818 \chardef\bbl@opt@hyphenmap\z@

```

L^AT_EX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

4819 \ifx\if@files\@undefined
4820 \expandafter\let\csname if@files\expandafter\endcsname
4821 \csname iffalse\endcsname
4822 \fi

```

Mimick L^AT_EX's commands to define control sequences.

```

4823 \def\newcommand{\@star@or@long\new@command}
4824 \def\new@command#1{%
4825   \@testopt{\@newcommand#1}0}
4826 \def\@newcommand#1[#2]{%
4827   \@ifnextchar [{\@xargdef#1[#2]}%
4828                 {\@argdef#1[#2]}}
4829 \long\def\@argdef#1[#2]#3{%
4830   \@yargdef#1\@ne{#2}{#3}}
4831 \long\def\@xargdef#1[#2][#3]#4{%
4832   \expandafter\def\expandafter#1\expandafter{%
4833     \expandafter\@protected@testopt\expandafter #1%
4834     \csname\string#1\expandafter\endcsname{#3}}%
4835   \expandafter\@yargdef \csname\string#1\endcsname
4836   \tw@{#2}{#4}}
4837 \long\def\@yargdef#1#2#3{%
4838   \@tempcnta#3\relax
4839   \advance \@tempcnta \@ne
4840   \let\@hash@\relax
4841   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4842   \@tempcntb #2%
4843   \@whilenum \@tempcntb < \@tempcnta
4844   \do{%
4845     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
4846     \advance\@tempcntb \@ne}%
4847   \let\@hash@###
4848   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
4849 \def\providecommand{\@star@or@long\provide@command}
4850 \def\provide@command#1{%
4851   \begingroup
4852   \escapechar\m@ne\xdef\@gtempa{\string#1}%
4853   \endgroup
4854   \expandafter\ifundefined\@gtempa
4855     {\def\reserved@a{\newcommand#1}}%
4856     {\let\reserved@a\relax
4857      \def\reserved@a{\newcommand\reserved@a}}%
4858   \reserved@a}%

```

```

4859 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4860 \def\declare@robustcommand#1{%
4861   \edef\reserved@a{\string#1}%
4862   \def\reserved@b{#1}%
4863   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4864   \edef#1{%
4865     \ifx\reserved@a\reserved@b
4866       \noexpand\x@protect
4867       \noexpand#1%
4868     \fi
4869     \noexpand\protect
4870     \expandafter\noexpand\csname
4871       \expandafter\@gobble\string#1 \endcsname
4872   }%
4873   \expandafter\new@command\csname
4874     \expandafter\@gobble\string#1 \endcsname
4875 }
4876 \def\x@protect#1{%
4877   \ifx\protect\@typeset@protect\else
4878     \@x@protect#1%
4879   \fi
4880 }
4881 \def\@x@protect#1\fi#2#3{%
4882   \fi\protect#1%
4883 }

```

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`.

```

4884 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4885 \ifx\in@\@undefined
4886   \def\in@#1#2{%
4887     \def\in@@##1#1##2##3\in@@{%
4888       \ifx\in@##2\in@false\else\in@true\fi}%
4889     \in@@#2#1\in@\in@@}
4890 \else
4891   \let\bbl@tempa\@empty
4892 \fi
4893 \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).

```

4894 \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.

```

4895 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their \LaTeX 2_ϵ versions; just enough to make things work in plain \TeX environments.

```

4896 \ifx\@tempcnta\@undefined
4897   \csname newcount\endcsname\@tempcnta\relax

```

```

4898 \fi
4899 \ifx\@tempcntb\@undefined
4900   \csname newcount\endcsname\@tempcntb\relax
4901 \fi

```

To prevent wasting two counters in L^AT_EX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

```

4902 \ifx\bye\@undefined
4903   \advance\count10 by -2\relax
4904 \fi
4905 \ifx\@ifnextchar\@undefined
4906   \def\@ifnextchar#1#2#3{%
4907     \let\reserved@d=#1%
4908     \def\reserved@a{#2}\def\reserved@b{#3}%
4909     \futurelet\@let@token\@ifnch}
4910   \def\@ifnch{%
4911     \ifx\@let@token\@sptoken
4912       \let\reserved@c\@xifnch
4913     \else
4914       \ifx\@let@token\reserved@d
4915         \let\reserved@c\reserved@a
4916       \else
4917         \let\reserved@c\reserved@b
4918       \fi
4919     \fi
4920     \reserved@c}
4921   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
4922   \def\:\@xifnch\expandafter\def\:{\futurelet\@let@token\@ifnch}
4923 \fi
4924 \def\@testopt#1#2{%
4925   \@ifnextchar[#{1}{#1[#2]}}
4926 \def\@protected@testopt#1{%
4927   \ifx\protect\@typeset@protect
4928     \expandafter\@testopt
4929   \else
4930     \@x@protect#1%
4931   \fi}
4932 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4933   #2\relax}\fi}
4934 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4935   \else\expandafter\@gobble\fi{#1}}

```

17.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain T_EX environment.

