

Babel

Version 3.25

2018/10/03

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 version 3 and, to some extent, 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 and the like will be 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 lualatex	6
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	23
1.15	Modifying a language	25
1.16	Creating a language	25
1.17	Digits	28
1.18	Getting the current language name	28
1.19	Hyphenation tools	28
1.20	Selecting scripts	30
1.21	Selecting directions	31
1.22	Language attributes	34
1.23	Hooks	34
1.24	Languages supported by babel with ldf files	36
1.25	Tips, workarounds, know issues and notes	37
1.26	Current and future work	38
1.27	Tentative and experimental code	38
2	Loading languages with language.dat	39
2.1	Format	39
3	The interface between the core of babel and the language definition files	40
3.1	Guidelines for contributed languages	41
3.2	Basic macros	42
3.3	Skeleton	43
3.4	Support for active characters	44
3.5	Support for saving macro definitions	44
3.6	Support for extending macros	45
3.7	Macros common to a number of languages	45
3.8	Encoding-dependent strings	45
4	Changes	49
4.1	Changes in babel version 3.9	49
II	The code	49
5	Identification and loading of required files	50
6	locale directory	50

7	Tools	50
7.1	Multiple languages	54
8	The Package File (\LaTeX, babel.sty)	55
8.1	base	55
8.2	key=value options and other general option	57
8.3	Conditional loading of shorthands	58
8.4	Language options	59
9	The kernel of Babel (babel.def, common)	62
9.1	Tools	62
9.2	Hooks	65
9.3	Setting up language files	67
9.4	Shorthands	68
9.5	Language attributes	78
9.6	Support for saving macro definitions	80
9.7	Short tags	81
9.8	Hyphens	81
9.9	Multiencoding strings	83
9.10	Macros common to a number of languages	89
9.11	Making glyphs available	89
9.11.1	Quotation marks	89
9.11.2	Letters	90
9.11.3	Shorthands for quotation marks	91
9.11.4	Umlauts and tremas	92
9.12	Layout	93
9.13	Creating languages	94
10	The kernel of Babel (babel.def, only \LaTeX)	103
10.1	The redefinition of the style commands	103
10.2	Cross referencing macros	103
10.3	Marks	106
10.4	Preventing clashes with other packages	107
10.4.1	ifthen	107
10.4.2	varioref	108
10.4.3	hhline	109
10.4.4	hyperref	109
10.4.5	fancyhdr	109
10.5	Encoding and fonts	110
10.6	Basic bidi support	111
10.7	Local Language Configuration	114
11	Multiple languages (switch.def)	115
11.1	Selecting the language	116
11.2	Errors	123
12	Loading hyphenation patterns	125
13	Font handling with fontspec	130
14	Hooks for XeTeX and LuaTeX	133
14.1	XeTeX	133
14.2	Layout	135
14.3	LuaTeX	138

15	Southeast Asian scripts	143
15.1	Layout	145
15.2	Auto bidi with basic-r	147
16	The ‘nil’ language	157
17	Support for Plain T_EX (plain.def)	158
17.1	Not renaming hyphen.tex	158
17.2	Emulating some L ^A T _E X features	159
17.3	General tools	159
17.4	Encoding related macros	163
18	Acknowledgements	166

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’	7
Argument of \language@active@arg” has an extra }	11

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 on <https://github.com/latex3/latex2e/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 on <https://github.com/latex3/latex2e/tree/master/required/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 \LaTeX , the preamble of the document:

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

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

You can also set the main language explicitly:

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

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

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

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

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

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

```
\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.

1.4 xelatex and lualatex

Many languages are compatible with `xetex` and `lualatex`. 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.

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

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 \babel font 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 L^AT_EX (ie, \usepackage{<language>}) is deprecated and you will get the error:²

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

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

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

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


```
! 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:

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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 discretionaries and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbccode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

NOTE Note the following:

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

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 `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.

silent New 3.9l No warnings and no *infos* are written to the log file.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 `\MakeUppercase` 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 `\begin{document}`), but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated;¹¹

select sets it only at `\selectlanguage`;

other also sets it at `otherlanguage`;

other* also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.¹²

bidi= default | basic | basic-r

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}
```

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

af	Afrikaans ^{ul}	es	Spanish ^{ul}
agq	Aghem	et	Estonian ^{ul}
ak	Akan	eu	Basque ^{ul}
am	Amharic ^{ul}	ewo	Ewondo
ar	Arabic ^{ul}	fa	Persian ^{ul}
ar-DZ	Arabic ^{ul}	ff	Fulah
ar-MA	Arabic ^{ul}	fi	Finnish ^{ul}
ar-SY	Arabic ^{ul}	fil	Filipino
as	Assamese	fo	Faroese
asa	Asu	fr	French ^{ul}
ast	Asturian ^{ul}	fr-BE	French ^{ul}
az-Cyrl	Azerbaijani	fr-CA	French ^{ul}
az-Latn	Azerbaijani	fr-CH	French ^{ul}
az	Azerbaijani ^{ul}	fr-LU	French ^{ul}
bas	Basaa	fur	Friulian ^{ul}
be	Belarusian ^{ul}	fy	Western Frisian
bem	Bemba	ga	Irish ^{ul}
bez	Bena	gd	Scottish Gaelic ^{ul}
bg	Bulgarian ^{ul}	gl	Galician ^{ul}
bm	Bambara	gsw	Swiss German
bn	Bangla ^{ul}	gu	Gujarati
bo	Tibetan ^u	guz	Gusii
brx	Bodo	gv	Manx
bs-Cyrl	Bosnian	ha-GH	Hausa
bs-Latn	Bosnian ^{ul}	ha-NE	Hausa ^l
bs	Bosnian ^{ul}	ha	Hausa
ca	Catalan ^{ul}	haw	Hawaiian
ce	Chechen	he	Hebrew ^{ul}
cgg	Chiga	hi	Hindi ^u
chr	Cherokee	hr	Croatian ^{ul}
ckb	Central Kurdish	hsb	Upper Sorbian ^{ul}
cs	Czech ^{ul}	hu	Hungarian ^{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

kn	Kannada ^{ul}	pl	Polish ^{ul}
ko	Korean	pms	Piedmontese ^{ul}
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese ^{ul}
ksb	Shambala	pt-PT	Portuguese ^{ul}
ksf	Bafia	pt	Portuguese ^{ul}
ksh	Colognian	qu	Quechua
kw	Cornish	rm	Romansh ^{ul}
ky	Kyrgyz	rn	Rundi
lag	Langi	ro	Romanian ^{ul}
lb	Luxembourgish	rof	Rombo
lg	Ganda	ru	Russian ^{ul}
lkt	Lakota	rw	Kinyarwanda
ln	Lingala	rwk	Rwa
lo	Lao ^{ul}	sa-Beng	Sanskrit
lrc	Northern Luri	sa-Deva	Sanskrit
lt	Lithuanian ^{ul}	sa-Gujr	Sanskrit
lu	Luba-Katanga	sa-Knda	Sanskrit
luo	Luo	sa-Mlym	Sanskrit
luy	Luyia	sa-Telu	Sanskrit
lv	Latvian ^{ul}	sa	Sanskrit
mas	Masai	sah	Sakha
mer	Meru	saq	Samburu
mfe	Morisyen	sbp	Sangu
mg	Malagasy	se	Northern Sami ^{ul}
mgf	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian ^{ul}	sg	Sango
ml	Malayalam ^{ul}	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi ^{ul}	shi	Tachelhit
ms-BN	Malay ^l	si	Sinhala
ms-SG	Malay ^l	sk	Slovak ^{ul}
ms	Malay ^{ul}	sl	Slovenian ^{ul}
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian ^{ul}
naq	Nama	sr-Cyrl-BA	Serbian ^{ul}
nb	Norwegian Bokmål ^{ul}	sr-Cyrl-ME	Serbian ^{ul}
nd	North Ndebele	sr-Cyrl-XK	Serbian ^{ul}
ne	Nepali	sr-Cyrl	Serbian ^{ul}
nl	Dutch ^{ul}	sr-Latn-BA	Serbian ^{ul}
nmg	Kwasio	sr-Latn-ME	Serbian ^{ul}
nn	Norwegian Nynorsk ^{ul}	sr-Latn-XK	Serbian ^{ul}
nnh	Ngiemboon	sr-Latn	Serbian ^{ul}
nus	Nuer	sr	Serbian ^{ul}
nyn	Nyankole	sv	Swedish ^{ul}
om	Oromo	sw	Swahili
or	Odia	ta	Tamil ^u
os	Ossetic	te	Telugu ^{ul}
pa-Arab	Punjabi	teo	Teso
pa-Guru	Punjabi	th	Thai ^{ul}
pa	Punjabi	ti	Tigrinya

tk	Turkmen ^{ul}	wae	Walser
to	Tongan	xog	Soga
tr	Turkish ^{ul}	yav	Yangben
twq	Tasawaq	yi	Yiddish
tzm	Central Atlas Tamazight	yo	Yoruba
ug	Uyghur	yue	Cantonese
uk	Ukrainian ^{ul}	zgh	Standard Moroccan Tamazight
ur	Urdu ^{ul}		
uz-Arab	Uzbek	zh-Hans-HK	Chinese
uz-Cyrl	Uzbek	zh-Hans-MO	Chinese
uz-Latn	Uzbek	zh-Hans-SG	Chinese
uz	Uzbek	zh-Hans	Chinese
vai-Latn	Vai	zh-Hant-HK	Chinese
vai-Vaii	Vai	zh-Hant-MO	Chinese
vai	Vai	zh-Hant	Chinese
vi	Vietnamese ^{ul}	zh	Chinese
vun	Vunjo	zu	Zulu

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

aghem	bemba
akan	ben
albanian	bengali
american	bodo
amharic	bosnian-cyrillic
arabic	bosnian-cyrl
arabic-algeria	bosnian-latin
arabic-DZ	bosnian-latn
arabic-morocco	bosnian
arabic-MA	brazilian
arabic-syria	breton
arabic-SY	british
armenian	bulgarian
assamese	burmese
asturian	canadian
asu	cantonese
australian	catalan
austrian	centralatlastamazight
azerbaijani-cyrillic	centralkurdish
azerbaijani-cyrl	chechen
azerbaijani-latin	cherokee
azerbaijani-latn	chiga
azerbaijani	chinese-hans-hk
bafia	chinese-hans-mo
bambara	chinese-hans-sg
basaa	chinese-hans
basque	chinese-hant-hk
belarusian	chinese-hant-mo

chinese-hant	german
chinese-simplified-hongkongsarchina	greek
chinese-simplified-macausarchina	gujarati
chinese-simplified-singapore	gusii
chinese-simplified	hausa-gh
chinese-traditional-hongkongsarchina	hausa-ghana
chinese-traditional-macausarchina	hausa-ne
chinese-traditional	hausa-niger
chinese	hausa
cognian	hawaiian
cornish	hebrew
croatian	hindi
czech	hungarian
danish	icelandic
duala	igbo
dutch	inarisami
dzongkha	indonesian
embu	interlingua
english-au	irish
english-australia	italian
english-ca	japanese
english-canada	jolafoyi
english-gb	kabuverdianu
english-newzealand	kabyle
english-nz	kako
english-unitedkingdom	kalaallisut
english-unitedstates	kalenjin
english-us	kamba
english	kannada
esperanto	kashmiri
estonian	kazakh
ewe	khmer
ewondo	kikuyu
faroeese	kinyarwanda
filipino	konkani
finnish	korean
french-be	koyraborosenni
french-belgium	koyrachiini
french-ca	kwasio
french-canada	kyrgyz
french-ch	lakota
french-lu	langi
french-luxembourg	lao
french-switzerland	latvian
french	lingala
friulian	lithuanian
fulah	lowersorbian
galician	lsorbian
ganda	lubakatanga
georgian	luo
german-at	luxembourgish
german-austria	luyia
german-ch	macedonian
german-switzerland	machame

makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua

romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx

spanish	usorbian
standardmoroccantamazight	uyghur
swahili	uzbek-arab
swedish	uzbek-arabic
swissgerman	uzbek-cyrillic
tachelhit-latin	uzbek-cyrl
tachelhit-latn	uzbek-latin
tachelhit-tfng	uzbek-latn
tachelhit-tifinagh	uzbek
tachelhit	vai-latin
taita	vai-latn
tamil	vai-vai
tasawaq	vai-vaii
telugu	vai
teso	vietnam
thai	vietnamese
tibetan	vunjo
tigrinya	walser
tongan	welsh
turkish	westernfrisian
turkmen	yangben
ukenglish	yiddish
ukrainian	yoruba
upporsorbian	zarma
urdu	zulu afrikaans
usenglish	

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}
```

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


```

\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}

```

`\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 ones.

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).

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.

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.

WARNING Do not use `\setxxxxfont` and `\babelfont` at the same time. `\babelfont` follows the standard \TeX 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).

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

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

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

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

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.

`\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= *<language-tag>*

Loads only the strings. For example:

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

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

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

New 3.15 Sets the script name to be used by fontspec (eg, Devanagari). Overrides the value in the ini file. This value is particularly important because it sets the writing direction, so you must use it if for some reason the default value is wrong.

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

New 3.15 Sets the language name to be used by fontspec (eg, Hindi). Overrides the value in the ini file. 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. 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.¹⁵ So, there should be at most 3 directives of this kind.

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

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

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

¹⁵In future releases an new value (script) will be added.

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

NOTE (1) If you need shorthands, you can use `\usesorthands` and `\definesorthand` 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). 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*.

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` `{\<language>}{\<true>}{\<false>}`

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

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

1.19 Hyphenation tools

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

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

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

In \TeX , `-` and `\-` forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, `-` in Dutch, 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 opportunity without a hyphen at all.
- `\babelhyphen{<text>}` is a hard “hyphen” using `<text>` instead. A typical case is `\babelhyphen{/}`.

With all of them hyphenation in the rest of the word is enabled. If you don’t want 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 \LaTeX it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in \LaTeX , but it can be changed to another value by redefining `\babeInullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones.

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

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

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

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

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

New 3.9m In *luatex* only,¹⁶ 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.

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.¹⁷

Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the `babel` core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main latin encoding was `LY1`), and therefore it has been deprecated.¹⁸

`\ensureascii` { `<text>` }

New 3.9i This macro makes sure `<text>` 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 used for “ordinary” text, so they should be selected with some other encoding, but they are taken into account, just in case.

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.

¹⁶With `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.

¹⁷The so-called Unicode fonts do not improve the situation either. So, a font suited for Vietnamese is not necessarily suited for, say, the romanization of Indic languages, and the fact it contains glyphs for Modern Greek does not mean it includes them for Classic Greek.

¹⁸But still defined for backwards compatibility.

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 very likely until (Northern) Winter. This applies to text, but **graphical** elements, including the picture environment and PDF or PS based graphics, are not yet correctly handled. Also, indexes and the like are under study, as well as math.

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

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 in typical cases.

New 3.19 Finally, basic supports both L and R text. (They are named basic mainly because they only consider the intrinsic direction of scripts and weak directionality.) 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.¹⁹

```
\documentclass{article}

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

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

¹⁹At the time of this writing some Arabic fonts are not rendered correctly by the default luatex font loader, with misplaced kerns inside some words, so double check the resulting text. Have a look at the workaround available on GitHub, under /required/babel/samples

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{فصحى العصر} \textit{fuṣḥā l-‘aṣr} (MSA) and
\textit{فصحى التراث} \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 | extras

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements. You may use several options with a comma-separated list (eg, `layout=counters.contents.sectioning`). This list will be expanded in future releases (tables, captions, etc.). Note not all options are required by all engines.

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

counters required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection{.section}`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for

numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it could depend on the counter format.

With counters, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while 1.2 in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is *c2.c1*. Of course, you may always adjust the order by changing the language, if necessary.²⁰

lists required in xetex and pdfTeX, but only in multilingual documents in luatex.

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 multilingual documents in all engines; you may use alternatively `\BabelFootnote` described below (what this options does exactly is also explained there).

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

tabular required in luatex for R tabular (it has been tested only with simple tables, so expect some readjustments in the future); ignored in pdfTeX or xetex (which will not support a similar option in the short term) **New 3.18** ,

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

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

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

There are *three* R blocks and *two* L blocks, and the order is *RTL B and still ltr 1 ltr text RTL A*. This is by design to provide the proper behaviour 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.

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

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 `{⟨cmd⟩}{⟨local-language⟩}{⟨before⟩}{⟨after⟩}`

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 $\{\langle name \rangle\}\{\langle event \rangle\}\{\langle code \rangle\}$

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the 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	upporsorbian
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). 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 .

NOTE Please, for info about the support in `luatex` for some complex scripts, see the wiki, on <https://github.com/latex3/latex2e/wiki/Babel:-Remarks-on-the-luatex-support-for-some-scripts>.

1.25 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 \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.²¹ So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of \TeX , not of `babel`. Alternatively, you may use `\usesshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.

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

- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the ‘to do’ list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make \TeX enter in an infinite loop in some rare cases. (Another issue in the ‘to do’ list, although there is a partial solution.)

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex .

1.26 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).

It is possible now to typeset Arabic or Hebrew with numbers and L text. Next on the roadmap are line breaking in Thai and the like, as well as “non-European” digits. Also on the roadmap are R layouts (lists, footnotes, tables, column order), page and section numbering, and maybe kashida justification.

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

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.

1.27 Tentative and experimental code

Southeast Asian interword spacing

There is some preliminary interword spacing for Thai, Lao and Khemer in luatex (provided there are hyphenation patterns). It is activated automatically if a language with one of these scripts are loaded with `\babelprovide`. See the sample on the babel repository.

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:

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

- `\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}
```

Bidi writing in `luatex` is under development, but a basic implementation is almost finished. On the other hand, in `xetex` it is taking its first steps. The latter engine poses quite different challenges. An option to manage document layout in `luatex` (lists, footnotes, etc.) is almost finished, but `xetex` required more work. See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). `xetex` relies on the font to properly handle these unmarked changes, so it is not under the control of \TeX .

2 Loading languages with `language.dat`

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

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

2.1 Format

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

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

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.


```
% 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.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are

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

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

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

Some recommendations:

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

- 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.

²⁷But not removed, for backward compatibility.

- 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.

`\main@language` To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of

	<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\monthname{<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>}

```

3.4 Support for active characters

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

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

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

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

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

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

3.5 Support for saving macro definitions

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

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

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

²⁸This mechanism was introduced by Bernd Raichle.

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

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

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

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

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

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

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

\StartBabelCommands `{⟨language-list⟩}{⟨category⟩}[⟨selector⟩]`

The `⟨language-list⟩` specifies which languages the block is intended for. A block is taken into account only if the `\CurrentOption` is listed here. Alternatively, you can define `\BabelLanguages` to a comma-separated list of languages to be defined (if undefined, `\StartBabelCommands` sets it to `\CurrentOption`). You may write `\CurrentOption` as the language, but this is discouraged – a explicit name (or names) is much better and clearer.

A “selector” is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name `unicode` must be used for `xetex` and `luatex` (the key strings has also other two special values: `generic` and `encoded`). If a string is set several times (because several blocks are read), the first one take 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 honoured (in a encoded way).

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
```

²⁹In future releases further categories may be added.

```

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

\StartBabelCommands `* \langle language-list \rangle \langle category \rangle [\langle selector \rangle]`

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

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands `\langle code \rangle`

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

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

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

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

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

A convenient way to define several ordered names at once. For example, to define `\abmoniname`, `\abmoniiiname`, 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}

```

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

#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 \@uc\clist (eg, \bb\BB\cc\CC). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in L^AT_EX, we could set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

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

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

```
\SetHyphenMap{\BabelLowerMM{"100}{ "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 `other language*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with `^` (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that lead to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `.ldf` files not bundled with babel were not recognized when called as global options.

Part II

The 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 an uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). Multi-letter qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (all lowercase).

7 Tools

```
1 <<version=3.25>>
2 <<date=2018/10/03>>
```

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` 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.

`\bbl@afterfi`

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

`\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
```

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

```

34 \else
35 \expandafter\bb1@trim@b\expandafter#1%
36 \fi}%
37 \long\def\bb1@trim@b#1##1 \nil{\bb1@trim@i##1}}
38 \bb1@tempa{ }
39 \long\def\bb1@trim@i#1\@nil#2\relax#3{#3{#1}}
40 \long\def\bb1@trim@def#1{\bb1@trim{\def#1}}

```

`\bb1@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\bb1@ifunset#1{%
42 \expandafter\ifx\csname#1\endcsname\relax
43 \expandafter\@firstoftwo
44 \else
45 \expandafter\@secondoftwo
46 \fi}
47 \bb1@ifunset{ifcsname}%
48 {}%
49 {\def\bb1@ifunset#1{%
50 \ifcsname#1\endcsname
51 \expandafter\ifx\csname#1\endcsname\relax
52 \bb1@afterelse\expandafter\@firstoftwo
53 \else
54 \bb1@afterfi\expandafter\@secondoftwo
55 \fi
56 \else
57 \expandafter\@firstoftwo
58 \fi}}

```

`\bb1@ifblank` A tool from url, by Donald Arseneau, which tests if a string is empty or space.

```

59 \def\bb1@ifblank#1{%
60 \bb1@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
61 \long\def\bb1@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\bb1@forkv#1#2{%
63 \def\bb1@kvcmd##1##2##3{#2}%
64 \bb1@kvnext#1,\@nil,}
65 \def\bb1@kvnext#1,{%
66 \ifx\@nil#1\relax\else
67 \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
68 \expandafter\bb1@kvnext
69 \fi}
70 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
71 \bb1@trim@def\bb1@forkv@a{#1}%
72 \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@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\bb1@vforeach#1#2{%
74 \def\bb1@forcmd##1{#2}%
75 \bb1@fornext#1,\@nil,}
76 \def\bb1@fornext#1,{%
77 \ifx\@nil#1\relax\else
78 \bb1@ifblank{#1}{\bb1@trim\bb1@forcmd{#1}}%

```

```

79 \expandafter\bb1@for next
80 \fi}
81 \def\bb1@foreach#1{\expandafter\bb1@vforeach\expandafter{#1}}

```

\bb1@replace

```

82 \def\bb1@replace#1#2#3{% in #1 -> repl #2 by #3
83 \toks@{}}%
84 \def\bb1@replace@aux##1#2##2#2{%
85 \ifx\bb1@nil##2%
86 \toks@\expandafter{\the\toks@##1}%
87 \else
88 \toks@\expandafter{\the\toks@##1#3}%
89 \bb1@afterfi
90 \bb1@replace@aux##2#2%
91 \fi}%
92 \expandafter\bb1@replace@aux#1#2\bb1@nil#2%
93 \edef#1{\the\toks@}}

```

\bb1@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.

```

94 \def\bb1@exp#1{%
95 \begingroup
96 \let\>\noexpand
97 \def\<##1>{\expandafter\>\noexpand\csname##1\endcsname}%
98 \edef\bb1@exp@aux{\endgroup#1}%
99 \bb1@exp@aux}

```

Two further tools. \bb1@samestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bb1@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.

```

100 \def\bb1@ifsamestring#1#2{%
101 \begingroup
102 \protected@edef\bb1@tempb{#1}%
103 \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
104 \protected@edef\bb1@tempc{#2}%
105 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
106 \ifx\bb1@tempb\bb1@tempc
107 \aftergroup\@firstoftwo
108 \else
109 \aftergroup\@secondoftwo
110 \fi
111 \endgroup}
112 \chardef\bb1@engine=
113 \ifx\directlua\undefined
114 \ifx\XeTeXinputencoding\undefined
115 \z@
116 \else
117 \tw@
118 \fi
119 \else
120 \@ne
121 \fi
122 <</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 .

```
123 <<*Make sure ProvidesFile is defined>> ≡
124 \ifx\ProvidesFile\@undefined
125   \def\ProvidesFile#1[#2 #3 #4]{%
126     \wlog{File: #1 #4 #3 <#2>}%
127     \let\ProvidesFile\@undefined}
128 \fi
129 <</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`.

```
130 <<*Load patterns in luatex>> ≡
131 \ifx\directlua\@undefined\else
132   \ifx\bb1@luapatterns\@undefined
133     \input luababel.def
134   \fi
135 \fi
136 <</Load patterns in luatex>>
```

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

```
137 <<*Load macros for plain if not LaTeX>> ≡
138 \ifx\AtBeginDocument\@undefined
139   \input plain.def\relax
140 \fi
141 <</Load macros for plain if not LaTeX>>
```

7.1 Multiple languages

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

```
142 <<*Define core switching macros>> ≡
143 \ifx\language\@undefined
144   \csname newcount\endcsname\language
145 \fi
146 <</Define core switching macros>>
```

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

`\addlanguage` To add languages to \TeX 's memory plain \TeX version 3.0 supplies `\newlanguage`, 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 `\newlanguage` was defined to be `\outer`. For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefore `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain \TeX version 3.0.

For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`. Plain \TeX version 3.0 uses `\count 19` for this purpose.

```
147 <<*Define core switching macros>> ≡
148 \ifx\newlanguage\@undefined
149   \csname newcount\endcsname\last@language
150   \def\addlanguage#1{%
```

```

151 \global\advance\last@language\@e
152 \ifnum\last@language<\@cclvi
153 \else
154 \errmessage{No room for a new \string\language!}%
155 \fi
156 \global\chardef#1\last@language
157 \wlog{\string#1 = \string\language\the\last@language}}
158 \else
159 \countdef\last@language=19
160 \def\addlanguage{\alloc@9\language\chardef\@cclvi}
161 \fi
162 <</Define core switching macros>>

```

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

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

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

In order to make use of the features of $\text{\LaTeX}2_{\epsilon}$, 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 set 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.

```

163 (*package)
164 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
165 \ProvidesPackage{babel}[\langle\langle date \rangle\rangle \langle\langle version \rangle\rangle The Babel package]
166 \ifpackagewith{babel}{debug}
167 {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
168 \let\bbl@debug\@firstofone}
169 {\providecommand\bbl@trace[1]{}%
170 \let\bbl@debug\@gobble}
171 \ifx\bbl@switchflag\undefined % Prevent double input
172 \let\bbl@switchflag\relax
173 \input switch.def\relax
174 \fi
175 <<Load patterns in luatex>>
176 <<Basic macros>>

```



```

177 \def\AfterBabelLanguage#1{%
178   \global\expandafter\bbbl@add\csname#1.ldf-h@k\endcsname}%

```

If the format created a list of loaded languages (in \bbbl@languages), get the name of the 0-th to show the actual language used.

```

179 \ifx\bbbl@languages\undefined\else
180   \begingroup
181     \catcode'\^^I=12
182     \@ifpackagewith{babel}{showlanguages}{%
183       \begingroup
184         \def\bbbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
185         \wlog{<*languages>}%
186         \bbbl@languages
187         \wlog{</languages>}%
188       \endgroup}{%
189     \endgroup
190     \def\bbbl@elt#1#2#3#4{%
191       \ifnum#2=\z@
192         \gdef\bbbl@nulllanguage{#1}%
193         \def\bbbl@elt##1##2##3##4{%
194           \fi}%
195       \bbbl@languages
196     \fi
197   \ifodd\bbbl@engine
198     \let\bbbl@tempa\relax
199     \@ifpackagewith{babel}{bidi=basic}%
200     {\def\bbbl@tempa{basic}}%
201     {\@ifpackagewith{babel}{bidi=basic-r}%
202      {\def\bbbl@tempa{basic-r}}}%
203     {}%
204   \ifx\bbbl@tempa\relax\else
205     \let\bbbl@beforeforeign\leavevmode
206     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
207     \RequirePackage{luatexbase}%
208     \directlua{
209       require('babel-bidi.lua')
210       require('babel-bidi-\bbbl@tempa.lua')
211       luatexbase.add_to_callback('pre_linebreak_filter',
212         Babel.pre_otfload_v,
213         'Babel.pre_otfload_v',
214       luatexbase.priority_in_callback('pre_linebreak_filter',
215         'luaotfload.node_processor') or nil)
216       luatexbase.add_to_callback('hpack_filter',
217         Babel.pre_otfload_h,
218         'Babel.pre_otfload_h',
219       luatexbase.priority_in_callback('hpack_filter',
220         'luaotfload.node_processor') or nil)
221     }
222   \fi
223 \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.

```

224 \bbbl@trace{Defining option 'base'}
225 \@ifpackagewith{babel}{base}{%
226   \ifx\directlua\undefined
227     \DeclareOption*{\bbbl@patterns{\CurrentOption}}%
228   \else
229     \DeclareOption*{\bbbl@patterns@lua{\CurrentOption}}%

```

```

230 \fi
231 \DeclareOption{base}{}%
232 \DeclareOption{showlanguages}{}%
233 \ProcessOptions
234 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
235 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
236 \global\let@ifl@ter@@\ifl@ter
237 \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\@ifl@ter@@}%
238 \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.

```

239 \bbl@trace{key=value and another general options}
240 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
241 \def\bbl@tempb#1.#2{%
242   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
243 \def\bbl@tempd#1.#2\@nnil{%
244   \ifx\@empty#2%
245     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
246   \else
247     \in@{=}{#1}\ifin@
248       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
249     \else
250       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
251     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
252   \fi
253 \fi}
254 \let\bbl@tempc\@empty
255 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
256 \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.

```

257 \DeclareOption{KeepShorthandsActive}{}
258 \DeclareOption{activeacute}{}
259 \DeclareOption{activegrave}{}
260 \DeclareOption{debug}{}
261 \DeclareOption{noconfigs}{}
262 \DeclareOption{showlanguages}{}
263 \DeclareOption{silent}{}
264 \DeclareOption{mono}{}
265 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
266 <<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.

```

267 \let\bbl@opt@shorthands\@nnil

```

```

268 \let\bbl@opt@config\@nnil
269 \let\bbl@opt@main\@nnil
270 \let\bbl@opt@headfoot\@nnil
271 \let\bbl@opt@layout\@nnil

```

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

```

272 \def\bbl@tempa#1=#2\bbl@tempa{%
273   \bbl@csarg\ifx{opt@#1}\@nnil
274   \bbl@csarg\edef{opt@#1}{#2}%
275   \else
276   \bbl@error{%
277     Bad option `#1=#2'. Either you have misspelled the\\%
278     key or there is a previous setting of `#1'}{%
279     Valid keys are `shorthands', `config', `strings', `main',\\%
280     `headfoot', `safe', `math', among others.}
281   \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.

```

282 \let\bbl@language@opts\@empty
283 \DeclareOption*{%
284   \bbl@xin@{\string=}{\CurrentOption}%
285   \ifin@
286     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
287   \else
288     \bbl@add@list\bbl@language@opts{\CurrentOption}%
289   \fi}

```

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

```

290 \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=...

```

291 \bbl@trace{Conditional loading of shorthands}
292 \def\bbl@sh@string#1{%
293   \ifx#1\@empty\else
294     \ifx#1t\string~%
295     \else\ifx#1c\string,%
296     \else\string#1%
297   \fi\fi
298   \expandafter\bbl@sh@string
299   \fi}
300 \ifx\bbl@opt@shorthands\@nnil
301   \def\bbl@ifshorthand#1#2#3{#2}%
302 \else\ifx\bbl@opt@shorthands\@empty
303   \def\bbl@ifshorthand#1#2#3{#3}%
304 \else

```

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

```

305 \def\bbl@ifshorthand#1{%
306   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%

```

```

307 \ifin@
308 \expandafter\@firstoftwo
309 \else
310 \expandafter\@secondoftwo
311 \fi}

```

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

```

312 \edef\bbl@opt@shorthands{%
313 \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

314 \bbl@ifshorthand{'}%
315 {\PassOptionsToPackage{activeacute}{babel}}{}
316 \bbl@ifshorthand{'}%
317 {\PassOptionsToPackage{activegrave}{babel}}{}
318 \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.

```

319 \ifx\bbl@opt@headfoot\@nnil\else
320 \g@addto@macro\@resetactivechars{%
321 \set@typeset@protect
322 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
323 \let\protect\noexpand}
324 \fi

```

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

```

325 \ifx\bbl@opt@safe\@undefined
326 \def\bbl@opt@safe{BR}
327 \fi
328 \ifx\bbl@opt@main\@nnil\else
329 \edef\bbl@language@opts{%
330 \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
331 \bbl@opt@main}
332 \fi

```

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

```

333 \bbl@trace{Defining IfBabelLayout}
334 \ifx\bbl@opt@layout\@nnil
335 \newcommand\IfBabelLayout[3]{#3}%
336 \else
337 \newcommand\IfBabelLayout[1]{%
338 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
339 \ifin@
340 \expandafter\@firstoftwo
341 \else
342 \expandafter\@secondoftwo
343 \fi}
344 \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).

```

345 \bbl@trace{Language options}
346 \let\bbl@afterlang\relax
347 \let\BabelModifiers\relax
348 \let\bbl@loaded\@empty
349 \def\bbl@load@language#1{%
350   \InputIfFileExists{#1.ldf}%
351   {\edef\bbl@loaded{\CurrentOption
352     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
353     \expandafter\let\expandafter\bbl@afterlang
354       \csname\CurrentOption.ldf-h@k\endcsname
355     \expandafter\let\expandafter\BabelModifiers
356       \csname bbl@mod@\CurrentOption\endcsname}%
357   {\bbl@error{%
358     Unknown option '\CurrentOption'. Either you misspelled it\\%
359     or the language definition file \CurrentOption.ldf was not found}}{%
360     Valid options are: shorthands=, KeepShorthandsActive,\\%
361     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
362     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

363 \def\bbl@try@load@lang#1#2#3{%
364   \IfFileExists{\CurrentOption.ldf}%
365   {\bbl@load@language{\CurrentOption}}%
366   {#1\bbl@load@language{#2}#3}}
367 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
368 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
369 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
370 \DeclareOption{hebrew}{%
371   \input{rlbabel.def}%
372   \bbl@load@language{hebrew}}
373 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
374 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
375 \DeclareOption{nyorsk}{\bbl@try@load@lang{}{norsk}{}}
376 \DeclareOption{polutonikogreek}{%
377   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
378 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
379 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
380 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
381 \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.

```

382 \ifx\bbl@opt@config\@nnil
383   \@ifpackagewith{babel}{noconfigs}{}%
384   {\InputIfFileExists{bblopts.cfg}%
385     {\typeout{*****^J%
386       * Local config file bblopts.cfg used^^J%
387       *}}%
388     {}}%
389 \else
390   \InputIfFileExists{\bbl@opt@config.cfg}%
391   {\typeout{*****^J%
392     * Local config file \bbl@opt@config.cfg used^^J%

```

```

393         *}}%
394     {\bbl@error{%
395         Local config file '\bbl@opt@config.cfg' not found}{%
396         Perhaps you misspelled it.}}%
397 \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.

```

398 \bbl@for\bbl@tempa\bbl@language@opts{%
399     \bbl@ifunset{ds@\bbl@tempa}%
400     {\edef\bbl@tempb{%
401         \noexpand\DeclareOption
402         {\bbl@tempa}%
403         {\noexpand\bbl@load@language{\bbl@tempa}}}%
404     \bbl@tempb}%
405     \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.

```

406 \bbl@foreach\@classoptionslist{%
407     \bbl@ifunset{ds@#1}%
408     {\IfFileExists{#1.ldf}%
409     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
410     {}}%
411     {}}

```

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

```

412 \ifx\bbl@opt@main\@nnil\else
413     \expandafter
414     \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
415     \DeclareOption{\bbl@opt@main}{}
416 \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):

```

417 \def\AfterBabelLanguage#1{%
418     \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
419     \DeclareOption*{}
420 \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`.

```

421 \ifx\bbl@opt@main\@nnil
422     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
423     \let\bbl@tempc\empty
424     \bbl@for\bbl@tempb\bbl@tempa{%
425         \bbl@xin{,\bbl@tempb,}{,\bbl@loaded,}%
426         \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
427     \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}

```

```

428 \expandafter\bbbl@tempa\bbbl@loaded,\@nnil
429 \ifx\bbbl@tempb\bbbl@tempc\else
430   \bbbl@warning{%
431     Last declared language option is '\bbbl@tempc',\%
432     but the last processed one was '\bbbl@tempb'.\%
433     The main language cannot be set as both a global\%
434     and a package option. Use 'main=\bbbl@tempc' as\%
435     option. Reported}%
436 \fi
437 \else
438 \DeclareOption{\bbbl@opt@main}{\bbbl@loadmain}
439 \ExecuteOptions{\bbbl@opt@main}
440 \DeclareOption*{}
441 \ProcessOptions*
442 \fi
443 \def\AfterBabelLanguage{%
444   \bbbl@error
445   {Too late for \string\AfterBabelLanguage}%
446   {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 `\bbbl@main@language`, has become defined. If not, no language has been loaded and an error message is displayed.

```

447 \ifx\bbbl@main@language\@undefined
448   \bbbl@info{%
449     You haven't specified a language. I'll use 'nil'\%
450     as the main language. Reported}
451   \bbbl@load@language{nil}
452 \fi
453 \</package>
454 \<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

```

455 \ifx\ldf@quit\@undefined
456 \else
457   \expandafter\endinput
458 \fi
459 \<<Make sure ProvidesFile is defined>>

```

```

460 \ProvidesFile{babel.def}[\langle date \rangle \langle version \rangle Babel common definitions]
461 \langle Load macros for plain if not LaTeX \rangle

```

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).

```

462 \ifx\bbl@ifshorthand\@undefined
463   \let\bbl@opt@shorthands\@nnil
464   \def\bbl@ifshorthand#1#2#3{#2}%
465   \let\bbl@language@opts\@empty
466   \ifx\babeloptionstrings\@undefined
467     \let\bbl@opt@strings\@nnil
468   \else
469     \let\bbl@opt@strings\babeloptionstrings
470   \fi
471   \def\BabelStringsDefault{generic}
472   \def\bbl@tempa{normal}
473   \ifx\babeloptionmath\bbl@tempa
474     \def\bbl@mathnormal{\noexpand\textormath}
475   \fi
476   \def\AfterBabelLanguage#1#2{}
477   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
478   \let\bbl@afterlang\relax
479   \def\bbl@opt@safe{BR}
480   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
481   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
482 \fi

```

And continue.

```

483 \ifx\bbl@switchflag\@undefined % Prevent double input
484   \let\bbl@switchflag\relax
485   \input switch.def\relax
486 \fi
487 \bbl@trace{Compatibility with language.def}
488 \ifx\bbl@languages\@undefined
489   \ifx\directlua\@undefined
490     \openin1 = language.def
491     \ifeof1
492       \closein1
493       \message{I couldn't find the file language.def}
494     \else
495       \closein1
496       \begingroup
497         \def\addlanguage#1#2#3#4#5{%
498           \expandafter\ifx\csname lang@#1\endcsname\relax\else
499             \global\expandafter\let\csname l@#1\endcsname\expandafter\endcsname
500             \csname lang@#1\endcsname
501           \fi}%
502         \def\uselanguage#1{}\fi
503         \input language.def
504       \endgroup
505     \fi
506   \fi
507   \chardef\l@english\z@
508 \fi
509 \langle Load patterns in luatex \rangle

```


510 $\langle\langle$ Basic macros $\rangle\rangle$

`\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 \langle control sequence \rangle and T_EX-code to be added to the \langle control sequence \rangle .

If the \langle control sequence \rangle 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 \langle control sequence \rangle is expanded and stored in a token register, together with the T_EX-code to be added. Finally the \langle control sequence \rangle is redefined, using the contents of the token register.

```
511 \def\addto#1#2{%
512   \ifx#1\@undefined
513     \def#1{#2}%
514   \else
515     \ifx#1\relax
516       \def#1{#2}%
517     \else
518       {\toks@\expandafter{#1#2}%
519        \xdef#1{the\toks@}}%
520     \fi
521   \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.

```
522 \def\bb1@withactive#1#2{%
523   \begingroup
524   \lccode`~=#2\relax
525   \lowercase{\endgroup#1~}}
```

`\bb1@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 L^AT_EX 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 `\bb1@redefine` which takes care of this. It creates a new control sequence, `\org@...`

```
526 \def\bb1@redefine#1{%
527   \edef\bb1@tempa{\bb1@stripslash#1}%
528   \expandafter\let\csname org@\bb1@tempa\endcsname#1%
529   \expandafter\def\csname\bb1@tempa\endcsname}
```

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

```
530 \@onlypreamble\bb1@redefine
```

`\bb1@redefine@long` This version of `\babel@redefine` can be used to redefine `\long` commands such as `\ifthenelse`.

```
531 \def\bb1@redefine@long#1{%
532   \edef\bb1@tempa{\bb1@stripslash#1}%
533   \expandafter\let\csname org@\bb1@tempa\endcsname#1%
534   \expandafter\long\expandafter\def\csname\bb1@tempa\endcsname}
535 \@onlypreamble\bb1@redefine@long
```

`\bb1@redefineroast` 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_`.

```

536 \def\bbl@redefineroobust#1{%
537   \edef\bbl@tempa{\bbl@stripslash#1}%
538   \bbl@ifunset{\bbl@tempa\space}%
539   {\expandafter\let\csname org\bbl@tempa\endcsname#1%
540    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
541   {\bbl@exp{\let\<org\bbl@tempa>\<\bbl@tempa\space>}}}%
542   \@namedef{\bbl@tempa\space}}

```

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

```

543 \@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.

```

544 \bbl@trace{Hooks}
545 \def\AddBabelHook#1#2{%
546   \bbl@ifunset{\bbl@hk@#1}{\EnableBabelHook{#1}}}%
547   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
548   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
549   \bbl@ifunset{\bbl@ev@#1@#2}%
550   {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}}%
551   \bbl@csarg\newcommand}%
552   {\bbl@csarg\let{ev@#1@#2}\relax
553   \bbl@csarg\newcommand}%
554   {ev@#1@#2}[\bbl@tempb]}
555 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
556 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
557 \def\bbl@usehooks#1#2{%
558   \def\bbl@elt##1{%
559     \@nameuse{\bbl@hk@##1}{\@nameuse{\bbl@ev@##1@#1}#2}}%
560   \@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).

```

561 \def\bbl@evargs{,% <- don't delete this comma
562   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
563   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
564   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
565   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.

```

566 \bbl@trace{Defining babelensure}
567 \newcommand\babelensure[2][{}]{% TODO - revise test files
568   \AddBabelHook{babel-ensure}{afterextras}{%
569     \ifcase\bbl@select@type
570       \@nameuse{\bbl@e@\languagename}%
571     \fi}%
572   \beginngroup
573     \let\bbl@ens@include\@empty
574     \let\bbl@ens@exclude\@empty
575     \def\bbl@ens@fontenc{\relax}%
576     \def\bbl@tempb##1{%
577       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
578     \edef\bbl@tempa{\bbl@tempb##1\@empty}%
579     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
580     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
581     \def\bbl@tempc{\bbl@ensure}%
582     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
583       \expandafter{\bbl@ens@include}}%
584     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
585       \expandafter{\bbl@ens@exclude}}%
586     \toks@\expandafter{\bbl@tempc}%
587     \bbl@exp{%
588   \endgroup
589   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
590 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
591   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
592     \ifx##1\@empty\else
593       \in@{##1}{#2}%
594     \ifin\@empty\else
595       \bbl@ifunset{\bbl@ensure@\languagename}%
596       {\bbl@exp{%
597         \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
598           \\\foreignlanguage{\languagename}%
599           {\ifx\relax#3\else
600             \\\fontencoding{#3}\selectfont
601             \fi
602             #####1}}}%
603       }%
604       \toks@\expandafter{##1}%
605       \edef##1{%
606         \bbl@csarg\noexpand{ensure@\languagename}%
607         {\the\toks@}}%
608       \fi
609       \expandafter\bbl@tempb
610     \fi}%
611   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
612   \def\bbl@tempa##1{% elt for include list
613     \ifx##1\@empty\else
614       \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
615     \ifin\@empty\else
616       \bbl@tempb##1\@empty
617     \fi
618     \expandafter\bbl@tempa
619     \fi}%
620   \bbl@tempa#1\@empty}
621 \def\bbl@captionslist{%
622   \prefacename\refname\abstractname\bibname\chaptername\appendixname
623   \contentsname\listfigurename\listtablename\indexname\figurename
624   \tablename\partname\enclname\ccname\headtoname\pagename\seename

```

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 `@`-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`.

```

626 \bbl@trace{Macros for setting language files up}
627 \def\bbl@ldfinit{%
628   \let\bbl@screset\@empty
629   \let\BabelStrings\bbl@opt@string
630   \let\BabelOptions\@empty
631   \let\BabelLanguages\relax
632   \ifx\originalTeX\@undefined
633     \let\originalTeX\@empty
634   \else
635     \originalTeX
636   \fi}
637 \def\LdfInit#1#2{%
638   \chardef\atcatcode=\catcode`\@
639   \catcode`\@=11\relax
640   \chardef\eqcatcode=\catcode`\=
641   \catcode`\==12\relax
642   \expandafter\if\expandafter\@backslashchar
643     \expandafter\@car\string#2\@nil
644     \ifx#2\@undefined\else
645       \ldf@quit{#1}%
646     \fi
647   \else
648     \expandafter\ifx\csname#2\endcsname\relax\else
649       \ldf@quit{#1}%
650     \fi
651   \fi
652   \bbl@ldfinit}

```

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

```

653 \def\ldf@quit#1{%
654   \expandafter\main@language\expandafter{#1}%
655   \catcode`\@=\atcatcode \let\atcatcode\relax

```

```

656 \catcode`\==\eqcatcode \let\eqcatcode\relax
657 \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.

```

658 \def\bbl@afterldf#1{%
659   \bbl@afterlang
660   \let\bbl@afterlang\relax
661   \let\BabelModifiers\relax
662   \let\bbl@screset\relax}%
663 \def\ldf@finish#1{%
664   \loadlocalcfg{#1}%
665   \bbl@afterldf{#1}%
666   \expandafter\main@language\expandafter{#1}%
667   \catcode`\@=\atcatcode \let\atcatcode\relax
668   \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 .

```

669 \@onlypreamble\LdfInit
670 \@onlypreamble\ldf@quit
671 \@onlypreamble\ldf@finish

```

`\main@language` This command should be used in the various language definition files. It stores its
`\bbl@main@language` argument in `\bbl@main@language`; to be used to switch to the correct language at the beginning of the document.

```

672 \def\main@language#1{%
673   \def\bbl@main@language{#1}%
674   \let\language\bbl@main@language
675   \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`.

```

676 \AtBeginDocument{%
677   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
678   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

679 \def\select@language@x#1{%
680   \ifcase\bbl@select@type
681     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
682   \else
683     \select@language{#1}%
684   \fi}

```

9.4 Shorthands

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

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

```

685 \bbl@trace{Shorhands}
686 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
687   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
688   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
689   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
690     \begingroup
691       \catcode`#1\active
692       \nfss@catcodes
693       \ifnum\catcode`#1=\active
694         \endgroup
695         \bbl@add\nfss@catcodes{\@makeother#1}%
696       \else
697         \endgroup
698       \fi
699   \fi}

```

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

```

700 \def\bbl@remove@special#1{%
701   \begingroup
702   \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
703     \else\noexpand##1\noexpand##2\fi}%
704   \def\do{\x\do}%
705   \def\@makeother{\x\@makeother}%
706   \edef\x{\endgroup
707     \def\noexpand\dospecials{\dospecials}%
708     \expandafter\ifx\csgname @sanitize\endcsname\relax\else
709       \def\noexpand\@sanitize{\@sanitize}%
710     \fi}%
711   \x}

```

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char⟨char⟩` to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char⟨char⟩` by default (`⟨char⟩` being the character to be made active). Later its definition can be changed to expand to `\active@char⟨char⟩` by calling `\bbl@activate{⟨char⟩}`.

For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines " as `\active@prefix "\active@char"` (where the first " is the character with its original catcode, when the shorthand is created, and `\active@char"` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original "); otherwise `\active@char"` is executed. This macro in turn expands to `\normal@char"` in “safe” contexts (eg, `\label`), but `\user@active"` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char"` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char"`.

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, `\<level>@group`, `<level>@active` and `<next-level>@active` (except in system).

```

712 \def\bbl@active@def#1#2#3#4{%
713   \@namedef{#3#1}{%
714     \expandafter\ifx\csgname#2@sh@#1\endcsname\relax

```

```

715 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
716 \else
717 \bbl@afterfi\csname#2@sh@#1@endcsname
718 \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.

```

719 \long\@namedef{#3@arg#1}##1{%
720 \expandafter\ifx\csname#2@sh@#1@string##1@endcsname\relax
721 \bbl@afterelse\csname#4#1@endcsname##1%
722 \else
723 \bbl@afterfi\csname#2@sh@#1@string##1@endcsname
724 \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.

```

725 \def\initiate@active@char#1{%
726 \bbl@ifunset{active@char\string#1}%
727 {\bbl@withactive
728 {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
729 {}}

```

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`).

```

730 \def\@initiate@active@char#1#2#3{%
731 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
732 \ifx#1\@undefined
733 \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
734 \else
735 \bbl@csarg\let{oridef@#2}#1%
736 \bbl@csarg\edef{oridef@#2}{%
737 \let\noexpand#1%
738 \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
739 \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*").

```

740 \ifx#1#3\relax
741 \expandafter\let\csname normal@char#2@endcsname#3%
742 \else
743 \bbl@info{Making #2 an active character}%
744 \ifnum\mathcode`#2="8000
745 \@namedef{normal@char#2}{%
746 \textormath{#3}{\csname bbl@oridef@@#2@endcsname}}%
747 \else
748 \@namedef{normal@char#2}{#3}%
749 \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).

```

750 \bbl@restoreactive{#2}%
751 \AtBeginDocument{%
752   \catcode`#2\active
753   \if@filesw
754     \immediate\write\@mainaux{\catcode`\string#2\active}%
755   \fi}%
756 \expandafter\bbl@add@special\csname#2\endcsname
757 \catcode`#2\active
758 \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⟩`).

```

759 \let\bbl@tempa\@firstoftwo
760 \if\string^#2%
761   \def\bbl@tempa{\noexpand\textormath}%
762 \else
763   \ifx\bbl@mathnormal\@undefined\else
764     \let\bbl@tempa\bbl@mathnormal
765   \fi
766 \fi
767 \expandafter\edef\csname active@char#2\endcsname{%
768   \bbl@tempa
769     {\noexpand\if@safe@actives
770       \noexpand\expandafter
771       \expandafter\noexpand\csname normal@char#2\endcsname
772     \noexpand\else
773       \noexpand\expandafter
774       \expandafter\noexpand\csname bbl@doactive#2\endcsname
775     \noexpand\fi}%
776   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
777 \bbl@csarg\edef{doactive#2}{%
778   \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!).

```

779 \bbl@csarg\edef{active@#2}{%
780   \noexpand\active@prefix\noexpand#1%
781   \expandafter\noexpand\csname active@char#2\endcsname}%
782 \bbl@csarg\edef{normal@#2}{%
783   \noexpand\active@prefix\noexpand#1%
784   \expandafter\noexpand\csname normal@char#2\endcsname}%
785 \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.

```

786 \bbl@active@def#2\user@group{user@active}{language@active}%
787 \bbl@active@def#2\language@group{language@active}{system@active}%
788 \bbl@active@def#2\system@group{system@active}{normal@char}%

```


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

```
789 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
790   {\expandafter\noexpand\csname normal@char#2\endcsname}%
791 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
792   {\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.

```
793 \if\string'#2%
794   \let\prim@s\bbl@prim@s
795   \let\active@math@prime#1%
796 \fi
797 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
798 <<{*More package options}>> ≡
799 \DeclareOption{math=active}{}
800 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
801 <</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.

```
802 \ifpackagewith{babel}{KeepShorthandsActive}%
803   {\let\bbl@restoreactive\@gobble}%
804   {\def\bbl@restoreactive#1{%
805     \bbl@exp{%
806       \\\AfterBabelLanguage\\CurrentOption
807       {\catcode`#1=\the\catcode`#1\relax}%
808       \\\AtEndOfPackage
809       {\catcode`#1=\the\catcode`#1\relax}}}%
810   \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.

```
811 \def\bbl@sh@select#1#2{%
812   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
813     \bbl@afterelse\bbl@scndcs
814   \else
815     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
816   \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.

```

817 \def\active@prefix#1{%
818   \ifx\protect\@typeset@protect
819   \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).

```

820   \ifx\protect\@unexpandable@protect
821     \noexpand#1%
822   \else
823     \protect#1%
824   \fi
825   \expandafter\@gobble
826 \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⟩`.