```

4936 \def\DeclareTextCommand{%
4937   \@dec@text@cmd\providecommand
4938 }
4939 \def\ProvideTextCommand{%
4940   \@dec@text@cmd\providecommand
4941 }
4942 \def\DeclareTextSymbol#1#2#3{%
4943   \@dec@text@cmd\chardef#1{#2}#3\relax
4944 }
4945 \def\@dec@text@cmd#1#2#3{%
4946   \expandafter\def\expandafter#2%
4947     \expandafter{%
4948       \csname#3-cmd\expandafter\endcsname

```

```

4949         \expandafter#2%
4950         \csname#3\string#2\endcsname
4951     }%
4952 %    \let\@ifdefinable\@rc@ifdefinable
4953     \expandafter#1\csname#3\string#2\endcsname
4954 }
4955 \def\@current@cmd#1{%
4956     \ifx\protect\@typeset@protect\else
4957         \noexpand#1\expandafter\@gobble
4958     \fi
4959 }
4960 \def\@changed@cmd#1#2{%
4961     \ifx\protect\@typeset@protect
4962         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4963             \expandafter\ifx\csname ?\string#1\endcsname\relax
4964                 \expandafter\def\csname ?\string#1\endcsname{%
4965                     \@changed@x@err{#1}%
4966                 }%
4967             \fi
4968             \global\expandafter\let
4969                 \csname\cf@encoding \string#1\expandafter\endcsname
4970                 \csname ?\string#1\endcsname
4971             \fi
4972             \csname\cf@encoding\string#1%
4973                 \expandafter\endcsname
4974         \else
4975             \noexpand#1%
4976         \fi
4977 }
4978 \def\@changed@x@err#1{%
4979     \errhelp{Your command will be ignored, type <return> to proceed}%
4980     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4981 \def\DeclareTextCommandDefault#1{%
4982     \DeclareTextCommand#1?%
4983 }
4984 \def\ProvideTextCommandDefault#1{%
4985     \ProvideTextCommand#1?%
4986 }
4987 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
4988 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4989 \def\DeclareTextAccent#1#2#3{%
4990     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
4991 }
4992 \def\DeclareTextCompositeCommand#1#2#3#4{%
4993     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
4994     \edef\reserved@b{\string##1}%
4995     \edef\reserved@c{%
4996         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
4997     \ifx\reserved@b\reserved@c
4998         \expandafter\expandafter\expandafter\ifx
4999             \expandafter\@car\reserved@a\relax\relax\nil
5000             \@text@composite
5001         \else
5002             \edef\reserved@b##1{%
5003                 \def\expandafter\noexpand
5004                     \csname#2\string#1\endcsname####1{%
5005                     \noexpand\@text@composite
5006                     \expandafter\noexpand\csname#2\string#1\endcsname
5007                     ####1\noexpand\empty\noexpand\@text@composite

```

```

5008         {##1}%
5009     }%
5010 }%
5011 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
5012 \fi
5013 \expandafter\def\csname\expandafter\string\csname
5014     #2\endcsname\string#1-\string#3\endcsname{#4}
5015 \else
5016     \errhelp{Your command will be ignored, type <return> to proceed}%
5017     \errmessage{\string\DeclareTextCompositeCommand\space used on
5018         inappropriate command \protect#1}
5019 \fi
5020 }
5021 \def\@text@composite#1#2#3\@text@composite{%
5022     \expandafter\@text@composite@x
5023     \csname\string#1-\string#2\endcsname
5024 }
5025 \def\@text@composite@x#1#2{%
5026     \ifx#1\relax
5027         #2%
5028     \else
5029         #1%
5030     \fi
5031 }
5032 %
5033 \def\@strip@args#1:#2-#3\@strip@args{#2}
5034 \def\DeclareTextComposite#1#2#3#4{%
5035     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
5036     \bgroup
5037         \lcode`\@=#4%
5038         \lowercase{%
5039     \egroup
5040         \reserved@a @%
5041     }%
5042 }
5043 %
5044 \def\UseTextSymbol#1#2{%
5045 %     \let\@curr@enc\cf@encoding
5046 %     \@use@text@encoding{#1}%
5047     #2%
5048 %     \@use@text@encoding\@curr@enc
5049 }
5050 \def\UseTextAccent#1#2#3{%
5051 %     \let\@curr@enc\cf@encoding
5052 %     \@use@text@encoding{#1}%
5053 %     #2{\@use@text@encoding\@curr@enc\selectfont#3}%
5054 %     \@use@text@encoding\@curr@enc
5055 }
5056 \def\@use@text@encoding#1{%
5057 %     \edef\font@name{#1}%
5058 %     \xdef\font@name{%
5059 %         \csname\curr@fontshape/\font@size\endcsname
5060 %     }%
5061 %     \pickup@font
5062 %     \font@name
5063 %     \@@enc@update
5064 }
5065 \def\DeclareTextSymbolDefault#1#2{%
5066     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%

```

```

5067 }
5068 \def\DeclareTextAccentDefault#1#2{%
5069   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
5070 }
5071 \def\cf@encoding{OT1}

```

Currently we only use the $\text{\LaTeX} 2_{\epsilon}$ method for accents for those that are known to be made active in *some* language definition file.

```

5072 \DeclareTextAccent{"}{OT1}{127}
5073 \DeclareTextAccent{'}{OT1}{19}
5074 \DeclareTextAccent{^}{OT1}{94}
5075 \DeclareTextAccent{`}{OT1}{18}
5076 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for plain \TeX .

```

5077 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
5078 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
5079 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
5080 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
5081 \DeclareTextSymbol{\i}{OT1}{16}
5082 \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`.

```

5083 \ifx\scriptsize\@undefined
5084   \let\scriptsize\sevenrm
5085 \fi
5086 </plain>

```

18 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] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [3] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [4] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).
- [5] Hubert Partl, *German \TeX , TUGboat 9 (1988) #1*, p. 70–72.
- [6] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [7] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

- [8] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [9] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [10] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [11] Joachim Schrod, *International \TeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, p. 301–373.