```

827 \newif\if@safe@actives
828 \@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.

```

829 \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`.

`\bbl@deactivate`

```

830 \def\bbl@activate#1{%
831   \bbl@withactive{\expandafter\let\expandafter}#1%
832   \csname bbl@active@\string#1\endcsname}
833 \def\bbl@deactivate#1{%
834   \bbl@withactive{\expandafter\let\expandafter}#1%
835   \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.

`\bbl@scndcs`

```

836 \def\bbl@firstcs#1#2{\csname#1\endcsname}
837 \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.

```

838 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
839 \def\@decl@short#1#2#3\@nil#4{%
840   \def\bbl@tempa{#3}%
841   \ifx\bbl@tempa\@empty
842     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
843     \bbl@ifunset{#1@sh@\string#2@}{}%
844     {\def\bbl@tempa{#4}%

```

```

845     \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbbl@tempa
846     \else
847         \bbbl@info
848         {Redefining #1 shorthand \string#2\\%
849         in language \CurrentOption}%
850     \fi}%
851     \@namedef{#1@sh@\string#2@}{#4}%
852 \else
853     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbbl@firstcs
854     \bbbl@ifunset{#1@sh@\string#2@\string#3@}{}%
855     {\def\bbbl@tempa{#4}%
856     \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbbl@tempa
857     \else
858         \bbbl@info
859         {Redefining #1 shorthand \string#2\string#3\\%
860         in language \CurrentOption}%
861     \fi}%
862     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
863 \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.

```

864 \def\textormath{%
865     \ifmmode
866         \expandafter\@secondoftwo
867     \else
868         \expandafter\@firstoftwo
869     \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’.

```

870 \def\user@group{user}
871 \def\language@group{english}
872 \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.

```

873 \def\useshorthands{%
874     \@ifstar\bbbl@usesh@s{\bbbl@usesh@x{}}
875 \def\bbbl@usesh@s#1{%
876     \bbbl@usesh@x
877     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbbl@activate{#1}}}%
878     {#1}}
879 \def\bbbl@usesh@x#1#2{%
880     \bbbl@ifshorthand{#2}%
881     {\def\user@group{user}%
882     \initiate@active@char{#2}%
883     #1%
884     \bbbl@activate{#2}}%
885     {\bbbl@error
886     {Cannot declare a shorthand turned off (\string#2)}
887     {Sorry, but you cannot use shorthands which have been\\%
888     turned off in the package options}}}

```

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

```

889 \def\user@language@group{user@\language@group}
890 \def\bbl@set@user@generic#1#2{%
891   \bbl@ifunset{user@generic@active#1}%
892   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
893     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
894     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
895       \expandafter\noexpand\csname normal@char#1\endcsname}%
896     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
897       \expandafter\noexpand\csname user@active#1\endcsname}%
898   \@empty}
899 \newcommand\defineshorthand[3][user]{%
900   \edef\bbl@tempa{\zap@space#1 \@empty}%
901   \bbl@for\bbl@tempb\bbl@tempa{%
902     \if*\expandafter\@car\bbl@tempb\@nil
903       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
904       \@expandtwoargs
905       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
906     \fi
907     \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```

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

```

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

```

909 \def\aliasshorthand#1#2{%
910   \bbl@ifshorthand{#2}%
911   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
912     \ifx\document\@notprerr
913       \@notshorthand{#2}%
914     \else
915       \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 latest to `\active@char`.

```

916     \expandafter\let\csname active@char\string#2\endcsname
917     \csname active@char\string#1\endcsname
918     \expandafter\let\csname normal@char\string#2\endcsname
919     \csname normal@char\string#1\endcsname
920     \bbl@activate{#2}%
921   \fi
922 \fi}%
923 {\bbl@error
924   {Cannot declare a shorthand turned off (\string#2)}
925   {Sorry, but you cannot use shorthands which have been\%
926     turned off in the package options}}}

```

`\@notshorthand`

```

927 \def\@notshorthand#1{%
928   \bbl@error{%

```

```

929   The character '\string #1' should be made a shorthand character;\%
930   add the command \string\usesshorthands\string{#1\string} to
931   the preamble.\%
932   I will ignore your instruction}%
933   {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.

```

934 \newcommand*\shorthandon[1]{\bbl@switch@sh@ne#1\@nnil}
935 \DeclareRobustCommand*\shorthandoff{%
936   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
937 \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.

```

938 \def\bbl@switch@sh#1#2{%
939   \ifx#2\@nnil\else
940     \bbl@ifunset{\bbl@active@\string#2}%
941     {\bbl@error
942       {I cannot switch '\string#2' on or off--not a shorthand}%
943       {This character is not a shorthand. Maybe you made\\%
944         a typing mistake? I will ignore your instruction}}}%
945     {\ifcase#1%
946       \catcode`#212\relax
947       \or
948       \catcode`#2\active
949       \or
950       \csname bbl@oricat@\string#2\endcsname
951       \csname bbl@oridef@\string#2\endcsname
952       \fi}%
953     \bbl@afterfi\bbl@switch@sh#1%
954   \fi}

```

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

```

955 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
956 \def\bbl@putsh#1{%
957   \bbl@ifunset{\bbl@active@\string#1}%
958   {\bbl@putsh@i#1\@empty\@nnil}%
959   {\csname bbl@active@\string#1\endcsname}}
960 \def\bbl@putsh@i#1#2\@nnil{%
961   \csname\languagename @sh@\string#1@%
962     \ifx\@empty#2\else\string#2@\fi\endcsname}
963 \ifx\bbl@opt@shorthands\@nnil\else
964   \let\bbl@s@initiate@active@char\initiate@active@char
965   \def\initiate@active@char#1{%
966     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
967   \let\bbl@s@switch@sh\bbl@switch@sh
968   \def\bbl@switch@sh#1#2{%
969     \ifx#2\@nnil\else

```

```

970 \bbl@afterfi
971 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
972 \fi}
973 \let\bbl@s@activate\bbl@activate
974 \def\bbl@activate#1{%
975 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
976 \let\bbl@s@deactivate\bbl@deactivate
977 \def\bbl@deactivate#1{%
978 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
979 \fi

```

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

```

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

```

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

`\bbl@pr@m@s`

```

981 \def\bbl@prim@s{%
982 \prime\futurelet\@let@token\bbl@pr@m@s}
983 \def\bbl@if@primes#1#2{%
984 \ifx#1\@let@token
985 \expandafter\@firstoftwo
986 \else\ifx#2\@let@token
987 \bbl@afterelse\expandafter\@firstoftwo
988 \else
989 \bbl@afterfi\expandafter\@secondoftwo
990 \fi\fi}
991 \begingroup
992 \catcode`\^=7 \catcode`\*= \active \lccode`\^=\^
993 \catcode`\'=12 \catcode`\="= \active \lccode`\"="\ '
994 \lowercase{%
995 \gdef\bbl@pr@m@s{%
996 \bbl@if@primes""%
997 \pr@@@s
998 {\bbl@if@primes*\^ \pr@@@t\egroup}}}
999 \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).

```

1000 \initiate@active@char{~}
1001 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1002 \bbl@activate{~}

```

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

`\T1dqpos`

```

1003 \expandafter\def\csname OT1dqpos\endcsname{127}
1004 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

1005 \ifx\f@encoding\@undefined
1006   \def\f@encoding{OT1}
1007 \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.

```

1008 \bbl@trace{Language attributes}
1009 \newcommand\languageattribute[2]{%
1010   \def\bbl@tempc{#1}%
1011   \bbl@fixname\bbl@tempc
1012   \bbl@iflanguage\bbl@tempc{%
1013     \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.

```

1014     \ifx\bbl@known@attribs\@undefined
1015       \in@false
1016     \else

```

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

```

1017       \bbl@xin@{,\bbl@tempc-##1,},{,\bbl@known@attribs,%}
1018     \fi

```

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

```

1019     \ifin@
1020       \bbl@warning{%
1021         You have more than once selected the attribute '##1'\%
1022         for language #1. Reported}%
1023     \else

```

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

```

1024       \bbl@exp{%
1025         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1026       \edef\bbl@tempa{\bbl@tempc-##1}%
1027       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1028       {\csname\bbl@tempc @attr##1\endcsname}%
1029       {\@attrerr{\bbl@tempc}{##1}}%
1030     \fi}}

```

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

```

1031 \@onlypreamble\languageattribute

```

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

```

1032 \newcommand*{\@attrerr}[2]{%
1033   \bbl@error
1034   {The attribute #2 is unknown for language #1.}%
1035   {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}`.

```

1036 \def\bbl@declare@ttribute#1#2#3{%
1037   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1038   \ifin@
1039     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1040   \fi
1041   \bbl@add@list\bbl@attributes{#1-#2}%
1042   \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret T_EX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```

1043 \def\bbl@ifattributeset#1#2#3#4{%

```

First we need to find out if any attributes were set; if not we're done.

```

1044   \ifx\bbl@known@attribs\undefined
1045     \in@false
1046     \else

```

The we need to check the list of known attributes.

```

1047     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1048   \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`'.

```

1049   \ifin@
1050     \bbl@afterelse#3%
1051   \else
1052     \bbl@afterfi#4%
1053   \fi
1054 }

```

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

```

1055 \def\bbl@ifknown@trib#1#2{%

```

We first assume the attribute is unknown.

```

1056   \let\bbl@tempa\@secondoftwo

```

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

```

1057   \bbl@loopx\bbl@tempb{#2}{%
1058     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1059   \ifin@

```

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

```

1060     \let\bbl@tempa\@firstoftwo
1061   \else
1062   \fi}%

```


Finally we execute `\bbl@tempa`.

```
1063 \bbl@tempa
1064 }
```

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

```
1065 \def\bbl@clear@ttribs{%
1066   \ifx\bbl@attributes\undefined\else
1067     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1068       \expandafter\bbl@clear@ttrib\bbl@tempa.
1069     }%
1070     \let\bbl@attributes\undefined
1071   \fi}
1072 \def\bbl@clear@ttrib#1-#2.{%
1073   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1074 \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`

```
1075 \bbl@trace{Macros for saving definitions}
1076 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1077 \newcount\babel@savecnt
1078 \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.

```
1079 \def\babel@save#1{%
1080   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1081   \toks@\expandafter{\originalTeX\let#1=}
1082   \bbl@exp{%
1083     \def\\originalTeX{the\toks@<babel@number\babel@savecnt>\relax}}
1084   \advance\babel@savecnt@ne}
```

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

```
1085 \def\babel@savevariable#1{%
1086   \toks@\expandafter{\originalTeX #1=}
1087   \bbl@exp{\def\\originalTeX{the\toks@the#1\relax}}}
```

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

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

```

1088 \def\bbl@frenchspacing{%
1089   \ifnum\the\scode\.\.=\@m
1090     \let\bbl@nonfrenchspacing\relax
1091   \else
1092     \frenchspacing
1093     \let\bbl@nonfrenchspacing\nonfrenchspacing
1094   \fi}
1095 \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.

```

1096 \bbl@trace{Short tags}
1097 \def\babeltags#1{%
1098   \edef\bbl@tempa{\zap@space#1 \@empty}%
1099   \def\bbl@tempb##1=##2\@{}%
1100   \edef\bbl@tempc{%
1101     \noexpand\newcommand
1102     \expandafter\noexpand\csname ##1\endcsname{%
1103       \noexpand\protect
1104       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1105     \noexpand\newcommand
1106     \expandafter\noexpand\csname text##1\endcsname{%
1107       \noexpand\foreignlanguage{##2}}
1108   \bbl@tempc}%
1109   \bbl@for\bbl@tempa\bbl@tempa{%
1110     \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.

```

1111 \bbl@trace{Hyphens}
1112 \@onlypreamble\babelhyphenation
1113 \AtEndOfPackage{%
1114   \newcommand\babelhyphenation[2][\@empty]{%
1115     \ifx\bbl@hyphenation@\relax
1116       \let\bbl@hyphenation@\@empty
1117     \fi
1118     \ifx\bbl@hyphlist\@empty\else
1119       \bbl@warning{%
1120         You must not intermingle \string\selectlanguage\space and\%
1121         \string\babelhyphenation\space or some exceptions will not\%
1122         be taken into account. Reported}%
1123     \fi
1124     \ifx\@empty#1%
1125       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1126     \else
1127       \bbl@vforeach{#1}{%

```

```

1128 \def\bb@tempa{##1}%
1129 \bb@fixname\bb@tempa
1130 \bb@iflanguage\bb@tempa{%
1131 \bb@csarg\protected@edef{hyphenation@\bb@tempa}{%
1132 \bb@ifunset{bb@hyphenation@\bb@tempa}%
1133 \@empty
1134 {\csname bb@hyphenation@\bb@tempa\endcsname\space}%
1135 #2}}}%
1136 \fi}}

```

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

```

1137 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1138 \def\bb@t@one{T1}
1139 \def\allowhyphens{\ifx\cf@encoding\bb@t@one\else\bb@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`.

```

1140 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1141 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1142 \def\bb@hyphen{%
1143 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
1144 \def\bb@hyphen@i#1#2{%
1145 \bb@ifunset{bb@hy@#1#2\@empty}%
1146 {\csname bb@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1147 {\csname bb@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 discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”.

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

```

1148 \def\bb@usehyphen#1{%
1149 \leavevmode
1150 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1151 \nobreak\hskip\z@skip}
1152 \def\bb@@usehyphen#1{%
1153 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1154 \def\bb@hyphenchar{%
1155 \ifnum\hyphenchar\font=\m@ne
1156 \babelnullhyphen
1157 \else
1158 \char\hyphenchar\font
1159 \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 `\bb@hy@nobreak` is redundant.

```

1160 \def\bb@hy@soft{\bb@usehyphen{\discretionary{\bb@hyphenchar}{}}{}}
1161 \def\bb@hy@@soft{\bb@usehyphen{\discretionary{\bb@hyphenchar}{}}{}}
1162 \def\bb@hy@hard{\bb@usehyphen\bb@hyphenchar}
1163 \def\bb@hy@@hard{\bb@usehyphen\bb@hyphenchar}

```

³³ $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1164 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1165 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1166 \def\bbl@hy@repeat{%
1167   \bbl@usehyphen{%
1168     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1169 \def\bbl@hy@@repeat{%
1170   \bbl@usehyphen{%
1171     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1172 \def\bbl@hy@empty{\hskip\z@skip}
1173 \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.

```

1174 \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.

```

1175 \bbl@trace{Multiencoding strings}
1176 \def\bbl@toglobal#1{\global\let#1#1}
1177 \def\bbl@recode#1{%
1178   \@tempcnta="7F
1179   \def\bbl@tempa{%
1180     \ifnum\@tempcnta>"FF\else
1181       \catcode\@tempcnta=#1\relax
1182       \advance\@tempcnta\@ne
1183       \expandafter\bbl@tempa
1184     \fi}%
1185   \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).

```

1186 \@ifpackagewith{babel}{nocase}%
1187 {\let\bbl@patchuclc\relax}%
1188 {\def\bbl@patchuclc{%
1189   \global\let\bbl@patchuclc\relax
1190   \@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1191   \gdef\bbl@uclc##1{%
1192     \let\bbl@encoded\bbl@encoded@uclc
1193     \bbl@ifunset{\language @bbl@uclc}% and resumes it
1194     {##1}%

```

```

1195      {\let\bbl@tempa##1\relax % Used by LANG@bbl@uc1c
1196      \csname\language @bbl@uc1c\endcsname}%
1197      {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1198      \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1199      \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}%
1200 <<(*More package options)>> ≡
1201 \DeclareOption{nocase}{}
1202 <</More package options>>

```

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

```

1203 <<(*More package options)>> ≡
1204 \let\bbl@opt@strings\@nnil % accept strings=value
1205 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1206 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1207 \def\BabelStringsDefault{generic}
1208 <</More package options>>

```

Main command This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

```

1209 \@onlypreamble\StartBabelCommands
1210 \def\StartBabelCommands{%
1211   \begingroup
1212   \bbl@recatcode{11}%
1213   <Macros local to BabelCommands>
1214   \def\bbl@provstring##1##2{%
1215     \providecommand##1{##2}%
1216     \bbl@tglobal##1}%
1217   \global\let\bbl@scafter\@empty
1218   \let\StartBabelCommands\bbl@startcmds
1219   \ifx\BabelLanguages\relax
1220     \let\BabelLanguages\CurrentOption
1221   \fi
1222   \begingroup
1223   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1224   \StartBabelCommands}
1225 \def\bbl@startcmds{%
1226   \ifx\bbl@screset\@nnil\else
1227     \bbl@usehooks{stopcommands}{}%
1228   \fi
1229   \endgroup
1230   \begingroup
1231   \@ifstar
1232     {\ifx\bbl@opt@strings\@nnil
1233       \let\bbl@opt@strings\BabelStringsDefault
1234       \fi
1235       \bbl@startcmds@i}%
1236     \bbl@startcmds@i}
1237 \def\bbl@startcmds@i#1#2{%
1238   \edef\bbl@L{\zap@space#1 \@empty}%
1239   \edef\bbl@G{\zap@space#2 \@empty}%
1240   \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.

```

1241 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1242   \let\SetString\@gobbletwo
1243   \let\bbbl@stringdef\@gobbletwo
1244   \let\AfterBabelCommands\@gobble
1245   \ifx\@empty#1%
1246     \def\bbbl@sc@label{generic}%
1247     \def\bbbl@encstring##1##2{%
1248       \ProvideTextCommandDefault##1{##2}%
1249       \bbbl@tglobal##1%
1250       \expandafter\bbbl@tglobal\csname\string?\string##1\endcsname}%
1251     \let\bbbl@sctest\in@true
1252   \else
1253     \let\bbbl@sc@charset\space % <- zapped below
1254     \let\bbbl@sc@fontenc\space % <- " "
1255     \def\bbbl@tempa##1=##2\@nil{%
1256       \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1257     \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
1258     \def\bbbl@tempa##1 ##2{% space -> comma
1259       ##1%
1260       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
1261     \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
1262     \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
1263     \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
1264     \def\bbbl@encstring##1##2{%
1265       \bbbl@foreach\bbbl@sc@fontenc{%
1266         \bbbl@ifunset{T#####1}%
1267         {}%
1268         {\ProvideTextCommand##1{#####1}{##2}%
1269         \bbbl@tglobal##1%
1270         \expandafter
1271         \bbbl@tglobal\csname#####1\string##1\endcsname}}}%
1272     \def\bbbl@sctest{%
1273       \bbbl@xin@{\bbbl@opt@strings,}{,\bbbl@sc@label,\bbbl@sc@fontenc,}}%
1274   \fi
1275   \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
1276   \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded
1277     \let\AfterBabelCommands\bbbl@aftercmds
1278     \let\SetString\bbbl@setstring
1279     \let\bbbl@stringdef\bbbl@encstring
1280   \else % ie, strings=value
1281     \bbbl@sctest
1282   \fin@
1283     \let\AfterBabelCommands\bbbl@aftercmds
1284     \let\SetString\bbbl@setstring
1285     \let\bbbl@stringdef\bbbl@provstring
1286   \fi\fi\fi
1287   \bbbl@scswitch
1288   \ifx\bbbl@G\@empty
1289     \def\SetString##1##2{%
1290       \bbbl@error{Missing group for string \string##1}%
1291       {You must assign strings to some category, typically\\%
1292       captions or extras, but you set none}}%
1293   \fi

```

```

1294 \ifx\@empty#1%
1295 \bbl@usehooks{defaultcommands}{}%
1296 \else
1297 \@expandtwoargs
1298 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1299 \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).

```

1300 \def\bbl@forlang#1#2{%
1301 \bbl@for#1\bbl@L{%
1302 \bbl@xin@{, #1, }{\, \BabelLanguages,}%
1303 \ifin@#2\relax\fi}}
1304 \def\bbl@scswitch{%
1305 \bbl@forlang\bbl@tempa{%
1306 \ifx\bbl@G\@empty\else
1307 \ifx\SetString\@gobbletwo\else
1308 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1309 \bbl@xin@{\, \bbl@GL, }{\, \bbl@screset,}%
1310 \ifin@\else
1311 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1312 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1313 \fi
1314 \fi
1315 \fi}}
1316 \AtEndOfPackage{%
1317 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\{#2\}}}%
1318 \let\bbl@scswitch\relax}
1319 \@onlypreamble\EndBabelCommands
1320 \def\EndBabelCommands{%
1321 \bbl@usehooks{stopcommands}{}%
1322 \endgroup
1323 \endgroup
1324 \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.

```

1325 \def\bbl@setstring#1#2{%
1326 \bbl@forlang\bbl@tempa{%
1327 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1328 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1329 {\global\expandafter % TODO - con \bbl@exp ?
1330 \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1331 {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
1332 {}%
1333 \def\BabelString{#2}%

```

```

1334 \bbl@usehooks{stringprocess}{}%
1335 \expandafter\bbl@stringdef
1336 \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include `\bbl@encoded` for string to be expanded in case transformations. It is `\relax` by default, but in `\MakeUppercase` and `\MakeLowercase` its value is a modified expandable `\@changed@cmd`.

```

1337 \ifx\bbl@opt@strings\relax
1338 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1339 \bbl@patchuclc
1340 \let\bbl@encoded\relax
1341 \def\bbl@encoded@uclc#1{%
1342   \@inmathwarn#1%
1343   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1344     \expandafter\ifx\csname ?\string#1\endcsname\relax
1345       \TextSymbolUnavailable#1%
1346     \else
1347       \csname ?\string#1\endcsname
1348     \fi
1349   \else
1350     \csname\cf@encoding\string#1\endcsname
1351   \fi}
1352 \else
1353 \def\bbl@scset#1#2{\def#1{#2}}
1354 \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.

```

1355 <<*Macros local to BabelCommands>> ≡
1356 \def\SetStringLoop##1##2{%
1357   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1358   \count@z@
1359   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1360     \advance\count@\@ne
1361     \toks@\expandafter{\bbl@tempa}%
1362     \bbl@exp{%
1363       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1364       \count@=\the\count@\relax}}}%
1365 <</Macros local to BabelCommands>>

```

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

```

1366 \def\bbl@aftercmds#1{%
1367   \toks@\expandafter{\bbl@scafter#1}%
1368   \xdef\bbl@scafter{\the\toks@}}

```

Case mapping The command `\SetCase` provides a way to change the behavior of `\MakeUppercase` and `\MakeLowercase`. `\bbl@tempa` is set by the patched `\@uclclist` to the parsing command.

```

1369 <<*Macros local to BabelCommands>> ≡
1370 \newcommand\SetCase[3][[]]{%
1371   \bbl@patchuclc
1372   \bbl@forlang\bbl@tempa{%
1373     \expandafter\bbl@encstring

```



```

1374      \csname\bb1@tempa @bb1@ucl\endcsname{\bb1@tempa##1}%
1375      \expandafter\bb1@encstring
1376      \csname\bb1@tempa @bb1@uc\endcsname{##2}%
1377      \expandafter\bb1@encstring
1378      \csname\bb1@tempa @bb1@lc\endcsname{##3}}}%
1379 <</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.

```

1380 <<{*Macros local to BabelCommands}>> ≡
1381 \newcommand\SetHyphenMap[1]{%
1382   \bb1@forlang\bb1@tempa{%
1383     \expandafter\bb1@stringdef
1384     \csname\bb1@tempa @bb1@hyphenmap\endcsname{##1}}}%
1385 <</Macros local to BabelCommands>>

```

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

```

1386 \newcommand\BabelLower[2]{% one to one.
1387   \ifnum\lccode#1=#2\else
1388     \babel@savevariable{\lccode#1}%
1389     \lccode#1=#2\relax
1390   \fi}
1391 \newcommand\BabelLowerMM[4]{% many-to-many
1392   \@tempcnta=#1\relax
1393   \@tempcntb=#4\relax
1394   \def\bb1@tempa{%
1395     \ifnum\@tempcnta>#2\else
1396       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1397       \advance\@tempcnta#3\relax
1398       \advance\@tempcntb#3\relax
1399       \expandafter\bb1@tempa
1400     \fi}%
1401   \bb1@tempa}
1402 \newcommand\BabelLowerMO[4]{% many-to-one
1403   \@tempcnta=#1\relax
1404   \def\bb1@tempa{%
1405     \ifnum\@tempcnta>#2\else
1406       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1407       \advance\@tempcnta#3
1408       \expandafter\bb1@tempa
1409     \fi}%
1410   \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1411 <<{*More package options}>> ≡
1412 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
1413 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
1414 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
1415 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@}
1416 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1417 <</More package options>>

```

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

```

1418 \AtEndOfPackage{%
1419   \ifx\bb1@opt@hyphenmap\undefined
1420     \bb1@xin{,}{\bb1@language@opts}%
1421     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
1422   \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.

```
1423 \bbl@trace{Macros related to glyphs}
1424 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
1425   \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
1426   \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.

```
1427 \def\save@sf@q#1{\leavevmode
1428   \begingroup
1429   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1430   \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.

```
1431 \ProvideTextCommand{\quotedblbase}{OT1}{%
1432   \save@sf@q{\set@low@box{\textquotedblright\}}%
1433   \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
1434 \ProvideTextCommandDefault{\quotedblbase}{%
1435   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1436 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1437   \save@sf@q{\set@low@box{\textquoteright\}}%
1438   \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
1439 \ProvideTextCommandDefault{\quotesinglbase}{%
1440   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

`\guillemotright`

```
1441 \ProvideTextCommand{\guillemotleft}{OT1}{%
1442   \ifmmode
1443     \ll
1444   \else
1445     \save@sf@q{\nobreak
1446       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1447   \fi}
1448 \ProvideTextCommand{\guillemotright}{OT1}{%
1449   \ifmmode
1450     \gg
1451   \else
```

```

1452 \save@sf@q{\nobreak
1453 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1454 \fi}

```

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

```

1455 \ProvideTextCommandDefault{\guillemotleft}{%
1456 \UseTextSymbol{OT1}{\guillemotleft}}
1457 \ProvideTextCommandDefault{\guillemotright}{%
1458 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

1459 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1460 \ifmmode
1461 <%
1462 \else
1463 \save@sf@q{\nobreak
1464 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1465 \fi}
1466 \ProvideTextCommand{\guilsinglright}{OT1}{%
1467 \ifmmode
1468 >%
1469 \else
1470 \save@sf@q{\nobreak
1471 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1472 \fi}

```

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

```

1473 \ProvideTextCommandDefault{\guilsinglleft}{%
1474 \UseTextSymbol{OT1}{\guilsinglleft}}
1475 \ProvideTextCommandDefault{\guilsinglright}{%
1476 \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.

```

1477 \DeclareTextCommand{\ij}{OT1}{%
1478 i\kern-0.02em\bbl@allowhyphens j}
1479 \DeclareTextCommand{\IJ}{OT1}{%
1480 I\kern-0.02em\bbl@allowhyphens J}
1481 \DeclareTextCommand{\ij}{T1}{\char188}
1482 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

1483 \ProvideTextCommandDefault{\ij}{%
1484 \UseTextSymbol{OT1}{\ij}}
1485 \ProvideTextCommandDefault{\IJ}{%
1486 \UseTextSymbol{OT1}{\IJ}}

```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, but not in the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipcevic Mario, (stipcevic@olimp.irb.hr).

```

1487 \def\crrtic@{\hrule height0.1ex width0.3em}

```

```

1488 \def\crttic@{\hrule height0.1ex width0.33em}
1489 \def\ddj@{%
1490   \setbox0\hbox{d}\dimen@=\ht0
1491   \advance\dimen@1ex
1492   \dimen@.45\dimen@
1493   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1494   \advance\dimen@ii.5ex
1495   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1496 \def\DDJ@{%
1497   \setbox0\hbox{D}\dimen@=.55\ht0
1498   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1499   \advance\dimen@ii.15ex % correction for the dash position
1500   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1501   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1502   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1503 %
1504 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1505 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

1506 \ProvideTextCommandDefault{\dj}{%
1507   \UseTextSymbol{OT1}{\dj}}
1508 \ProvideTextCommandDefault{\DJ}{%
1509   \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.

```

1510 \DeclareTextCommand{\SS}{OT1}{\SS}
1511 \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 1512 \ProvideTextCommandDefault{\glq}{%
1513   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

1514 \ProvideTextCommand{\grq}{T1}{%
1515   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1516 \ProvideTextCommand{\grq}{TU}{%
1517   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1518 \ProvideTextCommand{\grq}{OT1}{%
1519   \save@sf@q{\kern-.0125em
1520     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1521     \kern.07em\relax}}
1522 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

\glqq The ‘german’ double quotes.

```

\grqq 1523 \ProvideTextCommandDefault{\glqq}{%
1524   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```
1525 \ProvideTextCommand{\grqq}{T1}{%
1526   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1527 \ProvideTextCommand{\grqq}{TU}{%
1528   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1529 \ProvideTextCommand{\grqq}{OT1}{%
1530   \save@sf@q{\kern-.07em
1531     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1532     \kern.07em\relax}}
1533 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 1534 \ProvideTextCommandDefault{\flq}{%
1535   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1536 \ProvideTextCommandDefault{\frq}{%
1537   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 1538 \ProvideTextCommandDefault{\flqq}{%
1539   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1540 \ProvideTextCommandDefault{\frqq}{%
1541   \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).

```
1542 \def\umlauthigh{%
1543   \def\bbl@umlauta##1{\leavevmode\bgroup%
1544     \expandafter\accent\csname\fontencoding dqpos\endcsname
1545     ##1\bbl@allowhyphens\egroup}%
1546   \let\bbl@umlaute\bbl@umlauta}
1547 \def\umlautlow{%
1548   \def\bbl@umlauta{\protect\lower@umlaut}}
1549 \def\umlautelow{%
1550   \def\bbl@umlaute{\protect\lower@umlaut}}
1551 \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.

```
1552 \expandafter\ifx\csname U@D\endcsname\relax
1553   \csname newdimen\endcsname U@D
1554 \fi
```

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

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which

the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

1555 \def\lower@umlaut#1{%
1556   \leavevmode\bggroup
1557     \U@D 1ex%
1558     {\setbox\z@\hbox{%
1559       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1560       \dimen@ -.45ex\advance\dimen@\ht\z@
1561       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1562       \expandafter\accent\csname\fontencoding dqpos\endcsname
1563       \fontdimen5\font\U@D #1%
1564     \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).

```

1565 \AtBeginDocument{%
1566   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
1567   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
1568   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
1569   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
1570   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
1571   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
1572   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
1573   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
1574   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
1575   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
1576   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%
1577 }

```

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

```

1578 \ifx\l@english\@undefined
1579   \chardef\l@english\z@
1580 \fi
1581 \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.

```

1582 \bbl@trace{Bidi layout}
1583 \providecommand\IfBabelLayout[3]{#3}%
1584 \newcommand\BabelPatchSection[1]{%
1585   \@ifundefined{#1}{%
1586     \bbl@exp{\let\ss@#1>\<#1>}%
1587     \@namedef{#1}{%
1588       \ifstar{\bbl@presec@s{#1}}%
1589       {\@dblarg{\bbl@presec@x{#1}}}}%
1590   \def\bbl@presec@x#1[#2]#3{%
1591     \bbl@exp{%

```

```

1592   \\select@language@x{\bbl@main@language}%
1593   \\@nameuse{bbl@sspre@#1}%
1594   \\@nameuse{bbl@ss@#1}%
1595   [\\foreignlanguage{\language}{\unexpanded{#2}}]%
1596   {\\foreignlanguage{\language}{\unexpanded{#3}}}%
1597   \\select@language@x{\language}}
1598 \def\bbl@presec@#1#2{%
1599   \bbl@exp{%
1600     \\select@language@x{\bbl@main@language}%
1601     \\@nameuse{bbl@sspre@#1}%
1602     \\@nameuse{bbl@ss@#1}*%
1603     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
1604     \\select@language@x{\language}}
1605 \IfBabelLayout{sectioning}%
1606   {\BabelPatchSection{part}%
1607    \BabelPatchSection{chapter}%
1608    \BabelPatchSection{section}%
1609    \BabelPatchSection{subsection}%
1610    \BabelPatchSection{subsubsection}%
1611    \BabelPatchSection{paragraph}%
1612    \BabelPatchSection{subparagraph}%
1613    \def\babel@toc#1{%
1614      \select@language@x{\bbl@main@language}}}%
1615 \IfBabelLayout{captions}%
1616   {\BabelPatchSection{caption}}}%

```

Now we load definition files for engines.

```

1617 \bbl@trace{Input engine specific macros}
1618 \ifcase\bbl@engine
1619   \input txtbabel.def
1620 \or
1621   \input luababel.def
1622 \or
1623   \input xebabel.def
1624 \fi

```

9.13 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.

```

1625 \bbl@trace{Creating languages and reading ini files}
1626 \newcommand\babelprovide[2][{}]{%
1627   \let\bbl@savelangname\language
1628   \def\language{#2}%
1629   \let\bbl@KVP@captions\@nil
1630   \let\bbl@KVP@import\@nil
1631   \let\bbl@KVP@main\@nil
1632   \let\bbl@KVP@script\@nil
1633   \let\bbl@KVP@language\@nil
1634   \let\bbl@KVP@dir\@nil
1635   \let\bbl@KVP@hyphenrules\@nil
1636   \let\bbl@KVP@mapfont\@nil
1637   \let\bbl@KVP@maparabic\@nil
1638   \let\bbl@KVP@intraspace\@nil
1639   \let\bbl@KVP@intrapenalty\@nil
1640   \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}% TODO - error handling
1641   \ifx\bbl@KVP@import\@nil\else

```

```

1642 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1643 {\begingroup
1644 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1645 \InputIfFileExists{babel-#2.tex}{}}%
1646 \endgroup}%
1647 {}%
1648 \fi
1649 \ifx\bbl@KVP@captions\@nil
1650 \let\bbl@KVP@captions\bbl@KVP@import
1651 \fi
1652 % Load ini
1653 \bbl@ifunset{date#2}%
1654 {\bbl@provide@new{#2}}%
1655 {\bbl@ifblank{#1}%
1656 {\bbl@error
1657 {If you want to modify `#2' you must tell how in\\%
1658 the optional argument. Currently there are three\\%
1659 options: captions=lang-tag, hyphenrules=lang-list\\%
1660 import=lang-tag}%
1661 {Use this macro as documented}}%
1662 {\bbl@provide@renew{#2}}}%
1663 % Post tasks
1664 \bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}%
1665 \bbl@ifunset{\bbl@ensure@\language}%
1666 {\bbl@exp{%
1667 \\\DeclareRobustCommand\<\bbl@ensure@\language>[1]{%
1668 \\\foreignlanguage{\language}%
1669 {###1}}}%
1670 {}%
1671 % To override script and language names
1672 \ifx\bbl@KVP@script\@nil\else
1673 \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1674 \fi
1675 \ifx\bbl@KVP@language\@nil\else
1676 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1677 \fi
1678 % For bidi texts, to switch the language based on direction
1679 \ifx\bbl@KVP@mapfont\@nil\else
1680 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}%
1681 {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
1682 mapfont. Use `direction'.%
1683 {See the manual for details.}}}%
1684 \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
1685 \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
1686 \ifx\bbl@mapselect\@undefined
1687 \AtBeginDocument{%
1688 \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1689 {\selectfont}}%
1690 \def\bbl@mapselect{%
1691 \let\bbl@mapselect\relax
1692 \edef\bbl@prefontid{\fontid\font}}%
1693 \def\bbl@mapdir##1{%
1694 {\def\language{##1}%
1695 \let\bbl@ifrestoring\@firstoftwo % avoid font warning
1696 \bbl@switchfont
1697 \directlua{Babel.fontmap
1698 [\the\csname bbl@wdir@##1\endcsname]%
1699 [\bbl@prefontid]=\fontid\font}}}%
1700 \fi

```



```

1701 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
1702 \fi
1703 % For Southeast Asian, if interspace in ini
1704 \ifcase\bbl@engine\or
1705 \bbl@ifunset{\bbl@intsp@{\language}}{%
1706 {\expandafter\ifx\csname bbl@intsp@{\language}\endcsname\@empty\else
1707 \bbl@seaintraspacespace
1708 \ifx\bbl@KVP@intraspacespace\@nil
1709 \bbl@exp{%
1710 \bbl@intraspacespace\bbl@cs{intsp@{\language}}\@}%
1711 \fi
1712 \directlua{
1713 Babel = Babel or {}
1714 Babel.sea_ranges = Babel.sea_ranges or {}
1715 Babel.set_chranges('\bbl@cs{sbcpr@{\language}}',
1716 '\bbl@cs{chrng@{\language}}')
1717 }
1718 \ifx\bbl@KVP@intrapenalty\@nil
1719 \bbl@intrapenalty0\@
1720 \fi
1721 \fi
1722 \ifx\bbl@KVP@intraspacespace\@nil\else % We may override the ini
1723 \expandafter\bbl@intraspacespace\bbl@KVP@intraspacespace\@
1724 \fi
1725 \ifx\bbl@KVP@intrapenalty\@nil\else
1726 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
1727 \fi}%
1728 \or
1729 \bbl@ifunset{\bbl@intsp@{\language}}{%
1730 {\expandafter\ifx\csname bbl@intsp@{\language}\endcsname\@empty\else
1731 \ifx\bbl@KVP@intraspacespace\@nil
1732 \bbl@exp{%
1733 \bbl@intraspacespace\bbl@cs{intsp@{\language}}\@}%
1734 \fi
1735 \ifx\bbl@KVP@intrapenalty\@nil
1736 \bbl@intrapenalty0\@
1737 \fi
1738 \fi
1739 \ifx\bbl@KVP@intraspacespace\@nil\else % We may override the ini
1740 \expandafter\bbl@intraspacespace\bbl@KVP@intraspacespace\@
1741 \fi
1742 \ifx\bbl@KVP@intrapenalty\@nil\else
1743 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
1744 \fi
1745 \ifx\bbl@ispacesize\@undefined
1746 \AtBeginDocument{%
1747 \expandafter\bbl@add
1748 \csname selectfont \endcsname{\bbl@ispacesize}%
1749 \def\bbl@ispacesize{\bbl@cs{xisp@}\bbl@cs{sbcpr@{\language}}}%
1750 \fi}%
1751 \fi
1752 % Native digits, if provided in ini
1753 \ifcase\bbl@engine\else
1754 \bbl@ifunset{\bbl@dgnat@{\language}}{%
1755 {\expandafter\ifx\csname bbl@dgnat@{\language}\endcsname\@empty\else
1756 \expandafter\expandafter\expandafter
1757 \bbl@setdigits\csname bbl@dgnat@{\language}\endcsname
1758 \ifx\bbl@KVP@maparabic\@nil\else
1759 \ifx\bbl@latinarabic\@undefined

```

```

1760         \expandafter\let\expandafter\@arabic
1761             \csname bbl@counter@\language\endcsname
1762     \else % ie, if layout=counters, which redefines \@arabic
1763         \expandafter\let\expandafter\bbl@latin@arabic
1764             \csname bbl@counter@\language\endcsname
1765     \fi
1766 \fi
1767 \fi}%
1768 \fi
1769 % To load or reload the babel-*.tex, if require.babel in ini
1770 \bbl@ifunset{bbl@rqtex@\language}{}%
1771     {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
1772         \let\BabelBeforeIni\@gobbles
1773         \chardef\atcatcode=\catcode`\@
1774         \catcode`\@=11\relax
1775         \InputIfFileExists{babel-bbl@cs@rqtex@\language}.tex}{%}%
1776         \catcode`\@=\atcatcode
1777         \let\atcatcode\relax
1778     \fi}%
1779 \let\language\bbl@savelangname

```

[illegible]

```
1810 \def\bbl@provide@new#1{%
1811   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1812   \@namedef{extras#1}{}%
```

```

1813 \@namedef{noextras#1}{}%
1814 \StartBabelCommands*{#1}{captions}%
1815 \ifx\bbbl@KVP@captions\@nil % and also if import, implicit
1816 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
1817 \ifx##1\@empty\else
1818 \bbbl@exp{%
1819 \\\SetString\\##1{%
1820 \\\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
1821 \expandafter\bbbl@tempb
1822 \fi}%
1823 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
1824 \else
1825 \bbbl@read@ini{\bbbl@KVP@captions}% Here all letters cat = 11
1826 \bbbl@after@ini
1827 \bbbl@savestrings
1828 \fi
1829 \StartBabelCommands*{#1}{date}%
1830 \ifx\bbbl@KVP@import\@nil
1831 \bbbl@exp{%
1832 \\\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
1833 \else
1834 \bbbl@savetoday
1835 \bbbl@savedate
1836 \fi
1837 \EndBabelCommands
1838 \bbbl@exp{%
1839 \def\<#1hyphenmins>{%
1840 {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\@nameuse{\bbbl@lfthm@#1}}}%
1841 {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\@nameuse{\bbbl@rgthm@#1}}}}}%
1842 \bbbl@provide@hyphens{#1}%
1843 \ifx\bbbl@KVP@main\@nil\else
1844 \expandafter\main@language\expandafter{#1}%
1845 \fi}
1846 \def\bbbl@provide@renew#1{%
1847 \ifx\bbbl@KVP@captions\@nil\else
1848 \StartBabelCommands*{#1}{captions}%
1849 \bbbl@read@ini{\bbbl@KVP@captions}% Here all letters cat = 11
1850 \bbbl@after@ini
1851 \bbbl@savestrings
1852 \EndBabelCommands
1853 \fi
1854 \ifx\bbbl@KVP@import\@nil\else
1855 \StartBabelCommands*{#1}{date}%
1856 \bbbl@savetoday
1857 \bbbl@savedate
1858 \EndBabelCommands
1859 \fi
1860 \bbbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

1861 \def\bbbl@provide@hyphens#1{%
1862 \let\bbbl@tempa\relax
1863 \ifx\bbbl@KVP@hyphenrules\@nil\else
1864 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
1865 \bbbl@foreach\bbbl@KVP@hyphenrules{%
1866 \ifx\bbbl@tempa\relax % if not yet found
1867 \bbbl@ifsamestring{##1}{+}%
1868 {\bbbl@exp{\addlanguage\<1@##1>}}}%
1869 {}%

```

```

1870      \bbl@ifunset{l@##1}%
1871      {}%
1872      {\bbl@exp{\let\bbl@tempa<l@##1>}}%
1873      \fi}%
1874      \fi
1875      \ifx\bbl@tempa\relax %          if no opt or no language in opt found
1876      \ifx\bbl@KVP@import\@nil\else % if importing
1877      \bbl@exp{%                      and hyphenrules is not empty
1878      \\\bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
1879      {}%
1880      {\let\\bbl@tempa<l@\@nameuse{bbl@hyphr@language}>}}%
1881      \fi
1882      \fi
1883      \bbl@ifunset{bbl@tempa}%          ie, relax or undefined
1884      {\bbl@ifunset{l@#1}%              no hyphenrules found - fallback
1885      {\bbl@exp{\\adddialect<l@#1>\language}}%
1886      {}}%
1887      {\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.*

```

1888 \def\bbl@read@ini#1{%
1889   \openin1=babel-#1.ini
1890   \ifeof1
1891     \bbl@error
1892     {There is no ini file for the requested language\\%
1893     (#1). Perhaps you misspelled it or your installation\\%
1894     is not complete.}%
1895     {Fix the name or reinstall babel.}%
1896   \else
1897     \let\bbl@section\@empty
1898     \let\bbl@savestrings\@empty
1899     \let\bbl@savetoday\@empty
1900     \let\bbl@savestate\@empty
1901     \let\bbl@inireader\bbl@iniskip
1902     \bbl@info{Importing data from babel-#1.ini for \language}%
1903     \loop
1904     \if T\ifeof1F\fi T\relax % Trick, because inside \loop
1905       \endlinechar@m@ne
1906       \read1 to \bbl@line
1907       \endlinechar`^^M
1908       \ifx\bbl@line\@empty\else
1909         \expandafter\bbl@iniline\bbl@line\bbl@iniline
1910       \fi
1911     \repeat
1912   \fi}
1913 \def\bbl@iniline#1\bbl@iniline{%
1914   \@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.

```

1915 \def\bbl@iniskip#1\@{%          if starts with ;
1916 \def\bbl@inisec[#1]#2\@{%      if starts with opening bracket
1917   \@nameuse{bbl@secpost@\bbl@section}% ends previous section
1918   \def\bbl@section{#1}%
1919   \@nameuse{bbl@secpre@\bbl@section}% starts current section
1920   \bbl@ifunset{bbl@secline@#1}%
1921   {\let\bbl@inireader\bbl@iniskip}%

```

```
1922 {\bbl@exp{\let\\bbl@inireader<bbl@secline@#1>}}
```

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

```
1923 \def\bbl@inikv#1=#2\@{%      key=value
1924 \bbl@trim@def\bbl@tempa{#1}%
1925 \bbl@trim\toks@{#2}%
1926 \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.

```
1927 \def\bbl@exportkey#1#2#3{%
1928 \bbl@ifunset{bbl@kv@#2}%
1929 {\bbl@csarg\gdef{#1@\language}\bbl@tempa{#3}}%
1930 {\xexpandafter\ifx\csname bbl@kv@#2\endcsname\empty
1931 \bbl@csarg\gdef{#1@\language}\bbl@tempa{#3}}%
1932 \else
1933 \bbl@exp{\global\let\bbl@#1@\language}>\<bbl@kv@#2>%
1934 \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.

```
1935 \let\bbl@secline@identification\bbl@inikv
1936 \def\bbl@secpost@identification{%
1937 \bbl@exportkey{lname}{identification.name.english}{}%
1938 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
1939 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
1940 \bbl@exportkey{sname}{identification.script.name}{}%
1941 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
1942 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
1943 \let\bbl@secline@typography\bbl@inikv
1944 \let\bbl@secline@characters\bbl@inikv
1945 \let\bbl@secline@numbers\bbl@inikv
1946 \def\bbl@after@ini{%
1947 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
1948 \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
1949 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1950 \bbl@exportkey{intsp}{typography.intraspace}{}%
1951 \bbl@exportkey{jstfy}{typography.justify}{w}%
1952 \bbl@exportkey{chrng}{characters.ranges}{}%
1953 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
1954 \bbl@exportkey{rqtex}{identification.require.babel}{}%
1955 \bbl@xin@{0.5}{\@nameuse{bbl@kv@identification.version}}%
1956 \ifin@
1957 \bbl@warning{%
1958 There are neither captions nor date in '\language'.\%
1959 It may not be suitable for proper typesetting, and it\%
1960 could change. Reported}%
1961 \fi
1962 \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
1963 \ifin@
1964 \bbl@warning{%
1965 The '\language' date format may not be suitable\%
1966 for proper typesetting, and therefore it very likely will\%
1967 change in a future release. Reported}%
1968 \fi
1969 \bbl@toglobal\bbl@savetoday
1970 \bbl@toglobal\bbl@savestate}
```

Now captions and captions.licr, depending on the engine. And also for dates. They rely on a few auxiliary macros.

```

1971 \ifcase\bbl@engine
1972   \bbl@csarg\def{secline@captions.licr}#1=#2\@@{%
1973     \bbl@ini@captions@aux{#1}{#2}}
1974   \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%      for defaults
1975     \bbl@ini@dategreg#1...\relax{#2}}
1976   \bbl@csarg\def{secline@date.gregorian.licr}#1=#2\@@{%  override
1977     \bbl@ini@dategreg#1...\relax{#2}}
1978 \else
1979   \def\bbl@secline@captions#1=#2\@@{%
1980     \bbl@ini@captions@aux{#1}{#2}}
1981   \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%
1982     \bbl@ini@dategreg#1...\relax{#2}}
1983 \fi

```

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

```

1984 \def\bbl@ini@captions@aux#1#2{%
1985   \bbl@trim\def\bbl@tempa{#1}%
1986   \bbl@ifblank{#2}%
1987   {\bbl@exp{%
1988     \toks@{\bbl@nocaption{\bbl@tempa}{\language\language\bbl@tempa name}}}%
1989   {\bbl@trim\toks@{#2}}}%
1990   \bbl@exp{%
1991     \bbl@add\bbl@savestrings{%
1992       \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.

```

1993 \bbl@csarg\def{secpredate.gregorian.licr}{%
1994   \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
1995 \def\bbl@ini@dategreg#1.#2.#3.#4\relax#5{% TODO - ignore with 'captions'
1996   \bbl@trim\def\bbl@tempa{#1.#2}%
1997   \bbl@ifsamestring{\bbl@tempa}{months.wide}%
1998   {\bbl@trim\def\bbl@tempa{#3}%
1999     \bbl@trim\toks@{#5}%
2000     \bbl@exp{%
2001       \bbl@add\bbl@savestate{%
2002         \SetString\<month\romannumeral\bbl@tempa name>{\the\toks@}}}%
2003   {\bbl@ifsamestring{\bbl@tempa}{date.long}%
2004     {\bbl@trim\def\bbl@toreplace{#5}%
2005       \bbl@TG@date
2006       \global\bbl@csarg\let{date@\language}\bbl@toreplace
2007       \bbl@exp{%
2008         \gdef\<\language date>{\protect\<\language date >}%
2009         \gdef\<\language date >####1####2####3{%
2010           \bbl@usedategroupttrue
2011           \<\bbl@ensure@\language>{%
2012             \<\bbl@date@\language>{####1}{####2}{####3}}}%
2013         \bbl@add\bbl@savetoday{%
2014           \SetString\<\today{%
2015             \<\language date>{\the\year}{\the\month}{\the\day}}}}}%
2016       {}}}

```

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.

```

2017 \newcommand\BabelDateSpace{\nobreakspace}
2018 \newcommand\BabelDateDot{. \@}
2019 \newcommand\BabelDated[1]{\number#1}
2020 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}

```

```

2021 \newcommand\BabelDateM[1]{\number#1}
2022 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2023 \newcommand\BabelDateMMM[1]{\%
2024   \csname month\romannumeral#1name\endcsname}%
2025 \newcommand\BabelDatey[1]{\number#1}%
2026 \newcommand\BabelDateyy[1]{\%
2027   \ifnum#1<10 0\number#1 %
2028   \else\ifnum#1<100 \number#1 %
2029   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2030   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2031   \else
2032     \bbl@error
2033     {Currently two-digit years are restricted to the\
2034       range 0-9999.}%
2035     {There is little you can do. Sorry.}%
2036   \fi\fi\fi\fi}
2037 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2038 \def\bbl@replace@finish@iii#1{%
2039   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
2040 \def\bbl@TG@date{%
2041   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
2042   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot{}}%
2043   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
2044   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
2045   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
2046   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
2047   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
2048   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
2049   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
2050   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
2051 % Note after \bbl@replace \toks@ contains the resulting string.
2052 % TODO - Using this implicit behavior doesn't seem a good idea.
2053   \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.

```

2054 \def\bbl@provide@lsys#1{%
2055   \bbl@ifunset{bbl@lname@#1}%
2056     {\bbl@ini@ids{#1}}%
2057     {}%
2058   \bbl@csarg\let{lsys@#1}\@empty
2059   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
2060   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
2061   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2062   \bbl@ifunset{bbl@lname@#1}{%
2063     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2064     \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.

```

2065 \def\bbl@ini@ids#1{%
2066   \def\BabelBeforeIni##1##2{%
2067     \begingroup
2068       \bbl@add\bbl@secpost@identification{\closein1 }%
2069       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
2070       \bbl@read@ini{##1}%
2071     \endgroup}% boxed, to avoid extra spaces:

```

```
2072 {\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`.

```
2073 {\def\format{plain}}
2074 \ifx\fmtname\format
2075 \else
2076   \def\format{LaTeX2e}
2077   \ifx\fmtname\format
2078   \else
2079     \aftergroup\endinput
2080   \fi
2081 \fi}
```

10.2 Cross referencing macros

The L^AT_EX 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 T_EXbook [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.

```
2082 %\bbl@redefine\newlabel#1#2{%
2083 % \@safe@activetrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

`\@newl@bel` We need to change the definition of the L^AT_EX-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.

```
2084 << *More package options >> ≡
2085 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2086 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2087 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2088 << /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.

```

2089 \bbl@trace{Cross referencing macros}
2090 \ifx\bbl@opt@safe\empty\else
2091   \def\@newl@bel#1#2#3{%
2092     {\@safe@activestrue
2093       \bbl@ifunset{#1@#2}%
2094         \relax
2095       {\gdef\@multiplelabels{%
2096         \@latex@warning@no@line{There were multiply-defined labels}}}%
2097       \@latex@warning@no@line{Label `#2' multiply defined}}}%
2098     \global\@namedef{#1@#2}{#3}}}
```

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

```

2099 \CheckCommand*\@testdef[3]{%
2100   \def\reserved@a{#3}%
2101   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2102   \else
2103     \@tempswatrue
2104   \fi}
```

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

```

2105 \def\@testdef#1#2#3{%
2106   \@safe@activestrue
```

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

```

2107   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```

2108   \def\bbl@tempb{#3}%
2109   \@safe@activesfalse
```

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

```

2110   \ifx\bbl@tempa\relax
2111   \else
2112     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2113   \fi
```

We do the same for `\bbl@tempb`.

```

2114   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```

2115   \ifx\bbl@tempa\bbl@tempb
2116   \else
2117     \@tempswatrue
2118   \fi}
2119 \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.

```

2120 \bbl@xin@{R}\bbl@opt@safe
2121 \ifin@
2122 \bbl@redefineroast\ref#1{%
2123 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
2124 \bbl@redefineroast\pageref#1{%
2125 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
2126 \else
2127 \let\org@ref\ref
2128 \let\org@pageref\pageref
2129 \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.

```

2130 \bbl@xin@{B}\bbl@opt@safe
2131 \ifin@
2132 \bbl@redefine\@citex[#1]#2{%
2133 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
2134 \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.

```

2135 \AtBeginDocument{%
2136 \@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.)

```

2137 \def\@citex[#1][#2]#3{%
2138 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
2139 \org@citex[#1][#2]{\@tempa}}%
2140 }{}

```

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

```

2141 \AtBeginDocument{%
2142 \@ifpackageloaded{cite}{%
2143 \def\@citex[#1]#2{%
2144 \@safe@activetrue\org@citex[#1][#2]\@safe@activesfalse}%
2145 }{}

```

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

```

2146 \bbl@redefine\nocite#1{%
2147 \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

`\bibcite` The macro that is used in the `.aux` file to define citation labels. When packages such as `natbib` or `cite` are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where `\@safe@activetrue` is in effect. This switch needs to be reset inside

the `\hbox` which contains the citation label. In order to determine during `.aux` file processing which definition of `\bibcite` is needed we define `\bibcite` in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
2148 \bbl@redefine\bibcite{%
2149   \bbl@cite@choice
2150   \bibcite}
```

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

```
2151 \def\bbl@bibcite#1#2{%
2152   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
2153 \def\bbl@cite@choice{%
2154   \global\let\bibcite\bbl@bibcite
```

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

```
2155   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
2156   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
2157   \global\let\bbl@cite@choice\relax}
```

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

```
2158 \AtBeginDocument{\bbl@cite@choice}
```

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

```
2159 \bbl@redefine\@bibitem#1{%
2160   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
2161 \else
2162   \let\org@nocite\nocite
2163   \let\org@@citex\@citex
2164   \let\org@bibcite\bibcite
2165   \let\org@bibitem\@bibitem
2166 \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.

```
2167 \bbl@trace{Marks}
2168 \IfBabelLayout{sectioning}
2169   {\ifx\bbl@opt@headfoot\@nnil
```

```

2170 \g@addto@macro\@resetactivechars{%
2171 \set@typeset@protect
2172 \expandafter\select@language@x\expandafter{\bbl@main@language}%
2173 \let\protect\noexpand
2174 \edef\thepage{%
2175 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2176 \fi}
2177 {\bbl@redefine\markright#1{%
2178 \bbl@ifblank{#1}%
2179 {\org@markright{}}%
2180 {\toks@{#1}%
2181 \bbl@exp{%
2182 \\\org@markright{\\\protect\\foreignlanguage{\language}%
2183 {\\\protect\\bbl@restore@actives\the\toks@}}}}}%

```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers. The documentclasses `report` and `book` define and set the headings for the page. While doing so they also store a copy of `\markboth` in `\@mkboth`. Therefore we need to check whether `\@mkboth` has already been set. If so we need to do that again with the new definition of `\markboth`.

```

2184 \ifx\@mkboth\markboth
2185 \def\bbl@tempc{\let\@mkboth\markboth}
2186 \else
2187 \def\bbl@tempc{}
2188 \fi

```

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

```

2189 \bbl@redefine\markboth#1#2{%
2190 \protected@edef\bbl@tempb##1{%
2191 \protect\foreignlanguage
2192 {\language}\{\protect\bbl@restore@actives##1}}%
2193 \bbl@ifblank{#1}%
2194 {\toks@{}}%
2195 {\toks@\expandafter{\bbl@tempb{#1}}}%
2196 \bbl@ifblank{#2}%
2197 {\@temptokena{}}%
2198 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2199 \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}

```

and copy it to `\@mkboth` if necessary.

```

2200 \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.

```
2201 \bbl@trace{Preventing clashes with other packages}
2202 \bbl@xin@{R}\bbl@opt@safe
2203 \ifin@
2204 \AtBeginDocument{%
2205   \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```
2206   \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.

```
2207   \let\bbl@temp@pref\pageref
2208   \let\pageref\bbl@temp@pref
2209   \let\bbl@temp@ref\ref
2210   \let\ref\bbl@temp@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.

```
2211   \@safe@activestrue
2212   \org@ifthenelse{#1}%
2213   {\let\pageref\bbl@temp@pref
2214    \let\ref\bbl@temp@ref
2215    \@safe@activesfalse
2216    #2}%
2217   {\let\pageref\bbl@temp@pref
2218    \let\ref\bbl@temp@ref
2219    \@safe@activesfalse
2220    #3}%
2221   }%
2222   }{}%
2223 }
```

10.4.2 varioref

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

```
\vrefpagemum
\Ref
2224 \AtBeginDocument{%
2225   \@ifpackageloaded{varioref}{%
2226     \bbl@redefine\@@vpageref#1[#2]#3{%
2227       \@safe@activestrue
2228       \org@@vpageref{#1}[#2]#3}%
2229       \@safe@activesfalse}%
```

The same needs to happen for `\vrefpagemum`.

```
2230   \bbl@redefine\vrefpagemum#1#2{%
2231     \@safe@activestrue
2232     \org\vrefpagemum{#1}#2}%
2233     \@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.

```
2234 \expandafter\def\csname Ref\endcsname#1{%
2235 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2236 }{}%
2237 }
2238 \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.

```
2239 \AtEndOfPackage{%
2240 \AtBeginDocument{%
2241 \ifpackageloaded{hhline}%
```

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

```
2242 {\expandafter\ifx\csname normal@char:string\endcsname\relax
2243 \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.

```
2244 \makeatletter
2245 \def\@currname{hhline}\input{hhline.sty}\makeatother
2246 \fi}%
2247 {}}}
```

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.

```
2248 \AtBeginDocument{%
2249 \ifx\pdfstringdefDisableCommands\undefined\else
2250 \pdfstringdefDisableCommands{\languageshorthands{system}}%
2251 \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 `\MakeUppercase`. To prevent unexpected results we need to define `\FOREIGNLANGUAGE` here.

```
2252 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2253 \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.

```
2254 \def\substitutefontfamily#1#2#3{%
2255 \lowercase{\immediate\openout15=#1#2.fd\relax}%
2256 \immediate\write15{%
```

```

2257 \string\ProvidesFile{#1#2.fd}%
2258 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2259 \space generated font description file]^^J
2260 \string\DeclareFontFamily{#1}{#2}{ }^^J
2261 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^^J
2262 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^^J
2263 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^^J
2264 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^^J
2265 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^^J
2266 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^^J
2267 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^^J
2268 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^^J
2269 }%
2270 \closeout15
2271 }

```

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

```
2272 \@onlypreamble\substitutefontfamily
```

10.5 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ and $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 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`

```

2273 \bbl@trace{Encoding and fonts}
2274 \newcommand\BabelNonASCII
2275   {LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,TS1,T3,TS3}
2276 \let\org@TeX\TeX
2277 \let\org@LaTeX\LaTeX
2278 \let\ensureascii\@firstofone
2279 \AtBeginDocument{%
2280   \in@false
2281   \bbl@foreach\BabelNonASCII{% is there a non-ascii enc?
2282     \ifin@
2283       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
2284     \fi}%
2285   \ifin@ % if a non-ascii has been loaded
2286     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2287     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2288     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2289     \def\bbl@tempb#1@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
2290     \def\bbl@tempc#1ENC.DEF#2\@{\@%
2291       \ifx\@empty#2\else
2292         \bbl@ifunset{T@#1}%
2293         {}%
2294         {\bbl@xin@{,#1,}{,\BabelNonASCII,}}%
2295       \ifin@
2296         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2297         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2298       \else
2299         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
2300       \fi}%

```

```

2301     \fi}%
2302     \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
2303     \bbl@xin@{\, \cf@encoding,}{, \BabelNonASCII,}%
2304     \fin@else
2305         \edef\ensureascii#1{%
2306             \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2307     \fi
2308 \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.

```

2309 \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.

```

2310 \AtBeginDocument{%
2311     \@ifpackageloaded{fontspec}%
2312     {\xdef\latinencoding{%
2313         \ifx\UTFencname\@undefined
2314             EU\ifcase\bbl@engine\or2\or1\fi
2315         \else
2316             \UTFencname
2317         \fi}}%
2318     {\gdef\latinencoding{OT1}%
2319         \ifx\cf@encoding\bbl@t@one
2320             \xdef\latinencoding{\bbl@t@one}%
2321         \else
2322             \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
2323     \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.

```

2324 \DeclareRobustCommand{\latintext}{%
2325     \fontencoding{\latinencoding}\selectfont
2326     \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.

```

2327 \ifx\@undefined\DeclareTextFontCommand
2328     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2329 \else
2330     \DeclareTextFontCommand{\textlatin}{\latintext}
2331 \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 T_EX 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 LuaT_EX-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>>).

```

2332 \bbl@trace{Basic (internal) bidi support}
2333 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2334 \def\bbl@rscripts{%
2335   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2336   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
2337   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
2338   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2339   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2340   Old South Arabian,}%
2341 \def\bbl@provide@dirs#1{%
2342   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2343   \ifin@
2344     \global\bbl@csarg\chardef{wdir@#1}\@ne
2345     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2346     \ifin@
2347       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2348       \fi
2349     \else
2350       \global\bbl@csarg\chardef{wdir@#1}\z@
2351       \fi}
2352 \def\bbl@switchdir{%
2353   \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}}%
2354   \bbl@ifunset{bbl@wdir@language}{\bbl@provide@dirs{language}}}%
2355   \bbl@exp{\bbl@setdirs\bbl@cs{wdir@language}}}%
2356 \def\bbl@setdirs#1{% TODO - math
2357   \ifcase\bbl@select@type % TODO - strictly, not the right test
2358     \bbl@bodydir{#1}%
2359     \bbl@pardir{#1}%
2360   \fi
2361   \bbl@textdir{#1}}
2362 \ifodd\bbl@engine % luatex=1
2363   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2364   \DisableBabelHook{babel-bidi}
2365   \chardef\bbl@thepardir\z@
2366   \def\bbl@getluadir#1{%
2367     \directlua{

```

```

2368     if tex.#1dir == 'TLT' then
2369         tex.sprint('0')
2370     elseif tex.#1dir == 'TRT' then
2371         tex.sprint('1')
2372     end}}
2373 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
2374     \ifcase#3\relax
2375         \ifcase\bbl@getluadir{#1}\relax\else
2376             #2 TLT\relax
2377         \fi
2378     \else
2379         \ifcase\bbl@getluadir{#1}\relax
2380             #2 TRT\relax
2381         \fi
2382     \fi}
2383 \def\bbl@textdir#1{%
2384     \bbl@setluadir{text}\textdir{#1}% TODO - ?\linedir
2385     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2386 \def\bbl@pardir#1{\bbl@setluadir{par}\pardir{#1}%
2387     \chardef\bbl@thepardir#1\relax}
2388 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2389 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2390 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
2391 \else % pdftex=0, xetex=2
2392     \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2393     \DisableBabelHook{babel-bidi}
2394     \newcount\bbl@dirlevel
2395     \chardef\bbl@thetextdir\z@
2396     \chardef\bbl@thepardir\z@
2397     \def\bbl@textdir#1{%
2398         \ifcase#1\relax
2399             \chardef\bbl@thetextdir\z@
2400             \bbl@textdir@i\beginL\endL
2401         \else
2402             \chardef\bbl@thetextdir\@ne
2403             \bbl@textdir@i\beginR\endR
2404         \fi}
2405     \def\bbl@textdir@i#1#2{%
2406         \ifhmode
2407             \ifnum\currentgrouplevel>\z@
2408                 \ifnum\currentgrouplevel=\bbl@dirlevel
2409                     \bbl@error{Multiple bidi settings inside a group}%
2410                     {I'll insert a new group, but expect wrong results.}%
2411                     \bgroup\aftergroup#2\aftergroup\egroup
2412                 \else
2413                     \ifcase\currentgrouptype\or % 0 bottom
2414                         \aftergroup#2% 1 simple {}
2415                     \or
2416                         \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2417                     \or
2418                         \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2419                     \or\or\or % vbox vtop align
2420                     \or
2421                         \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2422                     \or\or\or\or\or\or % output math disc insert vcent mathchoice
2423                     \or
2424                         \aftergroup#2% 14 \begingroup
2425                     \else
2426                         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj

```

```

2427         \fi
2428     \fi
2429     \bbl@dirlevel\currentgrouplevel
2430     \fi
2431     #1%
2432     \fi}
2433 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2434 \let\bbl@bodydir\@gobble
2435 \let\bbl@pagedir\@gobble
2436 \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).

```

2437 \def\bbl@xebidipar{%
2438     \let\bbl@xebidipar\relax
2439     \TeXeTstate\@ne
2440     \def\bbl@xeverypar{%
2441         \ifcase\bbl@thepardir
2442             \ifcase\bbl@thetextdir\else\beginR\fi
2443         \else
2444             {\setbox\z@\lastbox\beginR\box\z@}%
2445         \fi}%
2446     \let\bbl@severypar\everypar
2447     \newtoks\everypar
2448     \everypar=\bbl@severypar
2449     \bbl@severypar{\bbl@xeverypar\the\everypar}}
2450 \fi

```

A tool for weak L (mainly digits).

```

2451 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}

```

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 `norsk.cfg` will be loaded when the language definition file `norsk.ldf` is loaded.

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

```

2452 \bbl@trace{Local Language Configuration}
2453 \ifx\loadlocalcfg\@undefined
2454     \@ifpackagewith{babel}{noconfigs}%
2455     {\let\loadlocalcfg\@gobble}%
2456     {\def\loadlocalcfg#1{%
2457         \InputIfFileExists{#1.cfg}%
2458         {\typeout{*****^J%
2459                 * Local config file #1.cfg used^^J%
2460                 *}}%
2461         \@empty}}
2462 \fi

```

Just to be compatible with \LaTeX 2.09 we add a few more lines of code:

```

2463 \ifx\@unexpandable@protect\@undefined
2464     \def\@unexpandable@protect{\noexpand\protect\noexpand}
2465     \long\def\protected@write#1#2#3{%
2466         \begingroup

```

```

2467 \let\thepage\relax
2468 #2%
2469 \let\protect\@unexpandable@protect
2470 \edef\reserved@a{\write#1{#3}}%
2471 \reserved@a
2472 \endgroup
2473 \if@nbreak\ifvmode\nobreak\fi\fi}
2474 \fi
2475 </core>
2476 <*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.

```

2477 <<Make sure ProvidesFile is defined>>
2478 \ProvidesFile{switch.def}[\<date>] [\<version>] Babel switching mechanism]
2479 <<Load macros for plain if not LaTeX>>
2480 <<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.

```

2481 \def\bbl@version{\<version>}
2482 \def\bbl@date{\<date>}
2483 \def\adddialect#1#2{%
2484 \global\chardef#1#2\relax
2485 \bbl@usehooks{adddialect}{\#1}{\#2}}%
2486 \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.

```

2487 \def\bbl@fixname#1{%
2488 \begingroup
2489 \def\bbl@tempe{l@}%
2490 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2491 \bbl@tempd
2492 {\lowercase\expandafter{\bbl@tempd}%
2493 {\uppercase\expandafter{\bbl@tempd}%
2494 \@empty
2495 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2496 \uppercase\expandafter{\bbl@tempd}}}%
2497 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2498 \lowercase\expandafter{\bbl@tempd}}}%
2499 \@empty
2500 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2501 \bbl@tempd}
2502 \def\bbl@iflanguage#1{%
2503 \@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.

```

2504 \def\iflanguage#1{%
2505   \bbl@iflanguage{#1}{%
2506     \ifnum\csname l@#1\endcsname=\language
2507       \expandafter\@firstoftwo
2508     \else
2509       \expandafter\@secondoftwo
2510     \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.

```

2511 \let\bbl@select@type\z@
2512 \edef\selectlanguage{%
2513   \noexpand\protect
2514   \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`.

```

2515 \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.

```

2516 \ifx\documentclass\@undefined
2517   \def\xstring{\string\string\string}
2518 \else
2519   \let\xstring\string
2520 \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 \TeX 's aftergroup mechanism to help us.

The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

`\bbl@language@stack` The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```
2521 \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:

```
2522 \def\bbl@push@language{%
2523   \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.

```
2524 \def\bbl@pop@lang#1+#2-#3{%
2525   \edef\language{#1}\xdef#3{#2}}
```

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

```
2526 \let\bbl@ifrestoring\@secondoftwo
2527 \def\bbl@pop@language{%
2528   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2529   \let\bbl@ifrestoring\@firstoftwo
2530   \expandafter\bbl@set@language\expandafter{\language}%
2531   \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.

```
2532 \expandafter\def\csname selectlanguage \endcsname#1{%
2533   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
2534   \bbl@push@language
2535   \aftergroup\bbl@pop@language
2536   \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 not well defined. 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.

```

2537 \def\BabelContentsFiles{toc,lof,lot}
2538 \def\bbl@set@language#1{%
2539   \edef\language{%
2540     \ifnum\escapechar=\expandafter`\string#1\@empty
2541     \else\string#1\@empty\fi}%
2542   \select@language{\language}%
2543   \expandafter\ifx\csname date\language\endcsname\relax\else
2544     \if@filesw
2545       \protected@write\@auxout{}\string\babel@aux{\language}{}%
2546       \bbl@usehooks{write}{}%
2547     \fi
2548   \fi}
2549 \def\select@language#1{%
2550   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2551   \edef\language{#1}%
2552   \bbl@fixname\language
2553   \bbl@iflanguage\language{%
2554     \expandafter\ifx\csname date\language\endcsname\relax
2555       \bbl@error
2556       {Unknown language `#1'. Either you have\\%
2557        misspelled its name, it has not been installed,\\%
2558        or you requested it in a previous run. Fix its name,\\%
2559        install it or just rerun the file, respectively. In\\%
2560        some cases, you may need to remove the aux file}%
2561       {You may proceed, but expect wrong results}%
2562     \else
2563       \let\bbl@select@type\z@
2564       \expandafter\bbl@switch\expandafter{\language}%
2565     \fi}}
2566 \def\babel@aux#1#2{%
2567   \expandafter\ifx\csname date#1\endcsname\relax
2568     \expandafter\ifx\csname bbl@auxwarn#1\endcsname\relax
2569       \@namedef{bbl@auxwarn#1}{}%
2570       \bbl@warning
2571       {Unknown language `#1'. Very likely you\\%
2572        requested it in a previous run. Expect some\\%
2573        wrong results in this run, which should vanish\\%
2574        in the next one. Reported}%
2575     \fi
2576   \else
2577     \select@language{#1}%
2578     \bbl@foreach\BabelContentsFiles{%
2579       \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
2580   \fi}
2581 \def\babel@toc#1#2{%
2582   \select@language{#1}}

```

A bit of optimization. Select in heads/foots the language only if necessary. The real thing is in `babel.def`.

```

2583 \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.

```

2584 \newif\ifbbl@usedategroup
2585 \def\bbl@switch#1{%
2586   \originalTeX
2587   \expandafter\def\expandafter\originalTeX\expandafter{%
2588     \csname noextras#1\endcsname
2589     \let\originalTeX\empty
2590     \babel@beginsave}%
2591   \bbl@usehooks{afterreset}}}%
2592   \languageshorthands{none}%
2593   \ifcase\bbl@select@type
2594     \ifhmode
2595       \hskip\z@skip % trick to ignore spaces
2596       \csname captions#1\endcsname\relax
2597       \csname date#1\endcsname\relax
2598       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2599     \else
2600       \csname captions#1\endcsname\relax
2601       \csname date#1\endcsname\relax
2602     \fi
2603   \else\ifbbl@usedategroup
2604     \bbl@usedategroupfalse
2605     \ifhmode
2606       \hskip\z@skip % trick to ignore spaces
2607       \csname date#1\endcsname\relax
2608       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2609     \else
2610       \csname date#1\endcsname\relax
2611     \fi
2612   \fi\fi
2613   \bbl@usehooks{beforeextras}}}%
2614   \csname extras#1\endcsname\relax
2615   \bbl@usehooks{afterextras}}}%
2616   \ifcase\bbl@opt@hyphenmap\or
2617     \def\BabelLower##1##2{\lccode##1=##2\relax}%
2618     \ifnum\bbl@hymapsel>4\else
2619       \csname\language @bbl@hyphenmap\endcsname
2620     \fi
2621     \chardef\bbl@opt@hyphenmap\z@
2622   \else
2623     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2624       \csname\language @bbl@hyphenmap\endcsname
2625     \fi
2626   \fi
2627   \global\let\bbl@hymapsel@cclv
2628   \bbl@patterns{#1}%
2629   \babel@savevariable\lefthyphenmin
2630   \babel@savevariable\righthyphenmin
2631   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2632     \set@hyphenmins\tw@\thr@@\relax

```



```

2633 \else
2634 \expandafter\expandafter\expandafter\set@hyphenmins
2635 \csname #1hyphenmins\endcsname\relax
2636 \fi}

```

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

```

2637 \long\def\otherlanguage#1{%
2638 \ifnum\bb1@hymapsel=\cclv\let\bb1@hymapsel\thr@@\fi
2639 \csname selectlanguage\endcsname{#1}%
2640 \ignorespaces}

```

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

```

2641 \long\def\endotherlanguage{%
2642 \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`.

```

2643 \expandafter\def\csname otherlanguage*\endcsname#1{%
2644 \ifnum\bb1@hymapsel=\cclv\chardef\bb1@hymapsel4\relax\fi
2645 \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”.

```

2646 \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`.

`\bb1@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`.

```

2647 \providecommand\bbl@beforeforeign{}
2648 \edef\foreignlanguage{%
2649   \noexpand\protect
2650   \expandafter\noexpand\csname foreignlanguage \endcsname}
2651 \expandafter\def\csname foreignlanguage \endcsname{%
2652   \@ifstar\bbl@foreign@s\bbl@foreign@x}
2653 \def\bbl@foreign@x#1#2{%
2654   \begingroup
2655     \let\BabelText\@firstofone
2656     \bbl@beforeforeign
2657     \foreign@language{#1}%
2658     \bbl@usehooks{foreign}{}%
2659     \BabelText{#2}% Now in horizontal mode!
2660   \endgroup}
2661 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
2662   \begingroup
2663     {\par}%
2664     \let\BabelText\@firstofone
2665     \foreign@language{#1}%
2666     \bbl@usehooks{foreign*}{}%
2667     \bbl@dirparastext
2668     \BabelText{#2}% Still in vertical mode!
2669     {\par}%
2670   \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`.

```

2671 \def\foreign@language#1{%
2672   \edef\language#1%
2673   \bbl@fixname\language
2674   \bbl@iflanguage\language%
2675     \expandafter\ifx\csname date\language\endcsname\relax
2676     \bbl@warning
2677       {Unknown language `#1'. Either you have\\%
2678        misspelled its name, it has not been installed,\\%
2679        or you requested it in a previous run. Fix its name,\\%
2680        install it or just rerun the file, respectively.\\%
2681        I'll proceed, but expect wrong results.\\%
2682        Reported}%
2683   \fi
2684   \let\bbl@select@type\@ne
2685   \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 `\lcode'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.

```

2686 \let\bbl@hyphlist\@empty
2687 \let\bbl@hyphenation@\relax
2688 \let\bbl@pttnlist\@empty
2689 \let\bbl@patterns@\relax

```

```

2690 \let\bbl@hymapsel=\@cclv
2691 \def\bbl@patterns#1{%
2692   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2693     \csname l@#1\endcsname
2694     \edef\bbl@tempa{#1}%
2695   \else
2696     \csname l@#1:\f@encoding\endcsname
2697     \edef\bbl@tempa{#1:\f@encoding}%
2698   \fi
2699   \@expandtwoargs\bbl@usehooks{patterns}{#{1}}{\bbl@tempa}}%
2700   \@ifundefined{bbl@hyphenation@}{ }{% Can be \relax!
2701     \begingroup
2702       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
2703       \ifin@else
2704         \@expandtwoargs\bbl@usehooks{hyphenation}{#{1}}{\bbl@tempa}}%
2705       \hyphenation{%
2706         \bbl@hyphenation@
2707         \@ifundefined{bbl@hyphenation@#1}%
2708         \@empty
2709         {\space\csname bbl@hyphenation@#1\endcsname}}%
2710       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2711     \fi
2712   \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*.

```

2713 \def\hyphenrules#1{%
2714   \edef\bbl@tempf{#1}%
2715   \bbl@fixname\bbl@tempf
2716   \bbl@iflanguage\bbl@tempf{%
2717     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2718     \languageshortands{none}%
2719     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2720       \set@hyphenmins\tw@\thr@@\relax
2721     \else
2722       \expandafter\expandafter\expandafter\set@hyphenmins
2723       \csname\bbl@tempf hyphenmins\endcsname\relax
2724     \fi}}
2725 \let\endhyphenrules\@empty

```

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

```

2726 \def\providehyphenmins#1#2{%
2727   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2728     \@namedef{#1hyphenmins}{#2}%
2729   \fi}

```

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

```

2730 \def\set@hyphenmins#1#2{%
2731   \lefthyphenmin#1\relax
2732   \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.

```

2733 \ifx\ProvidesFile\undefined
2734 \def\ProvidesLanguage#1[#2 #3 #4]{%
2735   \wlog{Language: #1 #4 #3 <#2>}%
2736 }
2737 \else
2738 \def\ProvidesLanguage#1{%
2739   \begingroup
2740     \catcode`\ 10 %
2741     \@makeother\/%
2742     \ifnextchar[%]
2743       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2744 \def\@provideslanguage#1[#2]{%
2745   \wlog{Language: #1 #2}%
2746   \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2747   \endgroup}
2748 \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`

```

2749 \def\LdfInit{%
2750   \chardef\atcatcode=\catcode`\@
2751   \catcode`\@=11\relax
2752   \input babel.def\relax
2753   \catcode`\@=\atcatcode \let\atcatcode\relax
2754   \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`.

```

2755 \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.

```

2756 \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’:

```

2757 \providecommand\setlocale{%
2758   \bbl@error
2759   {Not yet available}%
2760   {Find an armchair, sit down and wait}}
2761 \let\uselocale\setlocale
2762 \let\locale\setlocale
2763 \let\selectlocale\setlocale
2764 \let\textlocale\setlocale
2765 \let\textlanguage\setlocale
2766 \let\languagetext\setlocale

```

11.2 Errors

`\@nolanerr` The babel package will signal an error when a documents tries to select a language that
`\@nopatterns` hasn’t been defined earlier. When a user selects a language for which no hyphenation

patterns were loaded into the format he will be given a warning about that fact. We revert to the patterns for `\language=0` in that case. In most formats that will be (US)english, but it might also be empty.

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.
When the format knows about `\PackageError` it must be $\text{\LaTeX} 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

```

2767 \edef\bbl@nulllanguage{\string\language=0}
2768 \ifx\PackageError\undefined
2769   \def\bbl@error#1#2{%
2770     \begingroup
2771       \newlinechar=`^^J
2772       \def\{^^J(babel) }%
2773       \errhelp{#2}\errmessage{\{#1}%
2774     \endgroup}
2775   \def\bbl@warning#1{%
2776     \begingroup
2777       \newlinechar=`^^J
2778       \def\{^^J(babel) }%
2779       \message{\{#1}%
2780     \endgroup}
2781   \def\bbl@info#1{%
2782     \begingroup
2783       \newlinechar=`^^J
2784       \def\{^^J}%
2785       \wlog{#1}%
2786     \endgroup}
2787 \else
2788   \def\bbl@error#1#2{%
2789     \begingroup
2790       \def\{\MessageBreak}%
2791       \PackageError{babel}{#1}{#2}%
2792     \endgroup}
2793   \def\bbl@warning#1{%
2794     \begingroup
2795       \def\{\MessageBreak}%
2796       \PackageWarning{babel}{#1}%
2797     \endgroup}
2798   \def\bbl@info#1{%
2799     \begingroup
2800       \def\{\MessageBreak}%
2801       \PackageInfo{babel}{#1}%
2802     \endgroup}
2803 \fi
2804 \@ifpackagewith{babel}{silent}
2805   {\let\bbl@info@gobble
2806    \let\bbl@warning@gobble}
2807 {}
2808 \def\bbl@nocaption{\protect\bbl@nocaption@i}
2809 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
2810   \global\@namedef{#2}{\textbf{?#1?}}%
2811   \@nameuse{#2}%
2812   \bbl@warning{%
2813     \@backslashchar#2 not set. Please, define\%
2814     it in the preamble with something like:\%
2815     \string\renewcommand\@backslashchar#2{..\}%
2816     Reported}}

```

```

2817 \def\bbl@tentative{\protect\bbl@tentative@i}
2818 \def\bbl@tentative@i#1{%
2819   \bbl@warning{%
2820     Some functions in '#1' are tentative.\%
2821     They might not work as expected and their behavior\%
2822     could change in the future.\%
2823     Reported}}
2824 \def\@nolanerr#1{%
2825   \bbl@error
2826     {You haven't defined the language #1\space yet}%
2827     {Your command will be ignored, type <return> to proceed}}
2828 \def\@nopatterns#1{%
2829   \bbl@warning
2830     {No hyphenation patterns were preloaded for\%
2831     the language '#1' into the format.\%
2832     Please, configure your TeX system to add them and\%
2833     rebuild the format. Now I will use the patterns\%
2834     preloaded for \bbl@nulllanguage\space instead}}
2835 \let\bbl@usehooks\@gobbletwo
2836 \</kernel>
2837 \<*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.

```

2838 <<Make sure ProvidesFile is defined>>
2839 \ProvidesFile{hyphen.cfg}[\<<date>>] [\<<version>>] Babel hyphens]
2840 \xdef\bbl@format{\jobname}
2841 \ifx\AtBeginDocument\@undefined
2842   \def\@empty{}
2843   \let\orig@dump\dump
2844   \def\dump{%
2845     \ifx\@ztryfc\@undefined
2846       \else
2847         \toks0=\expandafter{\@preamblecmds}%
2848         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
2849         \def\@begindocumenthook{}%
2850       \fi
2851       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
2852 \fi
2853 <<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.

```

2854 \def\process@line#1#2 #3 #4 {%
2855   \ifx=#1%
2856     \process@synonym{#2}%
2857   \else
2858     \process@language{#1#2}{#3}{#4}%
2859   \fi
2860   \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.

```

2861 \toks@{}
2862 \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.

```

2863 \def\process@synonym#1{%
2864   \ifnum\last@language=\m@ne
2865     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
2866   \else
2867     \expandafter\chardef\csname l@#1\endcsname\last@language
2868     \wlog{\string\l@#1=\string\language\the\last@language}%
2869     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
2870     \csname\language\endcsname hyphenmins\endcsname
2871     \let\bbl@elt\relax
2872     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
2873   \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.

```

2874 \def\process@language#1#2#3{%
2875   \expandafter\addlanguage\csname l@#1\endcsname
2876   \expandafter\language\csname l@#1\endcsname
2877   \edef\language#1#2#3{%
2878     \bbl@hook@everylanguage{#1}%
2879     \bbl@get@enc#1::\@@@
2880     \begingroup
2881       \lefthyphenmin\m@ne
2882       \bbl@hook@loadpatterns{#2}%
2883       \ifnum\lefthyphenmin=\m@ne
2884         \else
2885           \expandafter\xdef\csname #1hyphenmins\endcsname{%
2886             \the\lefthyphenmin\the\righthyphenmin}%
2887           \fi
2888     \endgroup
2889     \def\bbl@tempa{#3}%
2890     \ifx\bbl@tempa\@empty\else
2891       \bbl@hook@loadexceptions{#3}%
2892     \fi
2893     \let\bbl@elt\relax
2894     \edef\bbl@languages{%
2895       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
2896     \ifnum\the\language=\z@
2897       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2898         \set@hyphenmins\tw@\thr@@\relax
2899       \else
2900         \expandafter\expandafter\expandafter\set@hyphenmins
2901           \csname #1hyphenmins\endcsname

```



```

2902 \fi
2903 \the\toks@
2904 \toks@{}%
2905 \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.

```

2906 \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.

```

2907 \def\bbl@hook@everylanguage#1{}
2908 \def\bbl@hook@loadpatterns#1{\input #1\relax}
2909 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
2910 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
2911 \begingroup
2912 \def\AddBabelHook#1#2{%
2913 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
2914 \def\next{\toks1}%
2915 \else
2916 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
2917 \fi
2918 \next}
2919 \ifx\directlua@undefined
2920 \ifx\XeTeXinputencoding@undefined\else
2921 \input xebabel.def
2922 \fi
2923 \else
2924 \input luababel.def
2925 \fi
2926 \openin1 = babel-\bbl@format.cfg
2927 \ifeof1
2928 \else
2929 \input babel-\bbl@format.cfg\relax
2930 \fi
2931 \closein1
2932 \endgroup
2933 \bbl@hook@loadkernel{switch.def}

```

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

```

2934 \openin1 = language.dat

```

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

```

2935 \def\language{english}%
2936 \ifeof1
2937 \message{I couldn't find the file language.dat,\space
2938 I will try the file hyphen.tex}
2939 \input hyphen.tex\relax
2940 \chardef\l@english\z@
2941 \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`.

```

2942 \last@language\m@ne

```

We now read lines from the file until the end is found

```
2943 \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.

```
2944 \endlinechar\m@ne
2945 \read1 to \bbl@line
2946 \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.

```
2947 \if T\ifeof1F\fi T\relax
2948 \ifx\bbl@line\@empty\else
2949 \edef\bbl@line{\bbl@line\space\space\space}%
2950 \expandafter\process@line\bbl@line\relax
2951 \fi
2952 \repeat
```

Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns.

```
2953 \begingroup
2954 \def\bbl@elt#1#2#3#4{%
2955 \global\language=#2\relax
2956 \gdef\language{#1}%
2957 \def\bbl@elt##1##2##3##4{}}%
2958 \bbl@languages
2959 \endgroup
2960 \fi
```

and close the configuration file.

```
2961 \closein1
```

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

```
2962 \if/\the\toks@\else
2963 \errhelp{language.dat loads no language, only synonyms}
2964 \errmessage{Orphan language synonym}
2965 \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.

```
2966 \let\bbl@line\@undefined
2967 \let\process@line\@undefined
2968 \let\process@synonym\@undefined
2969 \let\process@language\@undefined
2970 \let\bbl@get@enc\@undefined
2971 \let\bbl@hyph@enc\@undefined
2972 \let\bbl@tempa\@undefined
2973 \let\bbl@hook@loadkernel\@undefined
2974 \let\bbl@hook@everylanguage\@undefined
2975 \let\bbl@hook@loadpatterns\@undefined
2976 \let\bbl@hook@loadexceptions\@undefined
2977 \loadpatterns
```

Here the code for iniT_EX ends.

13 Font handling with fontspec

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

```
2978 <<{*More package options}>> ≡
2979 \ifodd\bb1@engine
2980   \DeclareOption{bidi=basic-r}%
2981     {\ExecuteOptions{bidi=basic}}
2982   \DeclareOption{bidi=basic}%
2983     {\let\bb1@beforeforeign\leavevmode
2984       \newattribute\bb1@attr@dir
2985       \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
2986     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
2987 \else
2988   \DeclareOption{bidi=basic-r}%
2989     {\ExecuteOptions{bidi=basic}}
2990   \DeclareOption{bidi=basic}%
2991     {\bb1@error
2992       {The bidi method 'basic' is available only in\\%
2993         luatex. I'll continue with 'bidi=default', so\\%
2994         expect wrong results}%
2995       {See the manual for further details.}%
2996       \let\bb1@beforeforeign\leavevmode
2997       \AtEndOfPackage{%
2998         \EnableBabelHook{babel-bidi}%
2999         \bb1@xebidipar}}
3000 \fi
3001 \DeclareOption{bidi=default}%
3002   {\let\bb1@beforeforeign\leavevmode
3003     \ifodd\bb1@engine
3004       \newattribute\bb1@attr@dir
3005       \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
3006   \fi
3007   \AtEndOfPackage{%
3008     \EnableBabelHook{babel-bidi}%
3009     \ifodd\bb1@engine\else
3010       \bb1@xebidipar
3011     \fi}}
3012 <</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.

```
3013 <<{*Font selection}>> ≡
3014 \bb1@trace{Font handling with fontspec}
3015 \@onlypreamble\babelfont
3016 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
3017   \edef\bb1@tempa{#1}%
3018   \def\bb1@tempb{#2}%
3019   \ifx\fontspec\undefined
3020     \usepackage{fontspec}%
3021   \fi
3022   \EnableBabelHook{babel-fontspec}% Just calls \bb1@switchfont
3023   \bb1@bb1font}
3024 \newcommand\bb1@bb1font[2][]{% 1=features 2=fontname
3025   \bb1@ifunset{\bb1@tempb family}{\bb1@providefam{\bb1@tempb}}{}}%
3026   % For the default font, just in case:
3027   \bb1@ifunset{\bb1@sys@\languagename}{\bb1@provide@sys{\languagename}}{}}%
```

```

3028 \expandafter\bb1@ifblank\expandafter{\bb1@tempa}%
3029 {\bb1@csarg\edef{\bb1@tempb dflt@}{<{#1}{#2}}}% save bbl@rmdflt@
3030 \bbl@exp{%
3031 \let<\bbl@tempb dflt@\language\language>\<\bbl@tempb dflt@>%
3032 \\\bbl@font@set<\bbl@tempb dflt@\language>%
3033 \<\bbl@tempb default>\<\bbl@tempb family>}}%
3034 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *srt
3035 \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}}%

```

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

```

3036 \def\bbl@providfam#1{%
3037 \bbl@exp{%
3038 \\\newcommand<#1default>{}% Just define it
3039 \\\bbl@add@list\\bbl@font@fams{#1}%
3040 \\\DeclareRobustCommand<#1family>{%
3041 \\\not@math@alphabet<#1family>\relax
3042 \\\fontfamily<#1default>\selectfont}%
3043 \\\DeclareTextFontCommand{\<text#1>}{<#1family>}}

```

The following macro is activated when the hook babel-fontspec is enabled.

```

3044 \def\bbl@switchfont{%
3045 \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3046 \bbl@exp{% eg Arabic -> arabic
3047 \lowercase{\edef\\bbl@tempa{\bbl@cs{sname@\language}}}%
3048 \bbl@foreach\bbl@font@fams{%
3049 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
3050 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
3051 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
3052 {}% 123=F - nothing!
3053 {\bbl@exp{% 3=T - from generic
3054 \global\let<\bbl@##1dflt@\language>%
3055 \<\bbl@##1dflt@>}}}%
3056 {\bbl@exp{% 2=T - from script
3057 \global\let<\bbl@##1dflt@\language>%
3058 \<\bbl@##1dflt@*\bbl@tempa>}}}%
3059 {}% 1=T - language, already defined
3060 \def\bbl@tempa{%
3061 \bbl@warning{The current font is not a standard family:\\%
3062 \fontname\font\\%
3063 Script and Language are not applied. Consider\\%
3064 defining a new family with \string\babelfont.\\%
3065 Reported}}%
3066 \bbl@foreach\bbl@font@fams{% don't gather with prev for
3067 \bbl@ifunset{\bbl@##1dflt@\language}%
3068 {\bbl@cs{famrst@##1}%
3069 \global\bbl@csarg\let{famrst@##1}\relax}%
3070 {\bbl@exp{% order is relevant
3071 \\\bbl@add\\originalTeX{%
3072 \\\bbl@font@rst{\bbl@cs{##1dflt@\language}}%
3073 \<##1default>\<##1family>{##1}}%
3074 \\\bbl@font@set<\bbl@##1dflt@\language>% the main part!
3075 \<##1default>\<##1family>}}}%
3076 \bbl@ifrestoring{}{\bbl@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 \bbl@mapselect because \selectfont is called internally when a font is defined.

```

3077 \def\bbfont@set#1#2#3{% eg \bb@rmdflt@lang \rmdefault \rmfamily
3078 \bb@xin@{<>}{#1}%
3079 \ifin@
3080 \bb@exp{\bbfont@fontspec@set\#1\expandafter\@gobbletwo#1}%
3081 \fi
3082 \bb@exp{%
3083 \def\#2{#1}% eg, \rmdefault{\bb@rmdflt@lang}
3084 \bb@ifsamestring{#2}{\f@family}{\#3\let\bb@tempe\relax}{}}
3085 \def\bbfont@fontspec@set#1#2#3{% eg \bb@rmdflt@lang fnt-opt fnt-nme
3086 \let\bb@tempe\bb@mapselect
3087 \let\bb@mapselect\relax
3088 \bb@exp{% TODO - should be global, but even local does its job
3089 % I'm still not sure -- must investigate
3090 \<keys_if_exist:nnF>{fontspec-opentype}%
3091 {Script/\bb@cs{sname@\language}}%
3092 {\newfontscript{\bb@cs{sname@\language}}}%
3093 {\bb@cs{sotf@\language}}}%
3094 \<keys_if_exist:nnF>{fontspec-opentype}%
3095 {Language/\bb@cs{lname@\language}}%
3096 {\newfontlanguage{\bb@cs{lname@\language}}}%
3097 {\bb@cs{lotf@\language}}}%
3098 \<fontspec_set_family:Nnn>\#1%
3099 {\bb@cs{lsys@\language},#2}{#3}% ie \bb@exp{..}{#3}
3100 \let\bb@mapselect\bb@tempe
3101 \bb@toglobal#1}%

```

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.

```

3102 \def\bbfont@rst#1#2#3#4{%
3103 \bb@csarg\def{famrst@#4}{\bbfont@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```

3104 \def\bbfont@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 :-).

```

3105 \newcommand\babelFSstore[2][{%
3106 \bb@ifblank{#1}%
3107 {\bb@csarg\def{sname@#2}{Latin}}%
3108 {\bb@csarg\def{sname@#2}{#1}}%
3109 \bb@provide@dirs{#2}%
3110 \bb@csarg\ifnum{wdir@#2}>\z@
3111 \let\bb@beforeforeign\leavevmode
3112 \EnableBabelHook{babel-bidi}%
3113 \fi
3114 \bb@foreach{#2}{%
3115 \bb@FSstore{##1}{rm}\rmdefault\bb@save@rmdefault
3116 \bb@FSstore{##1}{sf}\sfdefault\bb@save@sfdefault
3117 \bb@FSstore{##1}{tt}\ttdefault\bb@save@ttdefault}}
3118 \def\bb@FSstore#1#2#3#4{%
3119 \bb@csarg\edef{#2default#1}{#3}%
3120 \expandafter\addto\csname extras#1\endcsname{%
3121 \let#4#3%
3122 \ifx#3\f@family
3123 \edef#3{\csname bbl@#2default#1\endcsname}%
3124 \fontfamily{#3}\selectfont
3125 \else

```

```

3126     \edef#3{\csname bbl@#2default#1\endcsname}%
3127     \fi}%
3128 \expandafter\addto\csname noextras#1\endcsname{%
3129     \ifx#3\fontfamily
3130         \fontfamily{#4}\selectfont
3131     \fi
3132     \let#3#4}}
3133 \let\bbl@langfeatures\@empty
3134 \def\babelFSfeatures{% make sure \fontspec is redefined once
3135     \let\bbl@ori@fontspec\fontspec
3136     \renewcommand\fontspec[1][{}]{%
3137         \bbl@ori@fontspec[\bbl@langfeatures##1]}
3138     \let\babelFSfeatures\bbl@FSfeatures
3139     \babelFSfeatures}
3140 \def\bbl@FSfeatures#1#2{%
3141     \expandafter\addto\csname extras#1\endcsname{%
3142         \babel@save\bbl@langfeatures
3143         \edef\bbl@langfeatures{#2,}}
3144 <</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.

ℒ_{TEX} sets many “codes” just before loading hyphen.cfg. That is not a problem in luatex, but in xetex 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 ℒ_{TEX}. Anyway, for consistency Lua_{TEX} also resets the catcodes.

```

3145 <<{*Restore Unicode catcodes before loading patterns}>> ≡
3146 \begingroup
3147     % Reset chars "80-"C0 to category "other", no case mapping:
3148     \catcode`\@=11 \count@=128
3149     \loop\ifnum\count@<192
3150         \global\uccode\count@=0 \global\lccode\count@=0
3151         \global\catcode\count@=12 \global\sffcode\count@=1000
3152         \advance\count@ by 1 \repeat
3153     % Other:
3154     \def\O ##1 {%
3155         \global\uccode"##1=0 \global\lccode"##1=0
3156         \global\catcode"##1=12 \global\sffcode"##1=1000 }%
3157     % Letter:
3158     \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3159         \global\uccode"##1="##2
3160         \global\lccode"##1="##3
3161         % Uppercase letters have sffcode=999:
3162         \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
3163     % Letter without case mappings:
3164     \def\l ##1 {\L ##1 ##1 ##1 }%
3165     \l 00AA
3166     \L 00B5 039C 00B5
3167     \l 00BA
3168     \O 00D7
3169     \l 00DF
3170     \O 00F7
3171     \L 00FF 0178 00FF

```

```

3172 \endgroup
3173 \input #1\relax
3174 <</Restore Unicode catcodes before loading patterns>>

```

Some more common code.

```

3175 <<(*Footnote changes)>> ≡
3176 \bbl@trace{Bidi footnotes}
3177 \ifx\bbl@beforeforeign\leavevmode
3178 \def\bbl@footnote#1#2#3{%
3179 \ifnextchar[%
3180 {\bbl@footnote@o{#1}{#2}{#3}}%
3181 {\bbl@footnote@x{#1}{#2}{#3}}}
3182 \def\bbl@footnote@x#1#2#3#4{%
3183 \bgroup
3184 \select@language@x{\bbl@main@language}%
3185 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3186 \egroup}
3187 \def\bbl@footnote@o#1#2#3[#4]#5{%
3188 \bgroup
3189 \select@language@x{\bbl@main@language}%
3190 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3191 \egroup}
3192 \def\bbl@footnotetext#1#2#3{%
3193 \ifnextchar[%
3194 {\bbl@footnotetext@o{#1}{#2}{#3}}%
3195 {\bbl@footnotetext@x{#1}{#2}{#3}}}
3196 \def\bbl@footnotetext@x#1#2#3#4{%
3197 \bgroup
3198 \select@language@x{\bbl@main@language}%
3199 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3200 \egroup}
3201 \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3202 \bgroup
3203 \select@language@x{\bbl@main@language}%
3204 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3205 \egroup}
3206 \def\BabelFootnote#1#2#3#4{%
3207 \ifx\bbl@fn@footnote\undefined
3208 \let\bbl@fn@footnote\footnote
3209 \fi
3210 \ifx\bbl@fn@footnotetext\undefined
3211 \let\bbl@fn@footnotetext\footnotetext
3212 \fi
3213 \bbl@ifblank{#2}%
3214 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3215 \@namedef{\bbl@stripslash#1text}%
3216 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3217 {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
3218 \@namedef{\bbl@stripslash#1text}%
3219 {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
3220 \fi
3221 <</Footnote changes>>

```

Now, the code.

```

3222 <(*xetex)>
3223 \def\BabelStringsDefault{unicode}
3224 \let\xebbl@stop\relax
3225 \AddBabelHook{xetex}{encodedcommands}{%
3226 \def\bbl@tempa{#1}%

```

```

3227 \ifx\bb1@tempa\@empty
3228 \XeTeXinputencoding"bytes"%
3229 \else
3230 \XeTeXinputencoding"#1"%
3231 \fi
3232 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3233 \AddBabelHook{xetex}{stopcommands}{%
3234 \xebbl@stop
3235 \let\xebbl@stop\relax}
3236 \def\bb1@intraspace#1 #2 #3\@@{%
3237 \bb1@csarg\gdef{\xeisp@bb1@cs{sbcpr@languagename}}{%
3238 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3239 \def\bb1@intrapenalty#1\@@{%
3240 \bb1@csarg\gdef{\xeipn@bb1@cs{sbcpr@languagename}}{%
3241 {\XeTeXlinebreakpenalty #1\relax}}
3242 \AddBabelHook{xetex}{loadkernel}{%
3243 \langle Restore Unicode catcodes before loading patterns\rangle}
3244 \ifx\DisableBabelHook\undefined\endinput\fi
3245 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
3246 \DisableBabelHook{babel-fontspec}
3247 \langle Font selection\rangle
3248 \input txtbabel.def
3249 \xetex

```

14.2 Layout

In progress.

Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks). At least at this stage, babel will not do it and therefore a package like bidi (by Vafa Khalighi) would be necessary to overcome the limitations of xetex. Any help in making babel and bidi collaborate will be welcome, although the underlying concepts in both packages seem very different. Note also elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bb1@startskip and \bb1@endskip are available to package authors. Thanks to the \TeX expansion mechanism the following constructs are valid: \adim\bb1@startskip, \advance\bb1@startskip\adim, \bb1@startskip\adim.

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

```

3250 \*texxet
3251 \bb1@trace{Redefinitions for bidi layout}
3252 \def\bb1@sspre@caption{%
3253 \bb1@exp{\everyhbox{\bb1@texdir\bb1@cs{wdir@bb1@main@language}}}}
3254 \ifx\bb1@opt@layout\@nnil\endinput\fi % No layout
3255 \def\bb1@startskip{\ifcase\bb1@thepardir\leftskip\else\rightskip\fi}
3256 \def\bb1@endskip{\ifcase\bb1@thepardir\rightskip\else\leftskip\fi}
3257 \ifx\bb1@beforeforeign\leavevmode % A poor test for bidi=
3258 \def\@hangfrom#1{%
3259 \setbox\@tempboxa\hbox{{#1}}%
3260 \hangindent\ifcase\bb1@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3261 \noindent\box\@tempboxa}
3262 \def\raggedright{%
3263 \let\@centercr
3264 \bb1@startskip\z@skip
3265 \@rightskip\@flushglue
3266 \bb1@endskip\@rightskip
3267 \parindent\z@

```



```

3268 \parfillskip\bbl@startskip}
3269 \def\raggedleft{%
3270 \let\\@centercr
3271 \bbl@startskip\@flushglue
3272 \bbl@endskip\z@skip
3273 \parindent\z@
3274 \parfillskip\bbl@endskip}
3275 \fi
3276 \IfBabelLayout{lists}
3277 {\def\list#1#2{%
3278 \ifnum \@listdepth >5\relax
3279 \@toodeep
3280 \else
3281 \global\advance\@listdepth\@ne
3282 \fi
3283 \rightmargin\z@
3284 \listparindent\z@
3285 \itemindent\z@
3286 \csname @list\romannumeral\the\@listdepth\endcsname
3287 \def\@itemlabel{#1}%
3288 \let\makelabel\@mklab
3289 \@nmbrrlistfalse
3290 #2\relax
3291 \@trivlist
3292 \parskip\parsep
3293 \parindent\listparindent
3294 \advance\linewidth-\rightmargin
3295 \advance\linewidth-\leftmargin
3296 \advance\@totalleftmargin
3297 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi
3298 \parshape\@ne\@totalleftmargin\linewidth
3299 \ignorespaces}%
3300 \ifcase\bbl@engine
3301 \def\labelenumii{}\theenumii{%
3302 \def\p@enumiii{\p@enumii}\theenumii{%
3303 \fi
3304 \def\@verbatim{%
3305 \trivlist \item\relax
3306 \if@minipage\else\vskip\parskip\fi
3307 \bbl@startskip\textwidth
3308 \advance\bbl@startskip-\linewidth
3309 \bbl@endskip\z@skip
3310 \parindent\z@
3311 \parfillskip\@flushglue
3312 \parskip\z@skip
3313 \@@par
3314 \language\@nohyphenation
3315 \@tempswafalse
3316 \def\par{%
3317 \if@tempswa
3318 \leavevmode\@null
3319 \@@par\penalty\interlinepenalty
3320 \else
3321 \@tempswatrue
3322 \ifhmode\@@par\penalty\interlinepenalty\fi
3323 \fi}%
3324 \let\do\@makeother \dospecials
3325 \obeylines \verbatim@font \@noligs
3326 \everypar\expandafter{\the\everypar\unpenalty}}

```

```

3327 {}
3328 \IfBabelLayout{contents}
3329 {\def\@dottedtocline#1#2#3#4#5{%
3330   \ifnum#1>\c@tocdepth\else
3331     \vskip \z@ \@plus.2\p@
3332     {\bbl@startskip#2\relax
3333      \bbl@endskip\@tocrmarg
3334      \parfillskip-\bbl@endskip
3335      \parindent#2\relax
3336      \@afterindenttrue
3337      \interlinepenalty\@M
3338      \leavevmode
3339      \@tempdima#3\relax
3340      \advance\bbl@startskip\@tempdima
3341      \null\nobreak\hskip-\bbl@startskip
3342      {#4}\nobreak
3343      \leaders\hbox{%
3344        $\m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
3345        \hfill\nobreak
3346        \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
3347        \par}%
3348   \fi}}
3349 {}
3350 \IfBabelLayout{columns}
3351 {\def\@outputdblcol{%
3352   \if@firstcolumn
3353     \global\@firstcolumnfalse
3354     \global\setbox\@leftcolumn\copy\@outputbox
3355     \splitmaxdepth\maxdimen
3356     \vbadness\maxdimen
3357     \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
3358     \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3359     \toks\@expandafter{\topmark}%
3360     \xdef\@firstcoltopmark{\the\toks@}%
3361     \toks\@expandafter{\splitfirstmark}%
3362     \xdef\@firstcolfirstmark{\the\toks@}%
3363     \ifx\@firstcolfirstmark\@empty
3364       \global\let\@setmarks\relax
3365     \else
3366       \gdef\@setmarks{%
3367         \let\firstmark\@firstcolfirstmark
3368         \let\topmark\@firstcoltopmark}%
3369     \fi
3370   \else
3371     \global\@firstcolumntrue
3372     \setbox\@outputbox\vbox{%
3373       \hb@xt@\textwidth{%
3374         \hskip\columnwidth
3375         \hfil
3376         {\normalcolor\vrule \@width\columnseprule}%
3377         \hfil
3378         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3379         \hskip-\textwidth
3380         \hb@xt@\columnwidth{\box\@outputbox \hss}%
3381         \hskip\columnsep
3382         \hskip\columnwidth}}}%
3383   \@combinedblfloats
3384   \@setmarks
3385   \@outputpage

```

```

3386     \begingroup
3387     \@dblfloatplacement
3388     \@startdblcolumn
3389     \@whiles\if@fcolmade \fi{\@outputpage
3390     \@startdblcolumn}%
3391     \endgroup
3392     \fi}}%
3393     {}
3394     <<Footnote changes>>
3395     \IfBabelLayout{footnotes}%
3396     {\BabelFootnote\footnote\language\{}}%
3397     \BabelFootnote\localfootnote\language\{}}%
3398     \BabelFootnote\mainfootnote\{}}{}
3399     {}

```

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.

```

3400 \IfBabelLayout{counters}%
3401 {\let\bbbl@latinarabic=\@arabic
3402  \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
3403  \let\bbbl@asciroman=\@roman
3404  \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
3405  \let\bbbl@asciiRoman=\@Roman
3406  \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}}{}
3407 </texxet>

```

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, `\bbbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). 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.

```

3408 (*luatex)
3409 \ifx\AddBabelHook\@undefined
3410 \bbl@trace{Read language.dat}
3411 \begingroup
3412 \toks@{}
3413 \count@ \z@ % 0=start, 1=0th, 2=normal
3414 \def\bbl@process@line#1#2 #3 #4 {%
3415   \ifx=#1%
3416     \bbl@process@synonym{#2}%
3417   \else
3418     \bbl@process@language{#1#2}{#3}{#4}%
3419   \fi
3420   \ignorespaces}
3421 \def\bbl@manylang{%
3422   \ifnum\bbl@last>\@ne
3423     \bbl@info{Non-standard hyphenation setup}%
3424   \fi
3425   \let\bbl@manylang\relax}
3426 \def\bbl@process@language#1#2#3{%
3427   \ifcase\count@
3428     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
3429   \or
3430     \count@\tw@
3431   \fi
3432   \ifnum\count@=\tw@
3433     \expandafter\addlanguage\csname l@#1\endcsname
3434     \language\allocationnumber
3435     \chardef\bbl@last\allocationnumber
3436     \bbl@manylang
3437     \let\bbl@elt\relax
3438     \xdef\bbl@languages{%
3439       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3440   \fi
3441   \the\toks@
3442   \toks@{}}
3443 \def\bbl@process@synonym@aux#1#2{%
3444   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3445   \let\bbl@elt\relax
3446   \xdef\bbl@languages{%
3447     \bbl@languages\bbl@elt{#1}{#2}{}}}%
3448 \def\bbl@process@synonym#1{%
3449   \ifcase\count@
3450     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3451   \or
3452     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3453   \else
3454     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3455   \fi}
3456 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
3457   \chardef\l@english\z@
3458   \chardef\l@USenglish\z@
3459   \chardef\bbl@last\z@
3460   \global\namedef\bbl@hyphendata@0{{hyphen.tex}}
3461   \gdef\bbl@languages{%
3462     \bbl@elt{english}{0}{hyphen.tex}}%
3463   \bbl@elt{USenglish}{0}{}%

```

```

3464 \else
3465 \global\let\bbl@languages@format\bbl@languages
3466 \def\bbl@elt#1#2#3#4{% Remove all except language 0
3467 \ifnum#2>\z@\else
3468 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3469 \fi}%
3470 \xdef\bbl@languages{\bbl@languages}%
3471 \fi
3472 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3473 \bbl@languages
3474 \openin1=language.dat
3475 \ifeof1
3476 \bbl@warning{I couldn't find language.dat. No additional\\%
3477 patterns loaded. Reported}%
3478 \else
3479 \loop
3480 \endlinechar\m@ne
3481 \read1 to \bbl@line
3482 \endlinechar`\^^M
3483 \if T\ifeof1F\fi T\relax
3484 \ifx\bbl@line\@empty\else
3485 \edef\bbl@line{\bbl@line\space\space\space}%
3486 \expandafter\bbl@process@line\bbl@line\relax
3487 \fi
3488 \repeat
3489 \fi
3490 \endgroup
3491 \bbl@trace{Macros for reading patterns files}
3492 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
3493 \ifx\babelcatcodetablenum\@undefined
3494 \def\babelcatcodetablenum{5211}
3495 \fi
3496 \def\bbl@luapatterns#1#2{%
3497 \bbl@get@enc#1::\@@@
3498 \setbox\z@\hbox\bgroup
3499 \begingroup
3500 \ifx\catcodetable\@undefined
3501 \let\savecatcodetable\luatexsavecatcodetable
3502 \let\initcatcodetable\luatexinitcatcodetable
3503 \let\catcodetable\luatexcatcodetable
3504 \fi
3505 \savecatcodetable\babelcatcodetablenum\relax
3506 \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3507 \catcodetable\numexpr\babelcatcodetablenum+1\relax
3508 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3509 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
3510 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3511 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3512 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
3513 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
3514 \input #1\relax
3515 \catcodetable\babelcatcodetablenum\relax
3516 \endgroup
3517 \def\bbl@tempa{#2}%
3518 \ifx\bbl@tempa\@empty\else
3519 \input #2\relax
3520 \fi
3521 \egroup}%
3522 \def\bbl@patterns@lua#1{%

```

```

3523 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3524 \csname l@#1\endcsname
3525 \edef\bbl@tempa{#1}%
3526 \else
3527 \csname l@#1:\f@encoding\endcsname
3528 \edef\bbl@tempa{#1:\f@encoding}%
3529 \fi\relax
3530 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3531 \@ifundefined{bbl@hyphendata@the\language}%
3532 {\def\bbl@elt##1##2##3##4{%
3533 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3534 \def\bbl@tempb{##3}%
3535 \ifx\bbl@tempb\@empty\else % if not a synonymous
3536 \def\bbl@tempc{##3}{##4}}%
3537 \fi
3538 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3539 \fi}%
3540 \bbl@languages
3541 \@ifundefined{bbl@hyphendata@the\language}%
3542 {\bbl@info{No hyphenation patterns were set for\%
3543 language '\bbl@tempa'. Reported}}%
3544 {\expandafter\expandafter\expandafter\bbl@luapatterns
3545 \csname bbl@hyphendata@the\language\endcsname}}}%
3546 \endinput\fi
3547 \begingroup
3548 \catcode`\%=12
3549 \catcode`\'=12
3550 \catcode`\ "=12
3551 \catcode`\:=12
3552 \directlua{
3553 Babel = Babel or {}
3554 function Babel.bytes(line)
3555 return line:gsub("(.)",
3556 function (chr) return unicode.utf8.char(string.byte(chr)) end)
3557 end
3558 function Babel.begin_process_input()
3559 if luatexbase and luatexbase.add_to_callback then
3560 luatexbase.add_to_callback('process_input_buffer',
3561 Babel.bytes, 'Babel.bytes')
3562 else
3563 Babel.callback = callback.find('process_input_buffer')
3564 callback.register('process_input_buffer', Babel.bytes)
3565 end
3566 end
3567 function Babel.end_process_input ()
3568 if luatexbase and luatexbase.remove_from_callback then
3569 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3570 else
3571 callback.register('process_input_buffer', Babel.callback)
3572 end
3573 end
3574 function Babel.addpatterns(pp, lg)
3575 local lg = lang.new(lg)
3576 local pats = lang.patterns(lg) or ''
3577 lang.clear_patterns(lg)
3578 for p in pp:gmatch('[^%s]+') do
3579 ss = ''
3580 for i in string.utfcharacters(p:gsub('%d', '')) do
3581 ss = ss .. '%d?' .. i

```

```

3582     end
3583     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
3584     ss = ss:gsub('%.%d%?$', '%%.')
3585     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3586     if n == 0 then
3587         tex.sprint(
3588             [[\string\csname\space bbl@info\endcsname{New pattern: }
3589             .. p .. [{}]])
3590         pats = pats .. ' ' .. p
3591     else
3592         tex.sprint(
3593             [[\string\csname\space bbl@info\endcsname{Renew pattern: }
3594             .. p .. [{}]])
3595     end
3596 end
3597 lang.patterns(lg, pats)
3598 end
3599 }
3600 \endgroup
3601 \def\BabelStringsDefault{unicode}
3602 \let\luabbl@stop\relax
3603 \AddBabelHook{luatex}{encodedcommands}{%
3604     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3605     \ifx\bbl@tempa\bbl@tempb\else
3606         \directlua{Babel.begin_process_input()}%
3607         \def\luabbl@stop{%
3608             \directlua{Babel.end_process_input()}}%
3609     \fi}%
3610 \AddBabelHook{luatex}{stopcommands}{%
3611     \luabbl@stop
3612     \let\luabbl@stop\relax}
3613 \AddBabelHook{luatex}{patterns}{%
3614     \@ifundefined{bbl@hyphendata@the\language}%
3615     {\def\bbl@elt##1##2##3##4{%
3616         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3617         \def\bbl@tempb{##3}%
3618         \ifx\bbl@tempb\@empty\else % if not a synonymous
3619             \def\bbl@tempc{##3}{##4}}%
3620         \fi
3621         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3622         \fi}%
3623     \bbl@languages
3624     \@ifundefined{bbl@hyphendata@the\language}%
3625     {\bbl@info{No hyphenation patterns were set for\%
3626         language '#2'. Reported}}%
3627     {\expandafter\expandafter\expandafter\bbl@luapatterns
3628         \csname bbl@hyphendata@the\language\endcsname}}}%
3629 \@ifundefined{bbl@patterns@}{}%
3630 \begingroup
3631     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
3632     \ifin@else
3633         \ifx\bbl@patterns@\@empty\else
3634             \directlua{ Babel.addpatterns(
3635                 [[\bbl@patterns@]], \number\language) }%
3636             \fi
3637         \@ifundefined{bbl@patterns@#1}%
3638         {\@empty
3639             {\directlua{ Babel.addpatterns(
3640                 [[\space\csname bbl@patterns@#1\endcsname]],

```

```

3641         \number\language) } }%
3642     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3643     \fi
3644     \endgroup}}
3645 \AddBabelHook{luatex}{everylanguage}{%
3646     \def\process@language##1##2##3{%
3647         \def\process@line####1####2 ####3 ####4 {}}
3648 \AddBabelHook{luatex}{loadpatterns}{%
3649     \input #1\relax
3650     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3651         {{#1}}}}
3652 \AddBabelHook{luatex}{loadexceptions}{%
3653     \input #1\relax
3654     \def\bbl@tempb##1##2{{##1}{#1}}%
3655     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3656         {\expandafter\expandafter\expandafter\bbl@tempb
3657         \csname bbl@hyphendata@the\language\endcsname}}

```

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

```

3658 \@onlypreamble\babelpatterns
3659 \AtEndOfPackage{%
3660     \newcommand\babelpatterns[2][\@empty]{%
3661         \ifx\bbl@patterns@\relax
3662             \let\bbl@patterns@\@empty
3663         \fi
3664         \ifx\bbl@pttnlist\@empty\else
3665             \bbl@warning{%
3666                 You must not intermingle \string\selectlanguage\space and\%
3667                 \string\babelpatterns\space or some patterns will not\%
3668                 be taken into account. Reported}%
3669             \fi
3670             \ifx\@empty#1%
3671                 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
3672             \else
3673                 \edef\bbl@tempb{\zap@space#1 \@empty}%
3674                 \bbl@for\bbl@tempa\bbl@tempb{%
3675                     \bbl@fixname\bbl@tempa
3676                     \bbl@iflanguage\bbl@tempa{%
3677                         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3678                             \@ifundefined{bbl@patterns@\bbl@tempa}%
3679                             \@empty
3680                             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3681                             #2}}}%
3682             \fi}}

```

15 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).

```

3683 \def\bbl@intraspace#1 #2 #3\@{
3684     \directlua{
3685         Babel = Babel or {}

```



```

3686     Babel.intraspaces = Babel.intraspaces or {}
3687     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
3688         {b = #1, p = #2, m = #3}
3689     }}
3690 \def\bbl@intrapenalty#1\@{%
3691     \directlua{
3692         Babel = Babel or {}
3693         Babel.intrapenalties = Babel.intrapenalties or {}
3694         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
3695     }}
3696 \begingroup
3697 \catcode`\%=12
3698 \catcode`\^=14
3699 \catcode`\'=12
3700 \catcode`\~=12
3701 \gdef\bbl@seaintraspace{^
3702     \let\bbl@seaintraspace\relax
3703     \directlua{
3704         Babel = Babel or {}
3705         Babel.sea_ranges = Babel.sea_ranges or {}
3706         function Babel.set_chranges (script, chrng)
3707             local c = 0
3708             for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
3709                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
3710                 c = c + 1
3711             end
3712         end
3713         function Babel.sea_disc_to_space (head)
3714             local sea_ranges = Babel.sea_ranges
3715             local last_char = nil
3716             local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
3717             for item in node.traverse(head) do
3718                 local i = item.id
3719                 if i == node.id'glyph' then
3720                     last_char = item
3721                 elseif i == 7 and item.subtype == 3 and last_char
3722                     and last_char.char > 0x0C99 then
3723                     quad = font.getfont(last_char.font).size
3724                     for lg, rg in pairs(sea_ranges) do
3725                         if last_char.char > rg[1] and last_char.char < rg[2] then
3726                             lg = lg:sub(1, 4)
3727                             local intraspace = Babel.intraspaces[lg]
3728                             local intrapenalty = Babel.intrapenalties[lg]
3729                             local n
3730                             if intrapenalty ~= 0 then
3731                                 n = node.new(14, 0)      ^^ penalty
3732                                 n.penalty = intrapenalty
3733                                 node.insert_before(head, item, n)
3734                             end
3735                             n = node.new(12, 13)      ^^ (glue, spaceskip)
3736                             node.setglue(n, intraspace.b * quad,
3737                                 intraspace.p * quad,
3738                                 intraspace.m * quad)
3739                             node.insert_before(head, item, n)
3740                             node.remove(head, item)
3741                         end
3742                     end
3743                 end
3744             end

```

```

3745     end
3746     luatexbase.add_to_callback('hyphenate',
3747         function (head, tail)
3748             lang.hyphenate(head)
3749             Babel.sea_disc_to_space(head)
3750         end,
3751         'Babel.sea_disc_to_space')
3752 }}
3753 \endgroup

```

Common stuff.

```

3754 \AddBabelHook{luatex}{loadkernel}{%
3755 <<Restore Unicode catcodes before loading patterns>>}
3756 \ifx\DisableBabelHook\undefined\endinput\fi
3757 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3758 \DisableBabelHook{babel-fontspec}
3759 <<Font selection>>

```

15.1 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.) and with `bidirectional=basic-r`, 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`.

```

3760 \bbl@trace{Redefinitions for bidi layout}
3761 \ifx\@eqnnum\undefined\else
3762   \ifx\bbl@attr@dir\undefined\else
3763     \edef\@eqnnum{%
3764       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
3765       \unexpanded\expandafter{\@eqnnum}}
3766   \fi
3767 \fi
3768 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
3769 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3770   \def\bbl@nextfake#1{%
3771     \mathdir\bodydir % non-local, use always inside a group!
3772     \bbl@exp{%
3773       #1%           Once entered in math, set boxes to restore values
3774       \everyvbox{%
3775         \the\everyvbox
3776         \bodydir\the\bodydir
3777         \mathdir\the\mathdir
3778         \everyhbox{\the\everyhbox}%
3779         \everyvbox{\the\everyvbox}}%
3780       \everyhbox{%
3781         \the\everyhbox
3782         \bodydir\the\bodydir
3783         \mathdir\the\mathdir
3784         \everyhbox{\the\everyhbox}%
3785         \everyvbox{\the\everyvbox}}}%
3786   \def\hangfrom#1{%
3787     \setbox\@tempboxa\hbox{#1}%

```

```

3788 \hangindent\wd\@tempboxa
3789 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3790 \shapemode\@ne
3791 \fi
3792 \noindent\box\@tempboxa}
3793 \fi
3794 \IfBabelLayout{tabular}
3795 {\def\@tabular{%
3796 \leavevmode\hbox\bgroup\bbl@nextfake$% %$
3797 \let\@acol\@tabacol \let\@classz\@tabclassz
3798 \let\@classiv\@tabclassiv \let\@tabularcr\@tabarray}}
3799 {}
3800 \IfBabelLayout{lists}
3801 {\def\list#1#2{%
3802 \ifnum \@listdepth >5\relax
3803 \@toodeep
3804 \else
3805 \global\advance\@listdepth\@ne
3806 \fi
3807 \rightmargin\z@
3808 \listparindent\z@
3809 \itemindent\z@
3810 \csname @list\romannumeral\the\@listdepth\endcsname
3811 \def\itemlabel{#1}%
3812 \let\makelabel\@mklab
3813 \@nmbrrlistfalse
3814 #2\relax
3815 \@trivlist
3816 \parskip\parsep
3817 \parindent\listparindent
3818 \advance\linewidth -\rightmargin
3819 \advance\linewidth -\leftmargin
3820 \advance\@totalleftmargin \leftmargin
3821 \parshape \@ne
3822 \@totalleftmargin \linewidth
3823 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3824 \shapemode\tw@
3825 \fi
3826 \ignorespaces}}
3827 {}

```

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-r, but there are some additional readjustments for bidi=default.

```

3828 \IfBabelLayout{counters}%
3829 {\def\@textsuperscript#1{% lua has separate settings for math
3830 \m@th
3831 \mathdir\pagedir % required with basic-r; ok with default, too
3832 \ensuremath{^{\mbox{\fontsize\sfontsize\z@ #1}}}}%
3833 \let\bbl@latinarabic=\@arabic
3834 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3835 \@ifpackagewith{babel}{bidi=default}%
3836 {\let\bbl@asciroman=\@roman
3837 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3838 \let\bbl@asciiRoman=\@Roman
3839 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
3840 \def\labelenumii{}\theenumii}%
3841 \def\p@enumiii{\p@enumii}\theenumii}}{}
3842 <<Footnote changes>>

```

```

3843 \IfBabelLayout{footnotes}%
3844   {\BabelFootnote\footnote\language\language}%
3845   \BabelFootnote\localfootnote\language\language}%
3846   \BabelFootnote\mainfootnote\language\language}%
3847   {}

```

Some \TeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

3848 \IfBabelLayout{extras}%
3849   {\def\underline#1{%
3850     \relax
3851     \ifmmode\@underline{#1}%
3852     \else\bb1@nextfake$\@underline{\hbox{#1}}\m@th$\relax\fi}%
3853   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
3854     \if b\expandafter\@car\@series\@nil\boldmath\fi
3855     \babelsublr{%
3856       \LaTeX\kern.15em2\bb1@nextfake$_{\textstyle\varepsilon}$}}}
3857   {}
3858 \luatex

```

15.2 Auto bidi with basic-r

The file `babel-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.

TODO: math mode (as weak L?)

```

3859 (*basic-r)
3860 Babel = Babel or {}
3861
3862 require('babel-bidi.lua')

```

```

3863
3864 local characters = Babel.characters
3865 local ranges = Babel.ranges
3866
3867 local DIR = node.id("dir")
3868
3869 local function dir_mark(head, from, to, outer)
3870   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
3871   local d = node.new(DIR)
3872   d.dir = '+' .. dir
3873   node.insert_before(head, from, d)
3874   d = node.new(DIR)
3875   d.dir = '-' .. dir
3876   node.insert_after(head, to, d)
3877 end
3878
3879 function Babel.pre_otfload_v(head)
3880   -- head = Babel.numbers(head)
3881   head = Babel.bidi(head, true)
3882   return head
3883 end
3884
3885 function Babel.pre_otfload_h(head)
3886   -- head = Babel.numbers(head)
3887   head = Babel.bidi(head, false)
3888   return head
3889 end
3890
3891 function Babel.bidi(head, ispar)
3892   local first_n, last_n          -- first and last char with nums
3893   local last_es                  -- an auxiliary 'last' used with nums
3894   local first_d, last_d          -- first and last char in L/R block
3895   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):

```

3896   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
3897   local strong_lr = (strong == 'l') and 'l' or 'r'
3898   local outer = strong
3899
3900   local new_dir = false
3901   local first_dir = false
3902
3903   local last_lr
3904
3905   local type_n = ''
3906
3907   for item in node.traverse(head) do
3908
3909     -- three cases: glyph, dir, otherwise
3910     if item.id == node.id'glyph'
3911       or (item.id == 7 and item.subtype == 2) then
3912
3913       local itemchar
3914       if item.id == 7 and item.subtype == 2 then
3915         itemchar = item.replace.char
3916       else
3917         itemchar = item.char

```

```

3918     end
3919     local chardata = characters[itemchar]
3920     dir = chardata and chardata.d or nil
3921     if not dir then
3922         for nn, et in ipairs(ranges) do
3923             if itemchar < et[1] then
3924                 break
3925             elseif itemchar <= et[2] then
3926                 dir = et[3]
3927                 break
3928             end
3929         end
3930     end
3931     dir = dir or 'l'

```

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.

```

3932     if new_dir then
3933         attr_dir = 0
3934         for at in node.traverse(item.attr) do
3935             if at.number == luatexbase.registernumber'bbl@attr@dir' then
3936                 attr_dir = at.value % 3
3937             end
3938         end
3939         if attr_dir == 1 then
3940             strong = 'r'
3941         elseif attr_dir == 2 then
3942             strong = 'al'
3943         else
3944             strong = 'l'
3945         end
3946         strong_lr = (strong == 'l') and 'l' or 'r'
3947         outer = strong_lr
3948         new_dir = false
3949     end
3950
3951     if dir == 'nsm' then dir = strong end -- W1

```

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

```

3952     dir_real = dir -- We need dir_real to set strong below
3953     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:

```

3954     if strong == 'al' then
3955         if dir == 'en' then dir = 'an' end -- W2
3956         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
3957         strong_lr = 'r' -- W3
3958     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

3959     elseif item.id == node.id'dir' then
3960         new_dir = true
3961         dir = nil
3962     else
3963         dir = nil -- Not a char

```

3964 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>.

```
3965   if dir == 'en' or dir == 'an' or dir == 'et' then
3966     if dir ~= 'et' then
3967       type_n = dir
3968     end
3969     first_n = first_n or item
3970     last_n = last_es or item
3971     last_es = nil
3972   elseif dir == 'es' and last_n then -- W3+W6
3973     last_es = item
3974   elseif dir == 'cs' then           -- it's right - do nothing
3975   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
3976     if strong_lr == 'r' and type_n ~= '' then
3977       dir_mark(head, first_n, last_n, 'r')
3978     elseif strong_lr == 'l' and first_d and type_n == 'an' then
3979       dir_mark(head, first_n, last_n, 'r')
3980       dir_mark(head, first_d, last_d, outer)
3981       first_d, last_d = nil, nil
3982     elseif strong_lr == 'l' and type_n ~= '' then
3983       last_d = last_n
3984     end
3985     type_n = ''
3986     first_n, last_n = nil, nil
3987   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:

```
3988   if dir == 'l' or dir == 'r' then
3989     if dir ~= outer then
3990       first_d = first_d or item
3991       last_d = item
3992     elseif first_d and dir ~= strong_lr then
3993       dir_mark(head, first_d, last_d, outer)
3994       first_d, last_d = nil, nil
3995     end
3996   end
```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```
3997   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
3998     item.char = characters[item.char] and
3999       characters[item.char].m or item.char
4000   elseif (dir or new_dir) and last_lr ~= item then
4001     local mir = outer .. strong_lr .. (dir or outer)
4002     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
4003       for ch in node.traverse(node.next(last_lr)) do
```

```

4004         if ch == item then break end
4005         if ch.id == node.id'glyph' then
4006             ch.char = characters[ch.char].m or ch.char
4007         end
4008     end
4009 end
4010 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).

```

4011     if dir == 'l' or dir == 'r' then
4012         last_lr = item
4013         strong = dir_real          -- Don't search back - best save now
4014         strong_lr = (strong == 'l') and 'l' or 'r'
4015     elseif new_dir then
4016         last_lr = nil
4017     end
4018 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

4019     if last_lr and outer == 'r' then
4020         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
4021             ch.char = characters[ch.char].m or ch.char
4022         end
4023     end
4024     if first_n then
4025         dir_mark(head, first_n, last_n, outer)
4026     end
4027     if first_d then
4028         dir_mark(head, first_d, last_d, outer)
4029     end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

4030     return node.prev(head) or head
4031 end
4032 </basic-r>

```

And here the Lua code for bidi=basic:

```

4033 (*basic)
4034 Babel = Babel or {}
4035
4036 Babel.fontmap = Babel.fontmap or {}
4037 Babel.fontmap[0] = {}          -- l
4038 Babel.fontmap[1] = {}          -- r
4039 Babel.fontmap[2] = {}          -- al/an
4040
4041 function Babel.pre_otfload_v(head)
4042     -- head = Babel.numbers(head)
4043     head = Babel.bidi(head, true)
4044     return head
4045 end
4046
4047 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
4048     -- head = Babel.numbers(head)
4049     head = Babel.bidi(head, false, dir)
4050     return head
4051 end
4052

```



```

4053 require('babel-bidi.lua')
4054
4055 local characters = Babel.characters
4056 local ranges = Babel.ranges
4057
4058 local DIR = node.id('dir')
4059 local GLYPH = node.id('glyph')
4060
4061 local function insert_implicit(head, state, outer)
4062   local new_state = state
4063   if state.sim and state.eim and state.sim ~= state.eim then
4064     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
4065     local d = node.new(DIR)
4066     d.dir = '+' .. dir
4067     node.insert_before(head, state.sim, d)
4068     local d = node.new(DIR)
4069     d.dir = '-' .. dir
4070     node.insert_after(head, state.eim, d)
4071   end
4072   new_state.sim, new_state.eim = nil, nil
4073   return head, new_state
4074 end
4075
4076 local function insert_numeric(head, state)
4077   local new
4078   local new_state = state
4079   if state.san and state.ean and state.san ~= state.ean then
4080     local d = node.new(DIR)
4081     d.dir = '+TLT'
4082     _, new = node.insert_before(head, state.san, d)
4083     if state.san == state.sim then state.sim = new end
4084     local d = node.new(DIR)
4085     d.dir = '-TLT'
4086     _, new = node.insert_after(head, state.ean, d)
4087     if state.ean == state.eim then state.eim = new end
4088   end
4089   new_state.san, new_state.ean = nil, nil
4090   return head, new_state
4091 end
4092
4093 -- \hbox with an explicit dir can lead to wrong results
4094 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>
4095
4096 function Babel.bidi(head, ispar, hdir)
4097   local d -- d is used mainly for computations in a loop
4098   local prev_d = ''
4099   local new_d = false
4100
4101   local nodes = {}
4102   local outer_first = nil
4103
4104   local glue_d = nil
4105   local glue_i = nil
4106
4107   local has_en = false
4108   local first_et = nil
4109
4110   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
4111

```

```

4112 local save_outter
4113 local temp = node.get_attribute(head, ATDIR)
4114 if temp then
4115     temp = temp % 3
4116     save_outter = (temp == 0 and 'l') or
4117                  (temp == 1 and 'r') or
4118                  (temp == 2 and 'al')
4119 elseif ispar then -- Or error? Shouldn't happen
4120     save_outter = ('TRT' == tex.pardir) and 'r' or 'l'
4121 else
4122     save_outter = ('TRT' == hdir) and 'r' or 'l'
4123 end
4124 local outter = save_outter
4125 local last = outter
4126 -- 'al' is only taken into account in the first, current loop
4127 if save_outter == 'al' then save_outter = 'r' end
4128
4129 local fontmap = Babel.fontmap
4130
4131 for item in node.traverse(head) do
4132
4133     -- In what follows, #node is the last (previous) node, because the
4134     -- current one is not added until we start processing the neutrals.
4135
4136     -- three cases: glyph, dir, otherwise
4137     if item.id == GLYPH
4138         or (item.id == 7 and item.subtype == 2) then
4139
4140         local d_font = nil
4141         local item_r
4142         if item.id == 7 and item.subtype == 2 then
4143             item_r = item.replace -- automatic discs have just 1 glyph
4144         else
4145             item_r = item
4146         end
4147         local chardata = characters[item_r.char]
4148         d = chardata and chardata.d or nil
4149         if not d or d == 'nsm' then
4150             for nn, et in ipairs(ranges) do
4151                 if item_r.char < et[1] then
4152                     break
4153                 elseif item_r.char <= et[2] then
4154                     if not d then d = et[3]
4155                     elseif d == 'nsm' then d_font = et[3]
4156                     end
4157                     break
4158                 end
4159             end
4160         end
4161         d = d or 'l'
4162         d_font = d_font or d
4163
4164         d_font = (d_font == 'l' and 0) or
4165                 (d_font == 'nsm' and 0) or
4166                 (d_font == 'r' and 1) or
4167                 (d_font == 'al' and 2) or
4168                 (d_font == 'an' and 2) or nil
4169         if d_font and fontmap and fontmap[d_font][item_r.font] then
4170             item_r.font = fontmap[d_font][item_r.font]

```

```

4171     end
4172
4173     if new_d then
4174         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4175         attr_d = node.get_attribute(item, ATDIR)
4176         attr_d = attr_d % 3
4177         if attr_d == 1 then
4178             outer_first = 'r'
4179             last = 'r'
4180         elseif attr_d == 2 then
4181             outer_first = 'r'
4182             last = 'al'
4183         else
4184             outer_first = 'l'
4185             last = 'l'
4186         end
4187         outer = last
4188         has_en = false
4189         first_et = nil
4190         new_d = false
4191     end
4192
4193     if glue_d then
4194         if (d == 'l' and 'l' or 'r') ~= glue_d then
4195             table.insert(nodes, {glue_i, 'on', nil})
4196         end
4197         glue_d = nil
4198         glue_i = nil
4199     end
4200
4201     elseif item.id == DIR then
4202         d = nil
4203         new_d = true
4204
4205     elseif item.id == node.id'glue' and item.subtype == 13 then
4206         glue_d = d
4207         glue_i = item
4208         d = nil
4209
4210     else
4211         d = nil
4212     end
4213
4214     -- AL <= EN/ET/ES      -- W2 + W3 + W6
4215     if last == 'al' and d == 'en' then
4216         d = 'an'          -- W3
4217     elseif last == 'al' and (d == 'et' or d == 'es') then
4218         d = 'on'          -- W6
4219     end
4220
4221     -- EN + CS/ES + EN      -- W4
4222     if d == 'en' and #nodes >= 2 then
4223         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4224             and nodes[#nodes-1][2] == 'en' then
4225             nodes[#nodes][2] = 'en'
4226         end
4227     end
4228
4229     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases

```

```

4230   if d == 'an' and #nodes >= 2 then
4231       if (nodes[#nodes][2] == 'cs')
4232           and nodes[#nodes-1][2] == 'an' then
4233               nodes[#nodes][2] = 'an'
4234           end
4235       end
4236
4237       -- ET/EN          -- W5 + W7->l / W6->on
4238       if d == 'et' then
4239           first_et = first_et or (#nodes + 1)
4240       elseif d == 'en' then
4241           has_en = true
4242           first_et = first_et or (#nodes + 1)
4243       elseif first_et then      -- d may be nil here !
4244           if has_en then
4245               if last == 'l' then
4246                   temp = 'l'      -- W7
4247               else
4248                   temp = 'en'    -- W5
4249               end
4250           else
4251               temp = 'on'        -- W6
4252           end
4253           for e = first_et, #nodes do
4254               if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4255           end
4256           first_et = nil
4257           has_en = false
4258       end
4259
4260       if d then
4261           if d == 'al' then
4262               d = 'r'
4263               last = 'al'
4264           elseif d == 'l' or d == 'r' then
4265               last = d
4266           end
4267           prev_d = d
4268           table.insert(nodes, {item, d, outer_first})
4269       end
4270
4271       outer_first = nil
4272
4273   end
4274
4275   -- TODO -- repeated here in case EN/ET is the last node. Find a
4276   -- better way of doing things:
4277   if first_et then      -- dir may be nil here !
4278       if has_en then
4279           if last == 'l' then
4280               temp = 'l'      -- W7
4281           else
4282               temp = 'en'    -- W5
4283           end
4284       else
4285           temp = 'on'        -- W6
4286       end
4287       for e = first_et, #nodes do
4288           if nodes[e][1].id == GLYPH then nodes[e][2] = temp end

```

```

4289     end
4290 end
4291
4292 -- dummy node, to close things
4293 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4294
4295 ----- NEUTRAL -----
4296
4297 outer = save_outer
4298 last = outer
4299
4300 local first_on = nil
4301
4302 for q = 1, #nodes do
4303     local item
4304
4305     local outer_first = nodes[q][3]
4306     outer = outer_first or outer
4307     last = outer_first or last
4308
4309     local d = nodes[q][2]
4310     if d == 'an' or d == 'en' then d = 'r' end
4311     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4312
4313     if d == 'on' then
4314         first_on = first_on or q
4315     elseif first_on then
4316         if last == d then
4317             temp = d
4318         else
4319             temp = outer
4320         end
4321         for r = first_on, q - 1 do
4322             nodes[r][2] = temp
4323             item = nodes[r][1] -- MIRRORING
4324             if item.id == GLYPH and temp == 'r' then
4325                 item.char = characters[item.char].m or item.char
4326             end
4327         end
4328         first_on = nil
4329     end
4330
4331     if d == 'r' or d == 'l' then last = d end
4332 end
4333
4334 ----- IMPLICIT, REORDER -----
4335
4336 outer = save_outer
4337 last = outer
4338
4339 local state = {}
4340 state.has_r = false
4341
4342 for q = 1, #nodes do
4343
4344     local item = nodes[q][1]
4345
4346     outer = nodes[q][3] or outer
4347

```

```

4348   local d = nodes[q][2]
4349
4350   if d == 'nsm' then d = last end           -- W1
4351   if d == 'en' then d = 'an' end
4352   local isdir = (d == 'r' or d == 'l')
4353
4354   if outer == 'l' and d == 'an' then
4355     state.san = state.san or item
4356     state.ean = item
4357   elseif state.san then
4358     head, state = insert_numeric(head, state)
4359   end
4360
4361   if outer == 'l' then
4362     if d == 'an' or d == 'r' then          -- im -> implicit
4363       if d == 'r' then state.has_r = true end
4364       state.sim = state.sim or item
4365       state.eim = item
4366     elseif d == 'l' and state.sim and state.has_r then
4367       head, state = insert_implicit(head, state, outer)
4368     elseif d == 'l' then
4369       state.sim, state.eim, state.has_r = nil, nil, false
4370     end
4371   else
4372     if d == 'an' or d == 'l' then
4373       state.sim = state.sim or item
4374       state.eim = item
4375     elseif d == 'r' and state.sim then
4376       head, state = insert_implicit(head, state, outer)
4377     elseif d == 'r' then
4378       state.sim, state.eim = nil, nil
4379     end
4380   end
4381
4382   if isdir then
4383     last = d           -- Don't search back - best save now
4384   elseif d == 'on' and state.san then
4385     state.san = state.san or item
4386     state.ean = item
4387   end
4388
4389 end
4390
4391 return node.prev(head) or head
4392 end
4393 </basic>

```

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.

```

4394 <*nil>
4395 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
4396 \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.

```
4397 \ifx\l@nohyphenation\@undefined
4398   \nopatterns{nil}
4399   \adddialect\l@nil0
4400 \else
4401   \let\l@nil\l@nohyphenation
4402 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
4403 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 4404 \let\captionnil\@empty
4405 \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.

```
4406 \ldf@finish{nil}
4407 </nil>
```

17 Support for Plain T_EX (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 T_EX-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `locallyhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
4408 (*bplain | blplain)
4409 \catcode`\{=1 % left brace is begin-group character
4410 \catcode`\}=2 % right brace is end-group character
4411 \catcode`\#=6 % hash mark is macro parameter character
```

Now let’s see if a file called `hyphen.cfg` can be found somewhere on T_EX’s input path by trying to open it for reading...

```
4412 \openin 0 hyphen.cfg
```

If the file wasn’t found the following test turns out true.

```
4413 \ifeof0
4414 \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).

```
4415 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
4416 \def\input #1 {%
4417   \let\input\input
4418   \input{hyphen.cfg}
```

Once that's done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
4419 \let\input\undefined
4420 }
4421 \fi
4422 </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`.

```
4423 <bplain>\input{plain.tex}
4424 <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.

```
4425 <bplain>\def\fmtname{babel-plain}
4426 <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 \LaTeX features

The following code duplicates or emulates parts of $\text{\LaTeX} 2_{\epsilon}$ that are needed for `babel`.

```
4427 <*plain>
4428 \def\@empty{}
4429 \def\loadlocalcfg#1{%
4430   \openin0#1.cfg
4431   \ifeof0
4432     \closein0
4433   \else
4434     \closein0
4435     {\immediate\write16{*****}%
4436      \immediate\write16{* Local config file #1.cfg used}%
4437      \immediate\write16{*}%
4438     }
4439     \input #1.cfg\relax
4440   \fi
4441   \@endofldf}
```

17.3 General tools

A number of \LaTeX macro's that are needed later on.

```
4442 \long\def\@firstofone#1{#1}
4443 \long\def\@firstoftwo#1#2{#1}
4444 \long\def\@secondoftwo#1#2{#2}
```



```

4445 \def\@nnil{\@nil}
4446 \def\@gobbletwo#1#2{}
4447 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4448 \def\@star@or@long#1{%
4449   \@ifstar
4450   {\let\l@ngrel@x\relax#1}%
4451   {\let\l@ngrel@x\long#1}}
4452 \let\l@ngrel@x\relax
4453 \def\@car#1#2\@nil{#1}
4454 \def\@cdr#1#2\@nil{#2}
4455 \let\@typeset@protect\relax
4456 \let\protected@edef\edef
4457 \long\def\@gobble#1{}
4458 \edef\@backslashchar{\expandafter\@gobble\string\}
4459 \def\strip@prefix#1>{}
4460 \def\g@addto@macro#1#2{%
4461   \toks@\expandafter{#1#2}%
4462   \xdef#1{\the\toks@}}
4463 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4464 \def\@nameuse#1{\csname #1\endcsname}
4465 \def\@ifundefined#1{%
4466   \expandafter\ifx\csname#1\endcsname\relax
4467     \expandafter\@firstoftwo
4468   \else
4469     \expandafter\@secondoftwo
4470   \fi}
4471 \def\@expandtwoargs#1#2#3{%
4472   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4473 \def\zap@space#1 #2{%
4474   #1%
4475   \ifx#2\@empty\else\expandafter\zap@space\fi
4476   #2}

```

$\text{\LaTeX}_2\epsilon$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

4477 \ifx\@preamblecmds\@undefined
4478   \def\@preamblecmds{}
4479 \fi
4480 \def\@onlypreamble#1{%
4481   \expandafter\gdef\expandafter\@preamblecmds\expandafter{
4482     \@preamblecmds\do#1}}
4483 \@onlypreamble\@onlypreamble

```

Mimick \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

4484 \def\begindocument{%
4485   \@begindocumenthook
4486   \global\let\@begindocumenthook\@undefined
4487   \def\do##1{\global\let##1\@undefined}%
4488   \@preamblecmds
4489   \global\let\do\noexpand}
4490 \ifx\@begindocumenthook\@undefined
4491   \def\@begindocumenthook{}
4492 \fi
4493 \@onlypreamble\@begindocumenthook
4494 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick \LaTeX 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

4495 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
4496 \@onlypreamble\AtEndOfPackage
4497 \def\@endofldf{}
4498 \@onlypreamble\@endofldf
4499 \let\bbl@afterlang\@empty
4500 \chardef\bbl@opt@hyphenmap\z@

```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

4501 \ifx\if@files\@undefined
4502 \expandafter\let\csname if@files\expandafter\endcsname
4503 \csname iffalse\endcsname
4504 \fi

```

Mimick \LaTeX 's commands to define control sequences.

```

4505 \def\newcommand{\@star@or@long\new@command}
4506 \def\new@command#1{%
4507 \testopt{\@newcommand#1}0}
4508 \def\@newcommand#1[#2]{%
4509 \@ifnextchar [{\@xargdef#1[#2]}%
4510 {\@argdef#1[#2]}}
4511 \long\def\@argdef#1[#2]#3{%
4512 \@yargdef#1\@ne{#2}{#3}}
4513 \long\def\@xargdef#1[#2][#3]#4{%
4514 \expandafter\def\expandafter#1\expandafter{%
4515 \expandafter\@protected@testopt\expandafter #1%
4516 \csname\string#1\expandafter\endcsname{#3}}%
4517 \expandafter\@yargdef \csname\string#1\endcsname
4518 \tw@{#2}{#4}}
4519 \long\def\@yargdef#1#2#3{%
4520 \@tempcnta#3\relax
4521 \advance \@tempcnta \@ne
4522 \let\@hash@\relax
4523 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4524 \@tempcntb #2%
4525 \@whilenum \@tempcntb < \@tempcnta
4526 \do{%
4527 \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
4528 \advance \@tempcntb \@ne}%
4529 \let\@hash@###
4530 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
4531 \def\providecommand{\@star@or@long\provide@command}
4532 \def\provide@command#1{%
4533 \begingroup
4534 \escapechar\m@ne\xdef\@gtempa{\string#1}%
4535 \endgroup
4536 \expandafter\@ifundefined\@gtempa
4537 {\def\reserved@a{\new@command#1}}%
4538 {\let\reserved@a\relax
4539 \def\reserved@a{\new@command\reserved@a}}%
4540 \reserved@a}%
4541 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4542 \def\declare@robustcommand#1{%
4543 \edef\reserved@a{\string#1}%
4544 \def\reserved@b{#1}%
4545 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4546 \edef#1{%
4547 \ifx\reserved@a\reserved@b

```

```

4548      \noexpand\x@protect
4549      \noexpand#1%
4550      \fi
4551      \noexpand\protect
4552      \expandafter\noexpand\csname
4553      \expandafter\@gobble\string#1 \endcsname
4554    }%
4555    \expandafter\new@command\csname
4556      \expandafter\@gobble\string#1 \endcsname
4557  }
4558  \def\x@protect#1{%
4559    \ifx\protect\@typeset@protect\else
4560      \@x@protect#1%
4561    \fi
4562  }
4563  \def\@x@protect#1\fi#2#3{%
4564    \fi\protect#1%
4565  }

```

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`.

```

4566 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4567 \ifx\in@\@undefined
4568   \def\in@#1#2{%
4569     \def\in@##1##2##3\in@{%
4570       \ifx\in@##2\in@false\else\in@true\fi}%
4571     \in@##2#1\in@\in@}
4572 \else
4573   \let\bbl@tempa\@empty
4574 \fi
4575 \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).

```

4576 \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.

```

4577 \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.

```

4578 \ifx\@tempcnta\@undefined
4579   \csname newcount\endcsname\@tempcnta\relax
4580 \fi
4581 \ifx\@tempcntb\@undefined
4582   \csname newcount\endcsname\@tempcntb\relax
4583 \fi

```

To prevent wasting two counters in \LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

4584 \ifx\bye\@undefined
4585   \advance\count10 by -2\relax
4586 \fi
4587 \ifx\@ifnextchar\@undefined
4588   \def\@ifnextchar#1#2#3{%
4589     \let\reserved@d=#1%
4590     \def\reserved@a{#2}\def\reserved@b{#3}%
4591     \futurelet\@let@token\@ifnch}
4592 \def\@ifnch{%
4593   \ifx\@let@token\@sptoken
4594     \let\reserved@c\@xifnch
4595   \else
4596     \ifx\@let@token\reserved@d
4597       \let\reserved@c\reserved@a
4598     \else
4599       \let\reserved@c\reserved@b
4600     \fi
4601   \fi
4602   \reserved@c}
4603 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
4604 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
4605 \fi
4606 \def\@testopt#1#2{%
4607   \@ifnextchar[{#1}]{#1[#2]}}
4608 \def\@protected@testopt#1{%
4609   \ifx\protect\@typeset@protect
4610     \expandafter\@testopt
4611   \else
4612     \@x@protect#1%
4613   \fi}
4614 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4615   #2\relax}\fi}
4616 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4617   \else\expandafter\@gobble\fi{#1}}

```

17.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain \TeX environment.

```

4618 \def\DeclareTextCommand{%
4619   \@dec@text@cmd\providecommand
4620 }
4621 \def\ProvideTextCommand{%
4622   \@dec@text@cmd\providecommand
4623 }
4624 \def\DeclareTextSymbol#1#2#3{%
4625   \@dec@text@cmd\chardef#1{#2}#3\relax
4626 }
4627 \def\@dec@text@cmd#1#2#3{%
4628   \expandafter\def\expandafter#2%
4629     \expandafter{%
4630       \csname#3-cmd\expandafter\endcsname
4631       \expandafter#2%
4632       \csname#3\string#2\endcsname
4633     }%
4634   \let\@ifdefinable\@rc@ifdefinable
4635   \expandafter#1\csname#3\string#2\endcsname
4636 }
4637 \def\@current@cmd#1{%

```

```

4638 \ifx\protect\@typeset@protect\else
4639 \noexpand#1\expandafter\@gobble
4640 \fi
4641 }
4642 \def\@changed@cmd#1#2{%
4643 \ifx\protect\@typeset@protect
4644 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4645 \expandafter\ifx\csname ?\string#1\endcsname\relax
4646 \expandafter\def\csname ?\string#1\endcsname{%
4647 \@changed@x@err{#1}%
4648 }%
4649 \fi
4650 \global\expandafter\let
4651 \csname\cf@encoding \string#1\expandafter\endcsname
4652 \csname ?\string#1\endcsname
4653 \fi
4654 \csname\cf@encoding\string#1%
4655 \expandafter\endcsname
4656 \else
4657 \noexpand#1%
4658 \fi
4659 }
4660 \def\@changed@x@err#1{%
4661 \errhelp{Your command will be ignored, type <return> to proceed}%
4662 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4663 \def\DeclareTextCommandDefault#1{%
4664 \DeclareTextCommand#1?%
4665 }
4666 \def\ProvideTextCommandDefault#1{%
4667 \ProvideTextCommand#1?%
4668 }
4669 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
4670 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4671 \def\DeclareTextAccent#1#2#3{%
4672 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
4673 }
4674 \def\DeclareTextCompositeCommand#1#2#3#4{%
4675 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
4676 \edef\reserved@b{\string##1}%
4677 \edef\reserved@c{%
4678 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
4679 \ifx\reserved@b\reserved@c
4680 \expandafter\expandafter\expandafter\ifx
4681 \expandafter\@car\reserved@a\relax\relax\nil
4682 \@text@composite
4683 \else
4684 \edef\reserved@b##1{%
4685 \def\expandafter\noexpand
4686 \csname#2\string#1\endcsname###1{%
4687 \noexpand\@text@composite
4688 \expandafter\noexpand\csname#2\string#1\endcsname
4689 ###1\noexpand\@empty\noexpand\@text@composite
4690 {##1}%
4691 }%
4692 }%
4693 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
4694 \fi
4695 \expandafter\def\csname\expandafter\string\csname
4696 #2\endcsname\string#1-\string#3\endcsname{#4}

```

```

4697 \else
4698 \errhelp{Your command will be ignored, type <return> to proceed}%
4699 \errmessage{\string\DeclareTextCompositeCommand\space used on
4700 inappropriate command \protect#1}
4701 \fi
4702 }
4703 \def\@text@composite#1#2#3\@text@composite{%
4704 \expandafter\@text@composite@x
4705 \csname\string#1-\string#2\endcsname
4706 }
4707 \def\@text@composite@x#1#2{%
4708 \ifx#1\relax
4709 #2%
4710 \else
4711 #1%
4712 \fi
4713 }
4714 %
4715 \def\@strip@args#1:#2-#3\@strip@args{#2}
4716 \def\DeclareTextComposite#1#2#3#4{%
4717 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
4718 \bgroup
4719 \lcode` \@=#4%
4720 \lowercase{%
4721 \egroup
4722 \reserved@a @%
4723 }%
4724 }
4725 %
4726 \def\UseTextSymbol#1#2{%
4727 % \let\@curr@enc\cf@encoding
4728 % \@use@text@encoding{#1}%
4729 #2%
4730 % \@use@text@encoding\@curr@enc
4731 }
4732 \def\UseTextAccent#1#2#3{%
4733 % \let\@curr@enc\cf@encoding
4734 % \@use@text@encoding{#1}%
4735 % #2{\@use@text@encoding\@curr@enc\selectfont#3}%
4736 % \@use@text@encoding\@curr@enc
4737 }
4738 \def\@use@text@encoding#1{%
4739 % \edef\f@encoding{#1}%
4740 % \xdef\font@name{%
4741 % \csname\curr@fontshape/\f@size\endcsname
4742 % }%
4743 % \pickup@font
4744 % \font@name
4745 % \@@enc@update
4746 }
4747 \def\DeclareTextSymbolDefault#1#2{%
4748 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
4749 }
4750 \def\DeclareTextAccentDefault#1#2{%
4751 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
4752 }
4753 \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.

```
4754 \DeclareTextAccent{"}{OT1}{127}
4755 \DeclareTextAccent{'}{OT1}{19}
4756 \DeclareTextAccent{^}{OT1}{94}
4757 \DeclareTextAccent{`}{OT1}{18}
4758 \DeclareTextAccent{~}{OT1}{126}
```

The following control sequences are used in `babel.def` but are not defined for PLAIN \TeX .

```
4759 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
4760 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
4761 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
4762 \DeclareTextSymbol{\textquoteright}{OT1}{`}
4763 \DeclareTextSymbol{\i}{OT1}{16}
4764 \DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the \TeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \TeX has, we just `\let` it to `\sevenrm`.

```
4765 \ifx\scriptsize\@undefined
4766   \let\scriptsize\sevenrm
4767 \fi
4768 </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, *\TeX , 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 \TeX styles*, *TUGboat 10 (1989) #3*, p. 401–406.
- [8] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [9] Joachim Schrod, *International \TeX is ready to use*, *TUGboat 11 (1990) #1*, p. 87–90.
- [10] Apostolos Syropoulos, Antonis Tsolomititis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, p. 301–373.